

Lập trình hướng đối tượng và C++

Bài 7: Dẫn xuất và kế thừa

TS. Nguyễn Hiếu Cường

Bộ môn CNPM, Khoa CNTT, Trường Đại học GTVT

Email: cuonggt@gmail.com

Nội dung chính

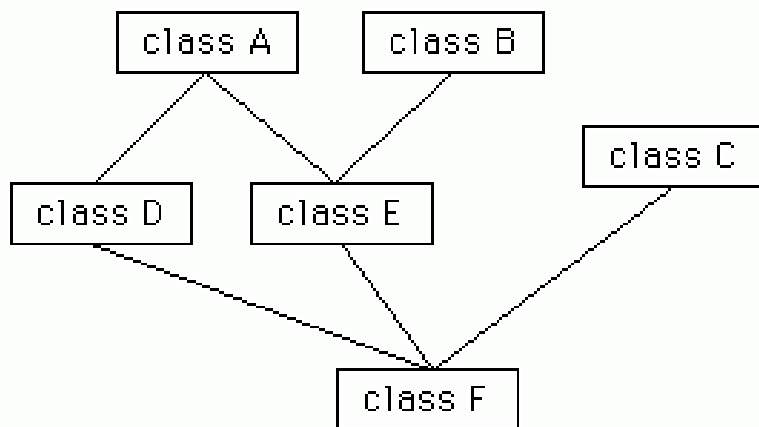
1. Giới thiệu môn học
2. Các khái niệm cơ bản
3. Hàm trong C++
4. Lớp và đối tượng
5. Định nghĩa chồng toán tử
6. Hàm tạo và hàm huỷ
- 7. Dẫn xuất và kế thừa**
8. Tương ứng bội
9. Khuôn hình

Khái niệm kế thừa

- Mục đích chính của kế thừa là nâng cao khả năng **sử dụng lại**
- Trong lập trình cấu trúc ta có cơ chế sử dụng lại không?
- Sử dụng lại trong lập trình hướng đối tượng tốt hơn không?

Tính chất của kế thừa

- Kế thừa nhằm xây dựng một lớp mới từ các lớp đã có
 - Lớp ban đầu: lớp cơ sở (lớp cha, base class, super class)
 - Lớp kế thừa: lớp dẫn xuất (lớp con, derived class, sub class)
- Một lớp có thể là cơ sở của một hoặc nhiều lớp khác
- Một lớp có thể được kế thừa từ một hoặc nhiều lớp khác



Tính chất của kế thừa

- Khi kế thừa thì đối tượng của lớp con sẽ kế thừa gì từ lớp cha?
- Kế thừa tất cả các thuộc tính
 - Đối tượng lớp con có các thuộc tính giống đối tượng lớp cha, ngoài ra có các thuộc tính của riêng nó
- Kế thừa tất cả các hành vi
 - Đối tượng lớp cha làm được gì thì đối tượng lớp con cũng làm được như vậy, ngoài ra có các hành vi của riêng nó

Ví dụ

- Kế thừa có dạng quan hệ “là một” hoặc “là một loại”

A car **is a** vehicle.

A bus **is a** vehicle.

~~A car **is a** bus.~~

An employee **is a** person.

A customer **is a** person.

~~A customer **is a** shopping cart.~~

A checking account **is a kind of** bank account.

A savings account **is a type of** bank account.

A Bentley Continental GT **is a** car **is a** vehicle.

