NGÔN NGỮ LẬP TRÌNH C++

Chương 6: Kế thừa và đa hình trong C++

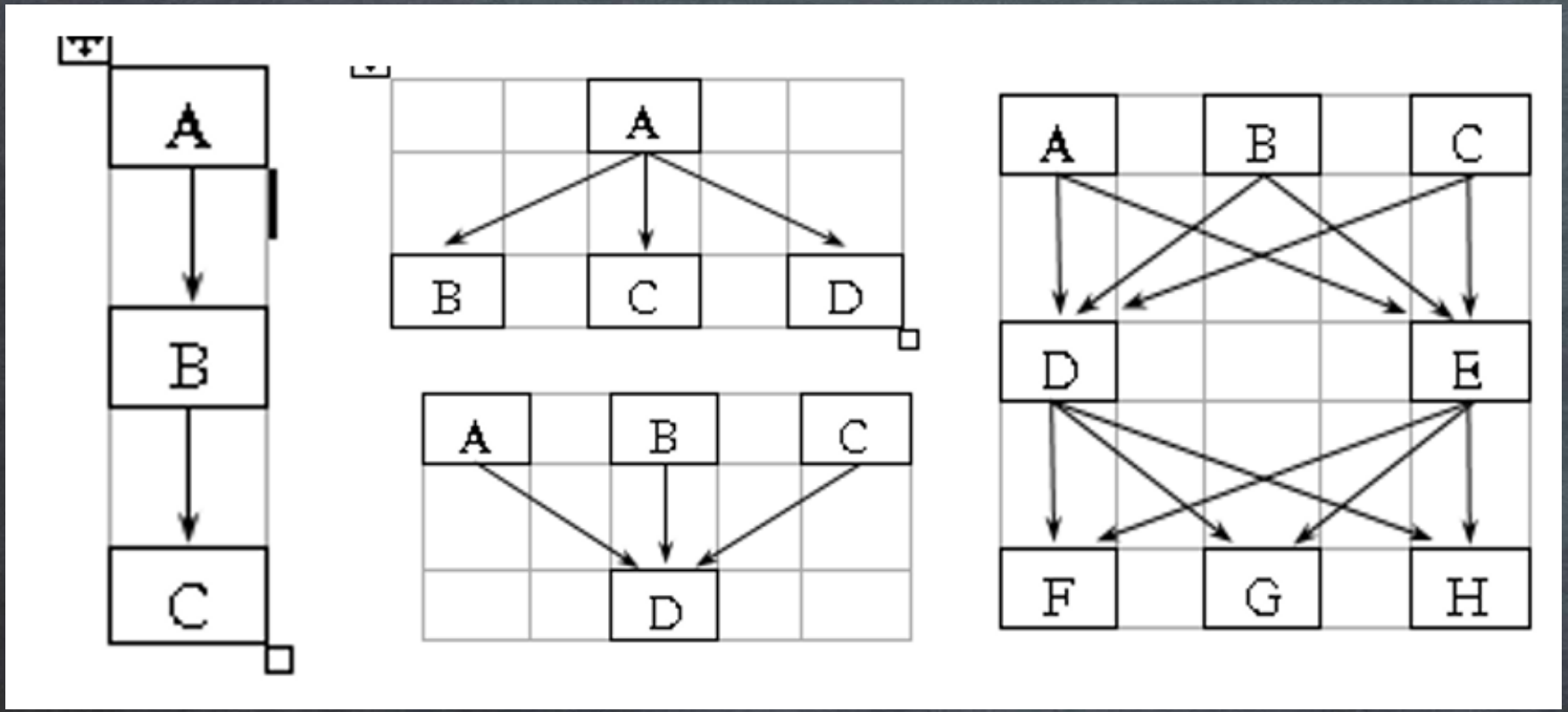
Nội dung

- Khái niệm kế thừa
- Hàm khởi tạo và hủy bỏ trong kế thừa
- Override / overload
- Đa kế thừa
- Lớp trừu tượng

Khái niệm kế thừa

- Lập trình hướng đối tượng có hai đặc trưng cơ bản:
  - Đóng gói dữ liệu, được thể hiện bằng cách dùng khái niệm lớp để biểu diễn đối tượng với các thuộc tính private, chỉ cho phép bên ngoài truy nhập vào thông qua các phương thức get/set.
  - Dùng lại mã, thể hiện bằng việc thừa kế giữa các lớp. Việc thừa kế cho phép các lớp thừa kế (gọi là lớp dẫn xuất) sử dụng lại các phương thức đã được định nghĩa trong các lớp gốc (gọi là lớp cơ sở)

