



华中科技大学

第9章 多继承与虚基类

许向阳

xuxy@hust.edu.cn





内容

9.1 多继承类

9.2 虚基类

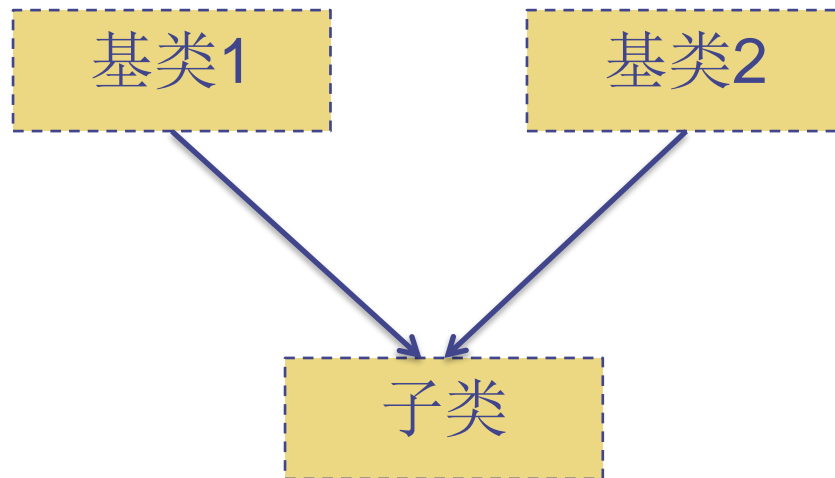
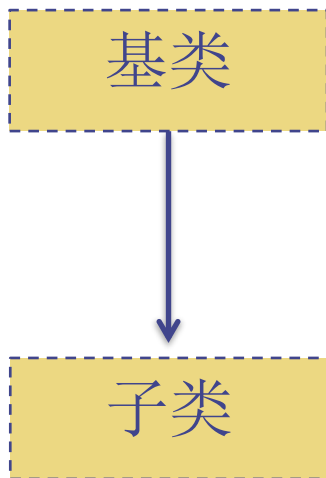
9.3 派生类成员

9.4 单重及多重继承的构造与析构

9.5 多继承类的内存布局



9.1 多继承类



```
Class 子类 : 派生控制 基类
{
    .....
};
```

```
Class 子类 : 派生控制 基类1,
             派生控制 基类2
{
    .....
};
```



9.1 多继承类

```
Class 子类 : 【virtual】 【派生控制】 基类1 ,  
             【派生控制】 【virtual】 基类2  
{  
    .....  
};
```

派生控制: private, protected , public

缺省派生控制: private

在基类名称前若有: virtual , 虚基类





9.1 多继承类

多继承派生类

- 有多个基类或虚基类;
- 同一个类不能多次作为某个派生类的直接基类
`class QUEUE: STACK, STACK{...};` // 错误, 出现两次
- 同一个类可多次作为一个派生类的间接基类;
`class QUEUE: STACK {STACK d;};`
- 继承所有基类的数据成员和函数成员;
- 在继承基类时, 各基类可采用不同的派生控制符;
- 基类之间的成员可同名, 基类与派生类的成员也可同名;
- 在出现同名时, 如面向对象的作用域不能解析, 可使用基类类名加作用域运算符::来指明所要访问的基类的成员。