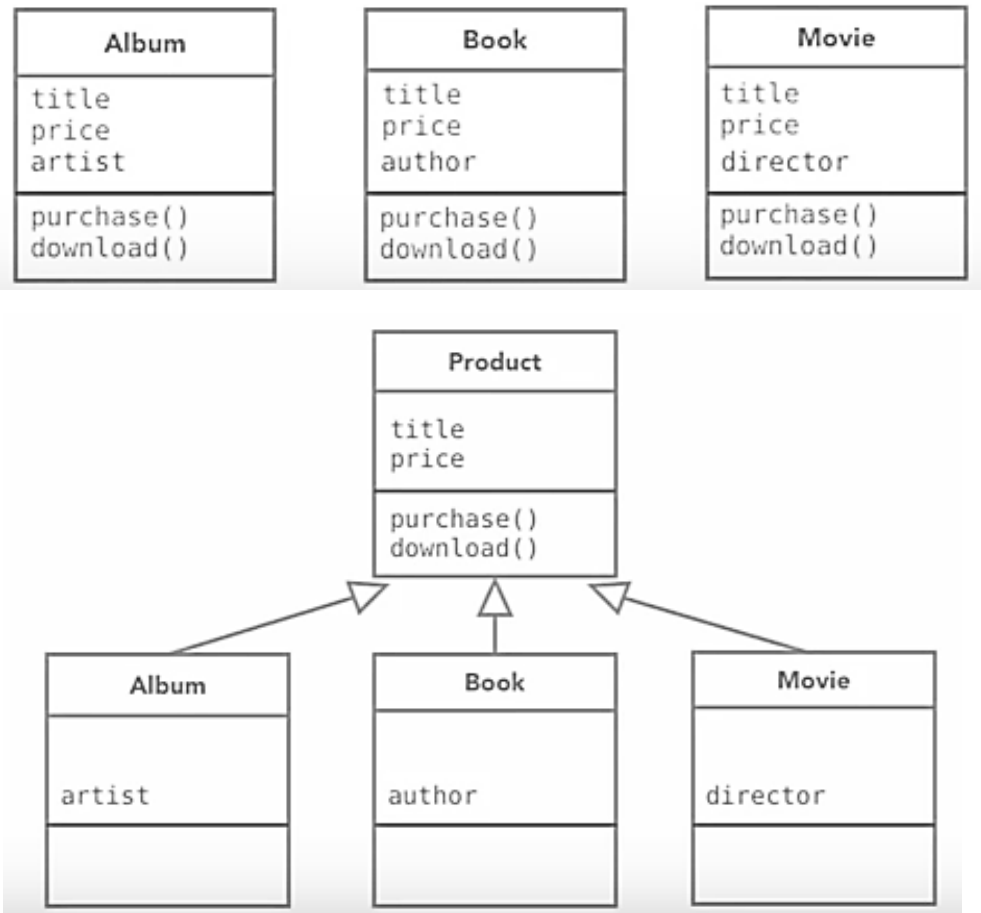
A Pomeranian **is a** dog **is a** mammal **is an** animal.

Tác dụng của kế thừa

- Nâng cao khả năng sử dụng lại
- Thiết kế các lớp hợp lý hơn (ví dụ: tránh trùng lặp)
- Các cách tiếp cận khi thiết kế các lớp kế thừa
 - Top-down: Từ một lớp, chi tiết hóa thành các lớp con
 - Bottom-up: Xây dựng lớp cơ sở từ các đặc tính chung của một số lớp


Ví dụ

GiaoDich



Xây dựng lớp dẫn xuất

- Xây dựng lớp D kế thừa từ lớp A



```
class D : public A
{
    // Khai báo các thuộc tính
public:
    // Khai báo các phương thức
};
```

Phạm vi trong kế thừa

- Phạm vi truy nhập
 - private
 - + public
 - # protected
- Lớp D kế thừa lớp A thì đối tượng lớp D bao gồm
 - Các thành phần (dữ liệu và phương thức) được kế thừa từ lớp A
 - Các thành phần của riêng của lớp D
- Có thể truy nhập các thành phần của A từ lớp D?
 - Có thể nếu dữ liệu của A là public hoặc protected
 - Không thể nếu dữ liệu của A là private
 - Cần thông qua các phương thức của A

Các kiểu kế thừa

Inheritance Type	Base class member access	Derived class member access
public	public	public
	protected	protected
	private	inaccessible
protected	public	protected
	protected	protected
	private	inaccessible
private	public	private
	protected	private
	private	inaccessible

Ví dụ

```
class DIEM
{
    double x, y;
public:
    DIEM() { x = y = 0.0; }
    DIEM(double x1, double y1) { x = x1; y = y1; }
    void hien() { cout << "\nx= " << x << "   y= " << y; }
};

class HINH_TRON : public DIEM
{
    double r;
public:
    HINH_TRON() { r = 0.0; }
    HINH_TRON(double x1, double y1, double r1): DIEM(x1,y1)
    {
        r = r1;
    }
    double getR() { return r; }
};
```

Trong thành phần của đối tượng HINH_TRON có thành phần riêng kế thừa từ DIEM, nên để khởi tạo nó cần phải dùng hàm tạo của lớp DIEM

Tại sao cần gọi hàm tạo của lớp DIEM?