Khái niệm kế thừa



Khai báo lớp kế thừa

- Cú pháp khai báo một lớp kế thừa từ một lớp khác như sau:

  class <Tên lớp dẫn xuất>: <Từ khóa dẫn xuất> <Tên lớp cơ sở>{

  … // Khai báo các thành phần lớp

  };

  - Tên lớp dẫn xuất: là tên lớp được cho kế thừa từ lớp khác. Tên lớp này tuân thủ theo quy tắc đặt tên biến trong C++.
  - Tên lớp cở sở: là tên lớp đã được định nghĩa trước đó để cho lớp khác kế thừa. Tên lớp này cũng tuân thủ theo quy tắc đặt tên biến của C++.
  - Từ khóa dẫn xuất: là từ khóa quy định tính chất của sự kế thừa. Có ba từ khóa dẫn xuất là private, protected và public.

Từ khóa dẫn xuất

Kiểu dẫn xuất	Tính chất lớp cơ sở	Tính chất lớp dẫn xuất
private	private protected public	không truy cập được private private
protected	private protected public	không truy cập được protected protected
public	private protected public	không truy cập được protected public

Bài tập

- Xây dựng lớp hình tròn là kế thừa lớp Điểm với các:
  - Hàm tạo, hàm hủy
  - Phương thức nhập xuất
- Xây dựng lớp hình tròn là lớp bao của lớp Điểm với:
  - Hàm tạo, hàm hủy
  - Phương thức nhập xuất

```
26  class Circle{
27      private:
28          Point A;
29          float r;
30      public:
31          Circle():A(){
32              r=0;
33          }
34
```

```
class Circle: public Point{
    private:
        float r;
    public:
        Circle(){
            r=0.0;
        }
```

Hàm khởi tạo trong kế thừa

- Khi khai báo một đối tượng có kiểu lớp được dẫn xuất từ một lớp cơ sở khác. Chương trình sẽ tự động gọi tới hàm khởi tạo của lớp dẫn xuất. Tuy nhiên, thứ tự được gọi sẽ bắt đầu từ hàm khởi tạo tương ứng của lớp cơ sở, sau đó đến hàm khởi tạo của lớp dẫn xuất. Do đó, thông thường, trong hàm khởi tạo của lớp dẫn xuất phải có hàm khởi tạo của lớp cơ sở.

```
<Tên hàm khởi tạo dẫn xuất>([<Các tham số>]):
<Tên hàm khởi tạo cơ sở>([<Các đối số>]){
… // Khởi tạo các thuộc tính mới bổ sung của lớp dẫn xuất
};
```

Hàm hủy trong kế thừa

- Khi một đối tượng lớp dẫn xuất bị giải phóng khỏi bộ nhớ, thứ tự gọi các hàm hủy bỏ ngược với thứ tự gọi hàm thiết lập: gọi hàm hủy bỏ của lớp dẫn xuất trước khi gọi hàm hủy bỏ của lớp cơ sở.
- Vì mỗi lớp chỉ có nhiều nhất là một hàm hủy bỏ, nên ta không cần phải chỉ ra hàm hủy bỏ nào của lớp cơ sở sẽ được gọi sau khi hủy bỏ lớp dẫn xuất. Do vậy, hàm hủy bỏ trong lớp dẫn xuất được khai báo và định nghĩa hoàn toàn giống với các lớp thông thường:
- <Tên lớp>::~<Tên lớp>([<Các tham số>]){
- … // giải phóng phần bộ nhớ cấp phát cho các thuộc tính bổ sung
- }

Bài tập

+ Lớp NGUOI gồm các thuộc tính:

char *ht ; // Họ tên

int ns ;

- Hai hàm tạo, phương thức in() và hàm huỷ

+ Lớp MON_HOC gồm các thuộc tính

char *monhoc ; // Tên môn học

int st ; // Số tiết

- Hai hàm tạo, phương thức in() và hàm huỷ

+ Lớp GIAO_VIEN :

- Kế thừa từ lớp NGUOI

- Đưa thêm các thuộc tính

char *bomon ; // Bộ môn công tác

MON_HOC mh ; // Môn học đang dậy

- Hai hàm tạo , phương thức in() và hàm huỷ

Câu hỏi

```
class A{
        protected: int a1;
        public: int a2;
        A(){a1=a2=0;}
        A(int t1, int t2){
        a1=t1; a2= t2;}
        void in(){
        cout << a1 <<"  " << a2 ;}
} ;
class B: private A{
        protected:int b1;
        public: int b2;
        B(){ b1=b2=0;}
        B(int t1, int t2, int u1, int u2){
        a1=t1; a2=t2; b1=u1;b2=u2;}
        void in(){
        cout << a1 <<"  " << a2 << "  " << b1 << "  " << b2;
} };
```

```
class C : public B{
        public: C(){b1=b2=0;}
        C(int t1, int t2, int u1,int u2){
        a1=t1; a2=t2; b1=u1;b2=u2; }
        void in(){
        cout << a1;
        cout  <<"  " << a2 << "  " << b1 << "  " << b2;
} };
void main(){
C c(1,2,3,4);
c.in();
}
```