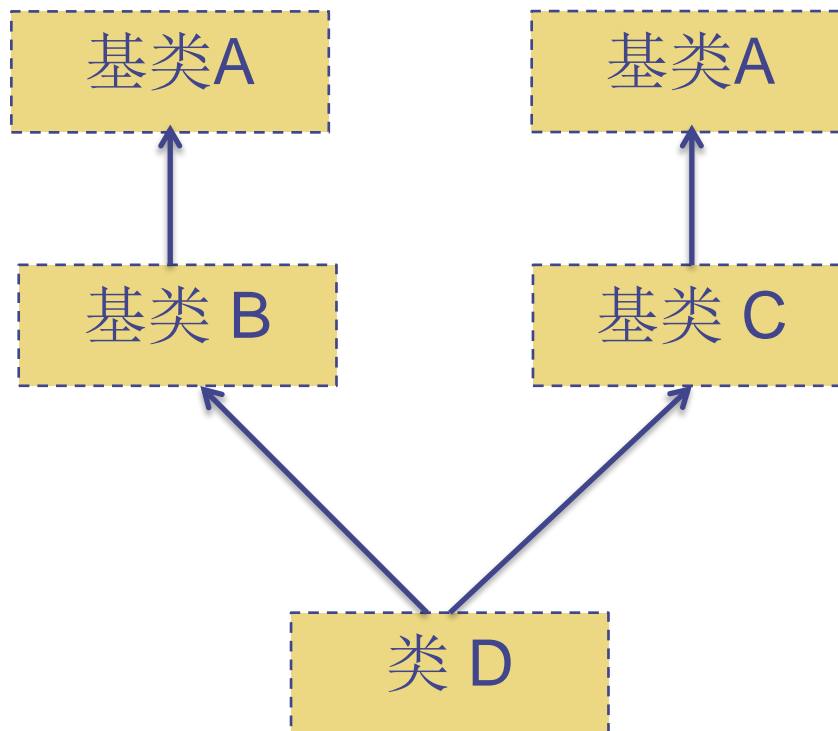
9.1 多继承类

```
struct A { int a;  
    A(int a) { this->a = a; cout << "a= " << a << endl; }  
};  
struct B :public A { int b;  
    B(int b):A(3) {B::b = b; cout << "b = " << b << endl; }  
};  
struct C :public A { int c;  
    C(int c):A(5) {C::c = c; cout << "c = " << c << endl; }  
};  
struct D :public B, public C { int d;  
    D(int b, int c, int d):C(c), B(b) {  
        D::d = d; cout << "d = " << d << endl; }  
};
```

D d(10,20,30); 先构造基类 B, 输出 a=3; b=10
再构造基类 C, 输出 a=5; c=20; 最后 d=30;



9.1 多继承类



D d(10,20,30); 先构造基类 B, 输出 a=3 b=10
再构造基类 C, 输出 a=5 c=20
最后 d=30



9.1 多继承类

- ◆ 当对象成员和基类存在共同的基类时，就可能对同一个物理对象重复初始化(可能是危险的和不必要的)。
- ◆ 两栖机车AmphibiousVehicle继承基类陆用机车LandVehicle，委托对象成员水上机车WaterVehicle完成水上功能。两栖机车可能对同一个物理对象Engine初始化(启动)两次。

```
class Engine { ...};  
class LandVehicle: Engine { ... };  
class WaterVehicle: Engine { ...};  
class AmphibiousVehicle: LandVehicle  
    {WaterVehicle  wv; ...};
```

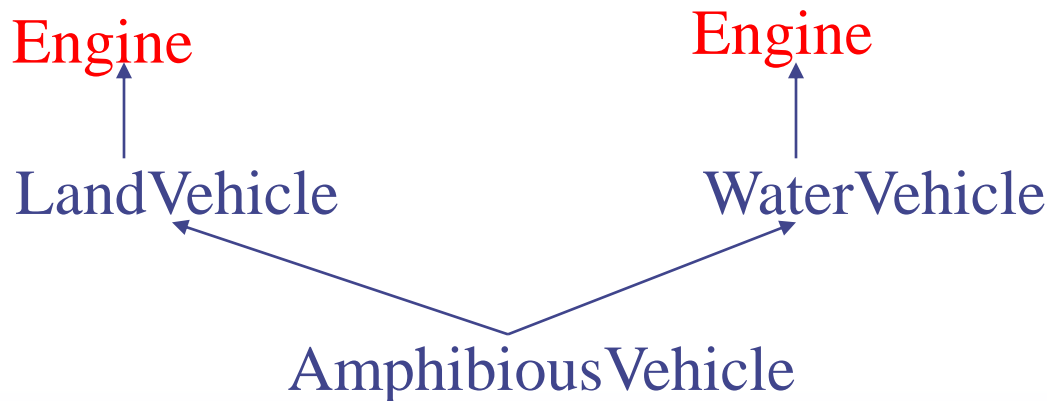


9.1 多继承类

- ◆ 多继承机制描述两栖机车 AmphibiousVehicle:

```
class AmphibiousVehicle: LandVehicle,  
WaterVehicle { /*...*/};
```