Ví dụ (tiếp)

```
void main()
{
    HINH_TRON h(2.5, 3.5, 8);
    cout << "\nHinh tron co tam: ";
    h.hien();
    cout << "\nCo ban kinh= " << h.getR();
}
```

Hàm tạo trong lớp dẫn xuất

- Tạo một đối tượng lớp dẫn xuất
 - Hàm tạo của lớp cơ sở → Hàm tạo của lớp dẫn xuất
- Hủy một đối tượng lớp dẫn xuất
 - Hàm hủy của lớp dẫn xuất → Hàm hủy của lớp cơ sở
- Thực hiện hàm tạo trong lớp dẫn xuất
 - *Thuộc tính kế thừa* từ lớp cơ sở bằng hàm tạo lớp cơ sở
 - *Đối tượng thành phần* (của lớp khác) bằng hàm tạo của các lớp đó
 - *Thuộc tính mới* của lớp dẫn xuất

Ví dụ: Lớp GIAO_VIEN

```
class MON_HOC
{
    char *monhoc;    // môn học
    int st;          // số tiết
public:
    MON_HOC() { ... }
    MON_HOC(char *monhoc1, int st1){...}
    ~MON_HOC() { ... }
    void xuat() { ... }
};
```

```
class NGUOI
{
    char *ht;        // họ tên
    int ns;          // năm sinh
public:
    NGUOI() { ... }
    NGUOI(char *ht1, int ns1) { ... }
    ~NGUOI() { ... }
    void xuat() { ... }
};
```

```
class GIAO_VIEN: public NGUOI
{
    char *bomon;
    MON_HOC mh;
public:
    GIAO_VIEN(): mh(), NGUOI() { ... }

    GIAO_VIEN(char *ht1,int ns1, char
        *monhoc1, int st1, char *bomon1):
        NGUOI(ht1,ns1), mh(monhoc1, st1)
    {
        int n = strlen(bomon1);
        bomon = new char[n+1];
        strcpy(bomon,bomon1);
    }

    ~GIAO_VIEN() {
        if (bomon!=NULL) delete bomon;
    }
    void xuat() { ... }
};
```

Ví dụ

- Hãy định nghĩa một lớp Shape trong đó có một hàm tạo có hai đối. Dẫn xuất từ lớp Shape hai lớp con là Triangle và Rectangle trong mỗi lớp đó có phương thức tính diện tích area().

Xây dựng hàm main(), trong đó khai báo hai biến đối tượng lớp Triangle và Rectangle, sau đó gọi hàm area() để tính diện tích của mỗi hình.

```
#include<iostream>
using namespace std;
class Shape {
protected:
    float width, height;
public:
    void set_data (float a, float b) {
        width = a; height = b;
    }
};
class Rectangle: public Shape {
public:
    float area () {
        return (width * height);
    }
};
```

```
class Triangle: public Shape
{
public:
    float area () {
        return (width * height / 2);
    }
};
int main ()
{
    Rectangle rect;
    Triangle tri;
    rect.set_data (5,3);
    tri.set_data (2,5);
    cout << rect.area() << endl;
    cout << tri.area() << endl;
}
```

Nhận xét?