Override method

- Một phương thức của lớp cơ sở được gọi là override nếu ở lớp dẫn xuất cũng định nghĩa một phương thức:
  - có cùng tên.
  - có cùng số lượng tham số.
  - có cùng kiểu các tham số (giống nhau từng đôi một theo thứ tự).
  - có cùng kiểu dữ liệu trả về.

Override method

Khi kế thừa nhiều mức thì quy tắc đặt tên các thành phần như sau:

- Tên các lớp không được trùng lặp
- Tên các thành phần trong một lớp cũng không được trùng lặp
- Tên các thành phần trong các lớp khác nhau có quyền được trùng lặp.
- Để phân biệt các thành phần trùng tên trong lớp dẫn xuất, cần sử dụng thêm tên lớp

Override method

```
class A{
        private:
            int x;
        public:
            void nhap()          {   cout<<“x=”;cin>>x; }
};
class B:public A{
        private:
            int y;
        public:
        void nhap()  {    cout<<“\n y=”;cin>>y};
};
```

```
void main(){
        B h;
        h.B::nhap();//chỉ rõ lớp
        h.A::nhap(); //chỉ rõ lớp
}
```

Override method

- Trong trường hợp không chỉ rõ tên lớp thì: Chương trình dịch C++ phải tự phán đoán để biết thành phần đó thuộc lớp nào. Cách phán đoán như sau:
  - Trước tiên xem thành phần đang xét có trùng tên với một thành phần nào của lớp dẫn xuất không?
  - Nếu trùng thì đó là thành phần của lớp dẫn xuất.
  - Nếu không trùng thì tiếp tục xét các lớp cơ sở theo thứ tự: Các lớp có quan hệ gần với lớp dẫn xuất xét trước, các lớp quan hệ xa xét sau.

Override method

```
class A{
        private:
            int x;
        public:
            void nhap()         {   cout<<“x=”;cin>>x; }
};
class B:public A{
        private:
            int y;
        public:
        void nhap()  {    cout<<“\n y=”;cin>>y};
};
```

```
void main(){
            B h;
            h.nhap();//ưu tiên gọi đến nhap()
của lớp B
            h.B::nhap();//chỉ rõ lớp
            h.A::nhap();
}
```

Overload method

- Một số phương thức trong cùng một lớp được coi là overload nếu:
- có cùng tên.
- khác số lượng tham số.

Overload method

- Một số toán tử như +, -, *, /, >>, <<, … có thể overload được
- Ví dụ:

```
PhanSo operator+(PhanSo ps) {
        PhanSo kq;
        kq.tu = this->tu * ps.mau + ps.tu * this->mau;
        kq.mau = this->mau * ps.mau;
        kq.rutGon();
        return kq;
}
```

Khi đó có thể sử dụng ps1 + ps2

Xây dựng lớp DaThuc

```
class DaThuc{
        private:
                int bac;
                int *heSo;
        public:
                DaThuc(){
                        bac=0;
                        heSo=new int[bac];   }
                ~DaThuc(){
                        bac=0;
                        delete heSo;         }
                DaThuc operator + (DaThuc);


};
```

Viết lại toán tử + để cộng 2 đa thức

```
DaThuc DaThuc:: operator +(DaThuc P){
        DaThuc tong;
        if (bac<=P.bac) {
                tong.bac=P.bac;
                tong.heSo=new int[P.bac];
                for (int i=0;i<=bac;i++)        tong.heSo[i]=heSo[i]+P.heSo[i];
                for (int j=bac+1;j<=P.bac;j++)   tong.heSo[j]=P.heSo[j];
        }
        else {
                tong.bac=bac;
                tong.heSo=new int[bac];
                for (int i=0;i<=P.bac;i++)       tong.heSo[i]=heSo[i]+P.heSo[i];
                for (int j=P.bac+1;j<=bac;j++)           tong.heSo[j]=heSo[j];
        }
        return tong;
}
```

Overload method

- Overload toán tử >> để sử dụng với cin
- Ví dụ:

```
friend istream& operator>> (istream &is, PhanSo &ps)
        {
                return is >> ps.tu >> ps.mau;
        }
```

Khi đó có thể sử dụng cin >> ps1;

Overload method

- Overload toán tử << để sử dụng với cout
- Ví dụ:

```
friend ostream& operator<< (ostream &os, PhanSo const& ps)
                {
                        return os << ps.tu << "/" << ps.mau;
                }
```

- Khi đó có thể sử dụng cout << ps1;

Bài tập

Viết chương trình nhập vào 2 điểm trong mặt phẳng tọa độ Oxy.