- ◆ 多继承不能解决同一个物理对象初始化两次的问题。
- ◆ 可以采用全局变量、静态数据成员，解决同一个物理对象初始化两次的问题，但解决相关析构问题复杂。



- ◆ 引入虚基类解决两个对象共享一个物理对象问题。

9.2 虚基类

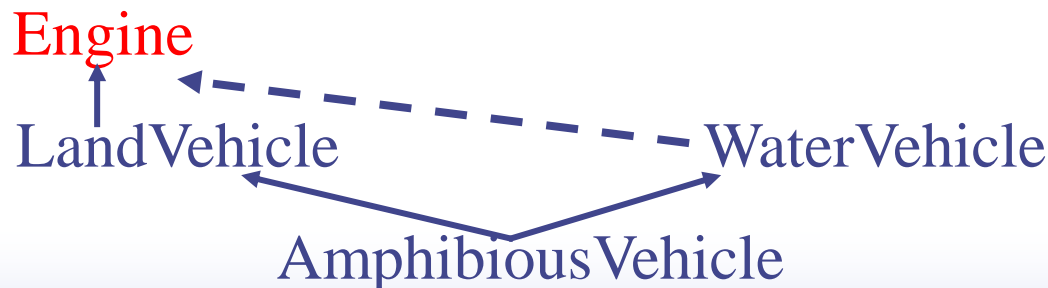
- 用virtual声明，把多个逻辑对象映射成**同一个物理对象**。
- 该物理对象尽可能早的构造、尽可能晚的析构，构造和析构都只进行一次。若虚基类的构造函数都有参数，必须在派生类构造函数的初始化列表中列出虚基类构造参数。

```
class Engine{ /*...*/ };
```

```
class LandVehicle: virtual public Engine{ /*...*/ };
```

```
class WaterVehicle: public virtual Engine{ /*...*/ };
```

```
class AmphibiousVehicle: LandVehicle, WaterVehicle { /*...*/ };
```





9.2 虚基类

- ◆ 同一棵派生树中的同名虚基类，共享同一个存储空间；其构造和析构仅执行1次，且构造尽可能最早执行，而析构尽可能最晚执行。
- ◆ 由派生类(根)、基类和虚基类构成一个派生树的节点，而对象成员将成为一棵新派生树的根。
- ◆ 如果虚基类与基类同名，则它们将分别拥有各自的存储空间，只有同一棵派生树的同名虚基类才共享存储空间，而同名基类则拥有各自的存储空间。
- ◆ 虚基类和基类同名必然会导致二义性访问
建议：将基类说明为对象成员，或将基类都说明为虚基类。
可用作用域运算符限定要访问的成员。





9.2 虚基类

```
struct A { int a;
    A(int a) { this->a = a; cout << "a= " << a << endl; }
};
struct B :public A { int b;
    B(int b):A(3) {B::b = b; cout << "b = " << b << endl; }
};
struct C :public A { int c;
    C(int c):A(5) {C::c = c; cout << "c = " << c << endl; }
};
struct D :virtual public B, virtual public C { int d;
    D(int b, int c, int d):C(c), B(b) {
        D::d = d; cout << "d = " << d << endl; }
};
```

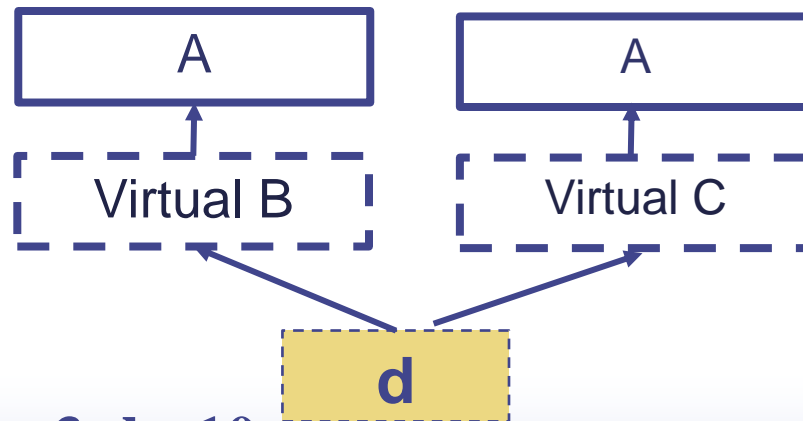
D d(10,20,30); 先构造虚基类 B, 输出 a=3; b=10
再构造虚基类 C, 输出 a=5; c=20; 最后 d=30;



9.2 虛基類

```
struct A { int a; A(int a) {cout << a<<endl; } };
struct B :public A { int b; B(int b):A(3) {...} };
struct C :public A { int c; C(int c):A(5) {...} };
struct D :virtual public B, virtual public C { int d;
    D(int b,int c,int d):C(c),B(b) {
        D::d = d; cout << "d = "<<d<<endl; }
};
```