- Trong chương trình trên:
 - Tại sao có thể truy nhập trực tiếp width và height từ lớp dẫn xuất?
- Hãy viết chương trình tương tự, trong đó:
 - Nếu không dùng hàm set_data() mà dùng hàm tạo có 2 đối?
 - Nếu width và height không phải là protected mà là private trong lớp Shape? [B6_hinhvuong.cpp]

Ví dụ

Để quản lý 2 công ty Phần mềm và Vận tải, cần tạo các lớp:

Lớp CTY gồm các thuộc tính (private): ten (tên công ty), ntl (năm thành lập)

Lớp CTYPM dẫn xuất từ lớp CTY, gồm các thuộc tính (private): sltv (số lập trình viên)

Lớp CTYVT dẫn xuất từ lớp CTY, gồm các thuộc tính (private): soto (số ô tô)

Viết hàm main() để thực hiện các yêu cầu sau:

- + Nhập danh sách n công ty phần mềm và m công ty vận tải.
- + In tên các công ty phần mềm thành lập sau năm 2000 và có trên 20 lập trình viên.
- + In các công ty vận tải thành lập trước năm 2000 và có không quá 10 ô tô.

Hàm tạo sao chép trong lớp dẫn xuất

- Mỗi lớp đều có hàm tạo sao chép mặc định
- Khi nào cần xây dựng hàm tạo sao chép?
 - Khi lớp dẫn xuất có thuộc tính (kể cả thuộc tính được thừa kế từ các lớp cơ sở) là biến con trỏ hoặc biến tham chiếu
- Khi xây dựng hàm tạo sao chép cho lớp dẫn xuất
 - Cần gọi tới hàm tạo sao chép của các lớp cơ sở và các lớp thành phần

Ví dụ

```
class A {  
    int a;  
    char* str;  
public:  
    A(A& h) {  
        a= h.a;  
        str= strdup(h.str);  
    }  
};
```

```
class B: public A {  
    int b;  
    char* str;  
public:
```

```
    B(B& h):A(h) {  
        b= h.b;  
        str= strdup(h.str);  
    }
```

```
};
```

```
void main()
```

```
{
```

```
    B h1;
```

```
    ...
```

```
    B h2(h1);
```

```
}
```

→ `str= strdup(h.str);`

tương đương:

```
str= malloc(strlen(h.str)+1);
```

```
strcpy(str, h.str);
```

Toán tử gán cho lớp dẫn xuất

- Mọi lớp đều có toán tử gán mặc định
- Khi nào cần xây dựng toán tử gán?
 - Khi trong lớp dẫn xuất có thuộc tính là biến con trỏ hoặc tham chiếu (kể cả thuộc tính thừa kế từ các lớp cơ sở)
- Khi xây dựng toán tử gán cho lớp dẫn xuất cần gọi toán tử gán của các lớp cơ sở và toán tử gán của các lớp thành phần
 - Cần ép kiểu theo mẫu sau (giả sử A là lớp cơ sở)

`A(*this) = A::operator=(h) ;`

Ví dụ

```
class A {
    int a;
    char* str;
public:
    A& operator=(A& h) {
        a = h.a; str = strdup(h.str);
        return h;
    }
};

class B: public A {
    int b;
    char* str;
public:
    B& operator=(B& h) {
        A(*this) = A::operator=(h);
        b = h.b; str = strdup(h.str);
        return h;
    }
};
```

```
void main()
{
    B h1, h2;
    ...
    h2 = h1;
}
```

Xây dựng hàm hủy trong lớp dẫn xuất

- Khi đối tượng lớp dẫn xuất bị hủy:
 - Hàm hủy của lớp dẫn xuất hủy các thành phần riêng của lớp này
 - Hàm hủy lớp cơ sở hủy các thành phần kế thừa từ lớp cơ sở
 - Hàm hủy của các lớp thành phần hủy các đối tượng thành phần

Ví dụ (kết quả của chương trình?)