- Nhập vào tọa độ cho từng điểm sử dụng cin
- Sử dụng cout để in ra tọa độ điểm
- Tính khoảng cách giữa 2 điểm

Đa kế thừa

- C++ cho phép đa kế thừa, tức là một lớp có thể được dẫn xuất từ nhiều lớp cơ sở khác nhau, với những kiểu dẫn xuất khác nhau
- Đa kế thừa được khai báo theo cú pháp:

class <Tên lớp dẫn xuất>: <Từ khoá dẫn xuất> <Tên lớp cơ sở 1>, <Từ khoá dẫn xuất> <Tên lớp cơ sở 2>, …<Từ khoá dẫn xuất> <Tên lớp cơ sở n>{

… // Khai báo thêm các thành phần lớp dẫn xuất

};

- Ví dụ:

class Bus: public Car, public PublicTransport{

… // Khai báo các thành phần bổ sung

};

Đa kế thừa

```
#include <iostream.h>
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; } ;
```

```
void main(){
D  h ;
h.d = 4 ;
h.c =  3 ; h.b = 2 ; h.a =  1 ;}
```

Trong ví dụ này A là cơ sở cho cả 2 lớp cơ sở trực tiếp của D là B và C. Nói cách khác có 2 lớp cơ sở A cho lớp D. Vì vậy trong câu lệnh:

```
h.a = 1 ;
```

thì Chương trình dịch C++ không thể nhận biết được thuộc tính a thừa kế thông qua B hay thông qua C và nó sẽ đưa ra thông báo lỗi sau:

Member is ambiguous: 'A::a' and 'A::a'

Lớp trừu tượng

- Sự cho phép đa kế thừa trong C++ dẫn đến một số hậu quả xấu, đó là sự xung đột giữa các thành phần của các lớp cơ sở, khi có ít nhất hai lớp cơ sở lại cùng được kế thừa từ một lớp cơ sở khác.
- Để tránh các vấn đề này, C++ cung cấp một khái niệm là kế thừa từ lớp cơ sở trừu tượng
- Việc chỉ ra một sự kế thừa trừu tượng được thực hiện bằng từ khoá virtual khi khai báo lớp cơ sở:

class <Tên lớp dẫn xuất>: <Từ khoá dẫn xuất> virtual <Tên lớp cơ sở>{

… // Khai báo các thành phần bổ sung

};

Lớp trừu tượng

Các lớp cơ sở ảo (virtual) sẽ được kết hợp để tạo một lớp cơ sở duy nhất cho bất kỳ lớp nào dẫn xuất từ chúng. Trong ví dụ trên, hai lớp cơ sở A (A là cơ sở của B và A là cơ sở của C) sẽ kết hợp lại để trở thành một lớp cơ sở A duy nhất cho bất kỳ lớp dẫn xuất nào từ B và C. Như vậy bây giờ D sẽ chỉ có một lớp cơ sở A duy nhất, do đó phép gán:

h.a = 1 ;

sẽ gán 1 cho thuộc tính a của lớp cơ sở A duy nhất mà D kế thừa.

Tương ứng bội

```
class Car{
        public:         void show();};
class Bus: public Car{
        public: void show();};
Bus myBus;
Car *ptrCar = &myBus;
ptrCar->show();
```

Phương thức ảo

Phương thức trừu tượng (còn gọi là phương thức ảo, hàm ảo) được khai báo với từ khoá virtual:

- Nếu khai báo trong phạm vi lớp:

virtual <Kiểu trả về> <Tên phương thức>([<Các tham số>]);

- Nếu định nghĩa ngoài phạm vi lớp:

virtual <Kiểu trả về> <Tên lớp>::<Tên phương thức>([<Các tham số>]){…}

Bài tập

Viết chương trình quản lý Sinh viên (Mã SV, Họ tên, tuổi, Lớp, Điểm TB) và Giảng viên (Mã GV, Họ tên, tuổi, Khoa), kế thừa từ lớp Người.

- Đưa ra danh sách Sinh viên có điểm TB >= 8.0
- Đưa ra danh sách GV thuộc khoa CNTT