D d(10,20,30);



先构造虚基类 B，输出 a=3; b=10
再构造虚基类 C，输出 a=5; c=20; 最后 d=30;



9.2 虚基类

```
struct A { int a;
  A(int a) { this->a = a; cout << "a= " << a << endl; } };
struct B : public A { int b;
  B(int b):A(3) { B::b = b; cout << "b = " << b << endl; } };
struct C : public A { int c;
  C(int c):A(5) { C::c = c; cout << "c = " << c << endl; } };
struct D : public B, virtual public C { int d;
  D(int b, int c, int d):C(c), B(b) {
    D::d = d; cout << "d = " << d << endl; }
};
```

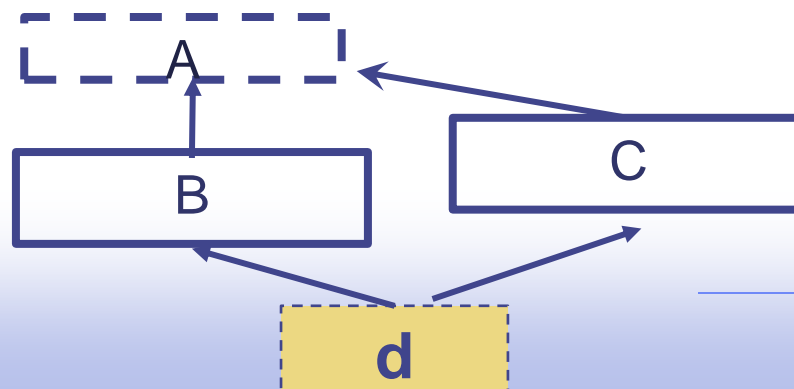
D d(10,20,30); 先构造虚基类 C，输出 a=5; c=20
再构造基类 B，输出 a=3; b=10; 最后 d=30;



9.2 虚基类

```
struct A { int a;  
    A(int a) { this->a = a; cout << "a= " << a << endl; } };  
struct B : virtual public A { int b;  
    B(int b):A(3) { B::b = b; cout << "b = " << b << endl; } };  
struct C : virtual public A { int c;  
    C(int c):A(5) { C::c = c; cout << "c = " << c << endl; } };  
struct D : public B, public C { int d;  
    D(int b, int c, int d):C(c), B(b) {  
        D::d = d; cout << "d = " << d << endl; } };
```

// 错误: A ::A 没有合适的默认构造函数





9.2 虚基类

```
struct A { int a;
    A(int a) { this->a = a; cout << "a= " << a << endl; } };
struct B : virtual public A { int b;
    B(int b):A(3) { B::b = b; cout << "b = " << b << endl; } };
struct C : virtual public A { int c;
    C(int c):A(5) { C::c = c; cout << "c = " << c << endl; } };
struct D : public B, public C { int d;
    D(int b, int c, int d):C(c), B(b), A(111) {
        D::d = d; cout << "d = " << d << endl; } };
D(int b, int c, int d):C(c), B(b) { ... }
```











输出 : a =111; d. B::A::a -> d. B::a
 b=10; d. C::A::a -> d. C::a
 c=20; d. a
 d=30;













9.2 虛基類

```
struct A { ..... };
struct B :virtual public A { int b;...};
struct C :virtual public A { int c; ...};
struct D : public B, public C { int d;
    D(int b,int c,int d):C(c),B(b),A(111) {
        D::d = d; cout << "d = " << d << endl; }
};
```

輸出 : a =111;
b=10;
c=20;
d=30;

▲  d	{d=30 }
▲  B	{b=10 }
▶  A	{a=111 }
 b	10
▲  C	{c=20 }
▶  A	{a=111 }
 c	20
▲  A	{a=111 }
 a	111
 d	30

9.2 虛基類

▲  d	{d=30 }
▲  B	{b=10 }
▶  A	{a=111 }
 b	10
▲  C	{c=20 }
▶  A	{a=111 }
 c	20
▲  A	{a=111 }
 a	111
 d	30

&d = 0x0030FCF0

0x0030FCF0: f4 9c 3c 00 0a 00 00 00
 e4 9c 3c 00 14 00 00 00
 1e 00 00 00 6f 00 00 00