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

class B: public A {
public:
    B() { cout<<"B"; }
    ~B() { cout<<"~B"; }
};

class C {
    A a;
public:
    C() { cout<<"C"; }
    ~C() { cout<<"~C"; }
};
```

```
class D: public C {
    B b;
public:
    D() { cout<<"D"; }
    ~D() { cout<<"~D"; }
};

int main()
{
    D d;
    cout<<endl;
}
```

Ví dụ (hàm tạo, hàm hủy)

```
class MON_HOC
{
    char *monhoc;    // môn học
    int st;           // số tiết
public:
    MON_HOC() {monhoc= NULL; st=0;}
    MON_HOC(char *monhoc1, int st1)
    {
        int n = strlen(monhoc1);
        monhoc = new char[n+1];
        strcpy(monhoc, monhoc1);
        st = st1;
    }
    ~MON_HOC()
    {
        if (monhoc!=NULL) {
            delete monhoc; st=0; }
    }
    void xuat() {
        cout<<monhoc <<" " << st <<endl;}
};
```

```
class NGUOI
{
    char *ht;        // họ tên
    int ns;          // năm sinh
public:
    NGUOI() { ht=NULL; ns=0; }
    NGUOI(char *ht1, int ns1)
    {
        int n = strlen(ht1);
        ht = new char[n+1];
        strcpy(ht, ht1);
        ns=ns1;
    }
    ~NGUOI()
    {
        if (ht!=NULL) {
            delete ht; ns=0; }
    }
    void xuat() {
        cout << ht <<" " <<ns<<endl;}
};
```

Ví dụ (tiếp)

```
class GIAO_VIEN : public NGUOI {
    char *bomon;
    MON_HOC mh;
public:
    GIAO_VIEN(): mh(), NGUOI() {
        bomon=NULL; }
    GIAO_VIEN(char *ht1,int ns1, char
        *monhoc1, int st1, char *bomon1):
        NGUOI(ht1,ns1), mh(monhoc1, st1)
    {
        int n = strlen(bomon1);
        bomon = new char[n+1];
        strcpy(bomon,bomon1);
    }
    ~GIAO_VIEN() {
        if (bomon!=NULL) delete bomon; }
    void xuat() {
        NGUOI::xuat();
        cout <<endl<< bomon;
        mh.xuat();
    }
};
```

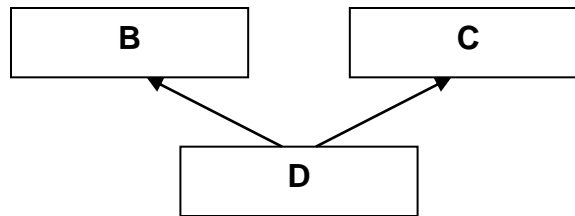
```
void main()
{
    // Goi toi cac ham tao khong co doi
    GIAO_VIEN g1;
    GIAO_VIEN *g2;

    //Goi toi cac ham tao co doi
    g2= new GIAO_VIEN("PHAM VAN AT",
        1945, "CNPM", 60, "CNPM");

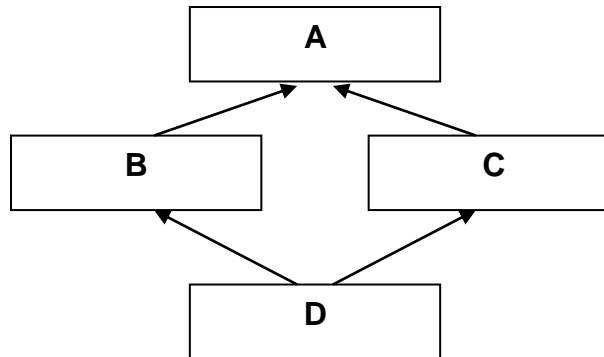
    g2->xuat();
}
```

Kế thừa nhiều mức và sự trùng tên

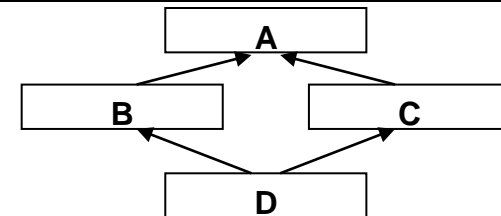
- Đa kế thừa



- Chuyện gì sẽ xảy ra trong trường hợp sau?



Ví dụ (kế thừa nhiều mức)