9.2 虚基类

d.B::a = 123;

```
mov     eax,dword ptr [d]
mov     ecx,dword ptr [eax+4]
mov     dword ptr d[ecx],7Bh
```

d.C::a = 124;

```
mov     eax,dword ptr [d]
mov     ecx,dword ptr [eax+4]
mov     dword ptr d[ecx],7Ch
```

d.a = 125;

```
mov     eax,dword ptr [d]
mov     ecx,dword ptr [eax+4]
mov     dword ptr d[ecx],7Dh
```

最开始的 003c9cf4
可以看成是一个指针，
但指向的并不是共享
单元。在该地址加4
的单元中，存放的是
共享单元在 对象中
的偏移地址





9.2 虚基类

```
class Engine{ int power; public: Engine(int p): power(p){ } };
class LandVehicle: virtual public Engine{
    int speed;
public: //如从AmphibiousVehicle调用LandVehicle, 则不会在此调用Engine(p)
    LandVehicle(int s, int p): Engine(p), speed(s){ }
};
class WaterVehicle: public virtual Engine{
    int speed;
public: //如从AmphibiousVehicle调用WaterVehicle, 则不会在此调用
    Engine(p)
    WaterVehicle(int s, int p): Engine(p), speed(s){ }
};
struct AmphibiousVehicle: LandVehicle, WaterVehicle {
    AmphibiousVehicle(int s1, int s2, int p): //先构造虚基类再基类
    WaterVehicle(s2, p), LandVehicle(s1, p), Engine(p){ } //整个派生树
    Engine(p)只1次
}; //初始化顺序: Engine(p), LandVehicle(s1, p), WaterVehicle(s2, p), 而且
    进入两个
    //LandVehicle, WaterVehicle 后, 不再初始化这两个基类的基类Engine
```





9.3 派生类成员

- 当派生类有多个基类或虚基类时，基类或虚基类的成员之间可能出现同名；派生类和基类或虚基类的成员之间也可能出现同名。
- 出现上述同名问题时，必须通过面向对象的作用域解析，或者用基类名加作用域运算符`::`指定要访问的成员，否则就会引起二义性问题。
- 当派生类成员和基类成员同名时，优先访问作用域小的成员，即优先访问派生类的成员。
- 当派生类数据成员和派生类函数成员的参数同名时，在函数成员内优先访问函数参数。





9.3 派生类成员

```
struct A{
    int a, b, c, d;
};
struct B{
    int b, c;
protected:
    int e;
};
class C: public A, public B{
    int a;
public:
    int b;  int f(int c);
};
```

```
int C::f(int c){
    int i=a;          //访问C::a
    i=A::a;
    i=b+c+d;          //访问C::b和参数c
    i=A::b+B::b;      //访问基类成员
    return A::c;
}
void main(void){
    C x;
    int i=x.A::a;
    i=x.b;             //访问C::b
    i=x.A::b+x.B::b;
    i=x.A::c;
}
```



9.4 单重及多重继承的构造与析构

- ◆ 析构和构造的顺序相反，派生类对象的构造顺序：
 - 按自左向右、自下而上地构造倒派生树中所有虚基类
 - 按定义顺序构造派生类的所有直接基类
 - 按定义顺序构造(初始化)派生类的所有数据成员，包括对象成员、`const`成员和引用成员
 - 执行派生类自身的构造函数体

- ◆ 如果构造中虚基类、基类、对象成员、`const`及引用成员又是派生类对象，则派生类对象重复上述构造过程，但同名虚基类对象在同一棵派生树中仅构造一次。

- ◆ 由派生类(根)、基类和虚基类构成一个派生树的节点，而对象成员将成为一棵新派生树的根。



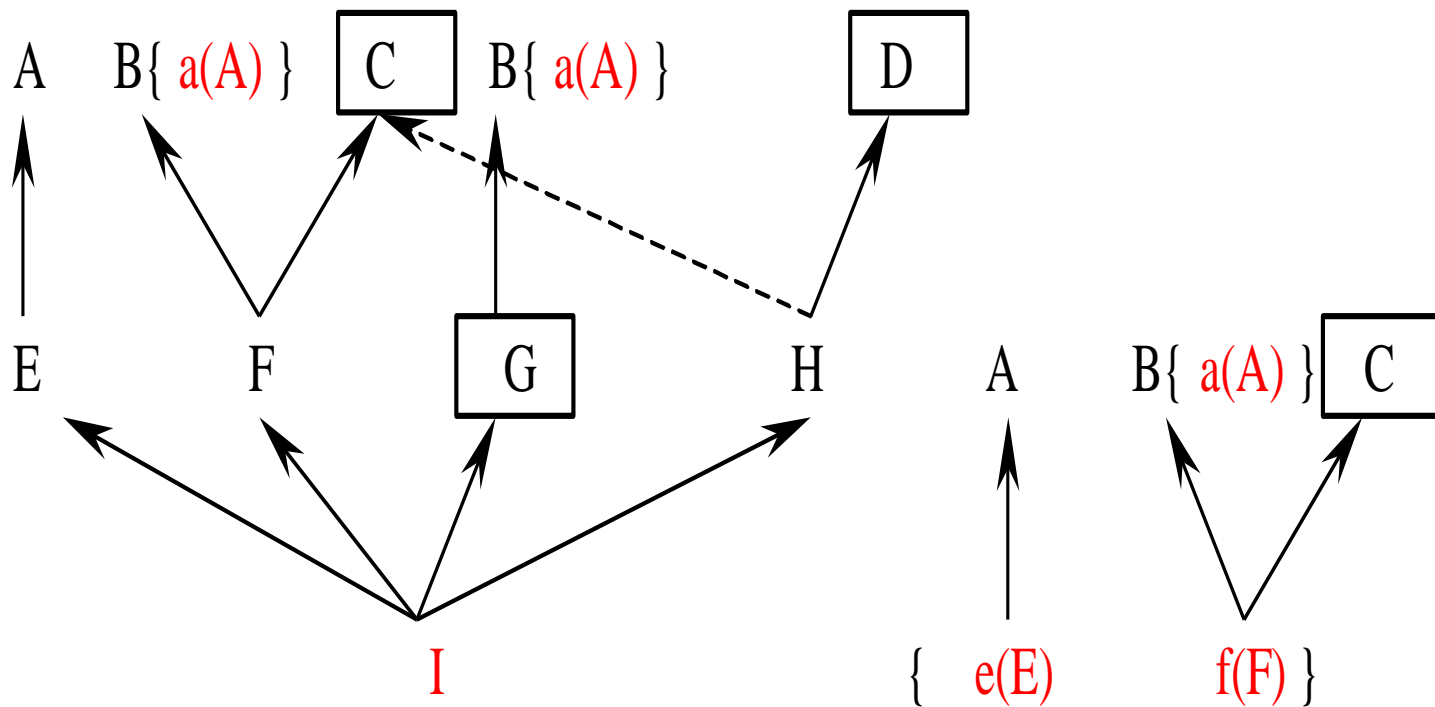


9.4 单重及多重继承的构造与析构

```
#include <iostream.h>
struct A{ A( ) { cout<<'A';} };
struct B { const A a; B( ) { cout<<'B';} }; //成员a将作为新根
struct C{ C( ) { cout<<'C';} };
struct D{ D( ) { cout<<'D';} };
struct E: A{ E( ) { cout<<'E';} };
struct F: B, virtual C{ F( ) { cout<<'F';} };
struct G: B{ G( ) { cout<<'G';} };
struct H: virtual C, virtual D{ H( ) { cout<<'H';} };
struct I: E, F, virtual G, H{ E e; F f; I( ) { cout<<'I';} };
void main(void) { I i; }
```



9.4 单重及多重继承的构造与析构



派生树(根红色), 输出: C ABG D AE ABF H AE C ABF I

先虚基类: 从左向右 (E→F→G→H)、自下而上。

判断顺序 E → F, F上有虚基类 C; 再是虚基类 G
最后是H上的虚基类 D。 用树的后根遍历找虚基类。



9.5 多继承类的内存布局

- 多继承派生类包含各个基类的存储空间。
- 如果存在虚基类和同名基类，虚基类和同名基类的存储空间是相互独立的。
- 如果派生类存在同名的虚基类，同一棵派生树的所有虚基类共享存储空间，虚基类通过偏移指向共享存储空间，该存储空间出现在所有直接基类之后。
- 如果基类或派生类存在虚函数，则在派生类存储空间中，包含一个单元存放虚函数入口地址表首地址。
- 派生类的存储空间不包括基类、虚基类和对象成员的静态数据成员。