```
class A {
    public: int a;
};

class B: public A {
    public: int b;
};

class C: public A {
    public: int c;
};

class D: public B, public C
{
    public: int d;
};
```

```
int main()
{
    D h;
    h.d= 4;
    cout<<h.d<<endl;    // 4

    h.c= 5;
    cout<<h.c<<endl;    // 5

    h.b= 6;
    cout<<h.b<<endl;    // 6

    h.B::a= 7;
    cout<<h.B::a<<endl; // 7

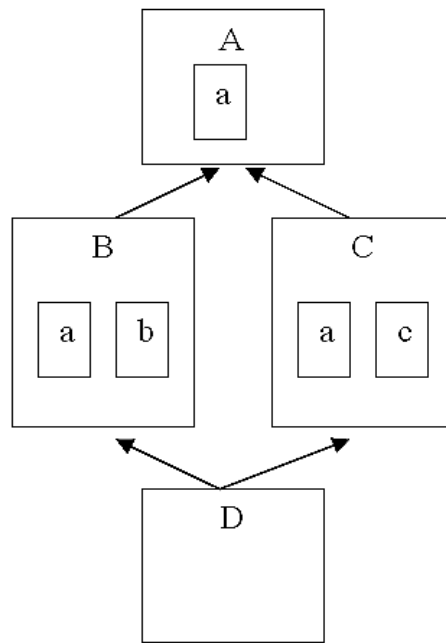
    cout<<h.a;           // lỗi!
}
```

Ví dụ (tiếp)

- Lệnh `cout<<h.a` gây lỗi

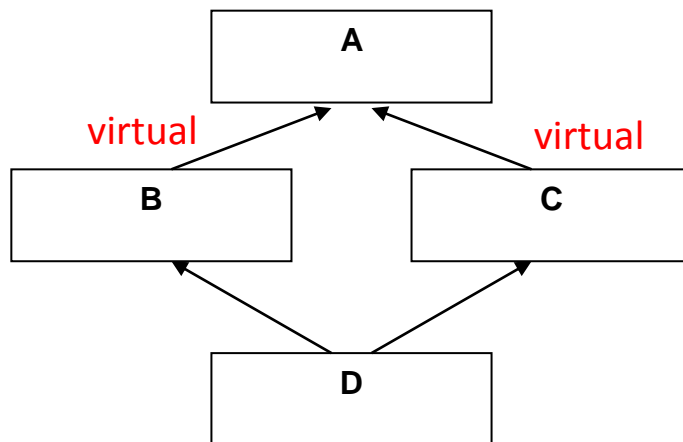
Request for member 'a' is ambiguous

Trình biên dịch không thể biết được thuộc tính 'a' của D sẽ kế thừa thông qua B hay thông qua C



Cách xử lý đa kế thừa

- Biến lớp A trở thành lớp cơ sở ảo
 - Các lớp cơ sở ảo sẽ được kết hợp để tạo một lớp cơ sở duy nhất
 - Lớp D sẽ chỉ có một lớp cơ sở A duy nhất
- Để thực hiện được điều trên:
 - Thêm từ khóa **virtual** khi khai báo B và C



Ví dụ (lớp cơ sở ảo)

```
// Đã sửa A thành lớp cơ sở ảo
#include <iostream>
class A {
public:
    int a;
};

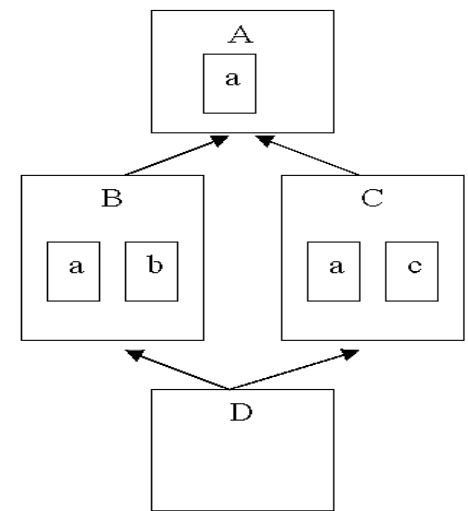
class B: virtual public A {
public: int b;
};

class C: virtual public A {
public: int c;
};

class D: public B, public C {
public: int d;
};
```

```
void main()
{
    D h;
    h.d=4 ;
    cout<<h.d<<endl;      // 4
    h.c=5;
    cout<<h.c<<endl;      // 5
    h.b=6 ;
    cout<<h.b<<endl;      // 6
    h.B::a= 7;
    cout<<h.B::a<<endl;    // 7
    cout<<h.C::a;          // 7

    cout<<h.a;             // 7
}
```



Bài tập

1. Cho biết kết quả?

```
class A {
public:
    A() { cout << "Start A ";}
    ~A() { cout << "End A ";}
};

class B : public A {
public:
    B() { cout << "Start B ";}
    ~B() { cout << "End B ";}
};

class C {
public:
    B b;
    C() { cout << "Start C ";}
};

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

Start A Start B Start C End B End A

```
class A {
public:
    int a;   // chu y: a la public
    A() { a = 1;}
};

class B : public A {
public:
    int b;
    B() { b = 0; a = 0;}
};

int main() {
    A ob1;
    B ob2;
    cout << ob1.a;
    cout << ob2.a << ob2.b;
}
```

1 0 0

Bài tập

2. Cho biết kết quả?

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

class B: public A {
public:
    B() { cout<<"B"; }
    ~B() { cout<<"~B"; }
};

class C {
    A a;
public:
    C() { cout<<"C"; }
    ~C() { cout<<"~C"; }
};
```

```
class D: public C {
    B b;
public:
    D() { cout<<"D"; }
    ~D() { cout<<"~D"; }
};

void main()
{
    D d;
    cout<<endl;
}
```

A C A B D ~D ~B ~A ~C ~A

Bài tập

3. Chỉ ra lỗi sai trong đoạn chương trình sau:

```
class A {
public:
    void func();
};
class B: private class A { };
...
int main() {
    A a;
    B b;
    a.func();
    b.func();           // error
    A* pa = &b;
    B* pb = &a;         // error
}
```

Ví dụ

Để quản lý 2 công ty Phần mềm và Vận tải, cần tạo các lớp:

Lớp CTY gồm các thuộc tính (private): ten (tên công ty), ntl (năm thành lập)

Lớp CTYPM dẫn xuất từ lớp CTY, gồm các thuộc tính (private): sltv (số lập trình viên)

Lớp CTYVT dẫn xuất từ lớp CTY, gồm các thuộc tính (private): soto (số ô tô)

Viết hàm main() để thực hiện các yêu cầu sau:

- + Nhập danh sách n công ty phần mềm và m công ty vận tải.
- + In tên các công ty phần mềm thành lập sau năm 2000 và có trên 20 lập trình viên.
- + In các công ty vận tải thành lập trước năm 2000 và có không quá 10 ô tô.

Bài tập

4. Xây dựng lớp Nguoi:

- Thuộc tính (private): họ tên, tuổi
- Phương thức: nhập, xuất, các phương thức khác

Xây dựng lớp Cauthu dẫn xuất của lớp Nguoi, trong đó bổ sung thêm các thuộc tính sbt (số bàn thắng), sptd (số phút thi đấu).

Viết hàm main() thực hiện:

- Nhập n cầu thủ.
- Tìm cầu thủ lớn tuổi nhất.
- Xác định số tiền thưởng của từng cầu thủ biết rằng cầu thủ thi đấu tổng cộng trên 300 phút thì được thưởng 10 triệu, và cứ ghi được 1 bàn thắng thì được thưởng thêm 5 triệu đồng.

Bài tập

5. Xây dựng lớp Nguoi:

- Thuộc tính: họ tên, tuổi
- Phương thức: Nhập, xuất và các phương thức khác

Xây dựng lớp QLNV(Quản lý nhân viên) dẫn xuất từ lớp Nguoi

- Thuộc tính: bổ sung thêm snct (số năm công tác), hsl (hệ số lương)
- Phương thức: tính tiền lương, biết: $tienluong = lcb * hsl + phucap$, trong đó: lcb là 1.5 triệu, $phucap = 0.2 \text{ triệu} * snct$.

Viết hàm main() thực hiện:

- Nhập vào n nhân viên.
- Tìm nhân viên có số năm công tác nhiều nhất.
- In danh sách nhân viên theo thứ tự lương từ cao đến thấp.