9.5 多继承类的内存布局

```
class A{  
    int a;  
public:  
    virtual void f1( ) { };  
};  
class B{  
    int b, c;  
public:  
    virtual void f2( ) { };  
};  
class C{  
    int d;  
public:  
    void f3( ) { };  
};  
class D: A, B, C{  
    int e;  
public:  
    virtual void f4( ) { };  
};
```

虚函数入口 地址表首址	A	D
int a		
虚函数入口 地址表首址	B	
int b		
int c		
int d	C	
int e		



9.5 多继承类的内存布局

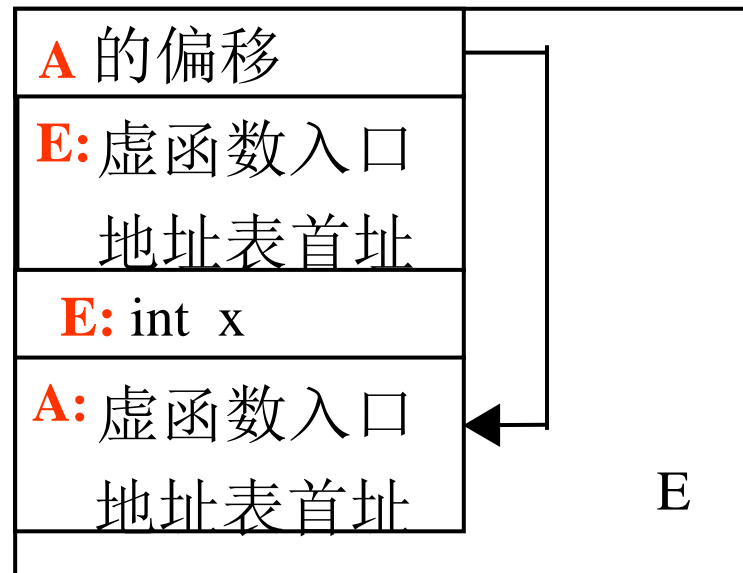
- 派生类有虚基类的情况下，虚基类的存储空间建于派生类的尾部，且按虚基类的构造顺序建立：
 - (1) 派生类依次处理每个直接基类或虚基类，如果为直接基类，则为其建立存储空间，如果为直接虚基类则建立一个到虚基类的偏移。
 - (2) 如果派生类继承的第一个类为非虚基类，且该基类定义了虚函数地址表，则派生类就共享该表首址占用的存储单元。对于其他任何情形，派生类在处理完所有基类或虚基类后，根据派生类是否新定义了虚函数，确定是否为该表首址分配存储单元。



9.5 多继承类的内存布局

```
struct A{  
    virtual void fa( ) { };  
};
```

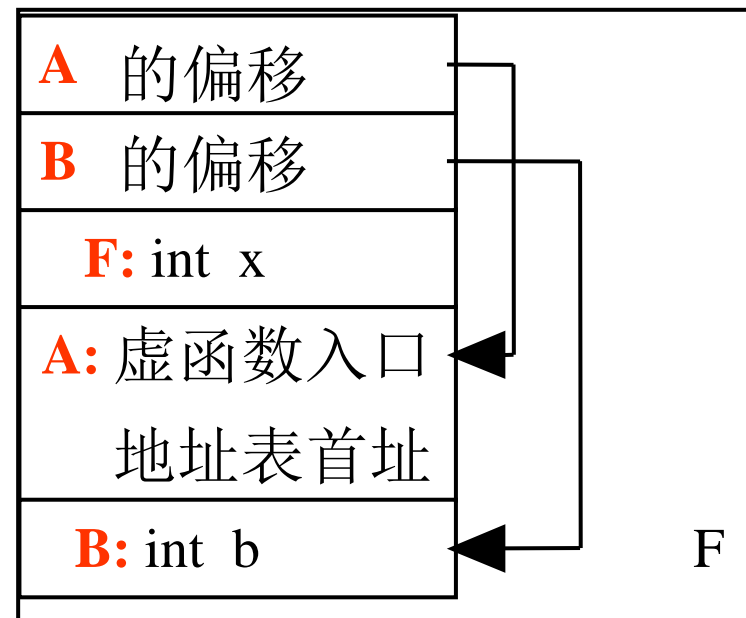
```
struct E: virtual A{  
    int x;  
    virtual void fe( ) { };  
};
```



9.5 多继承类的内存布局

```
struct B{  
    int b;  
    void fb( );  
};
```

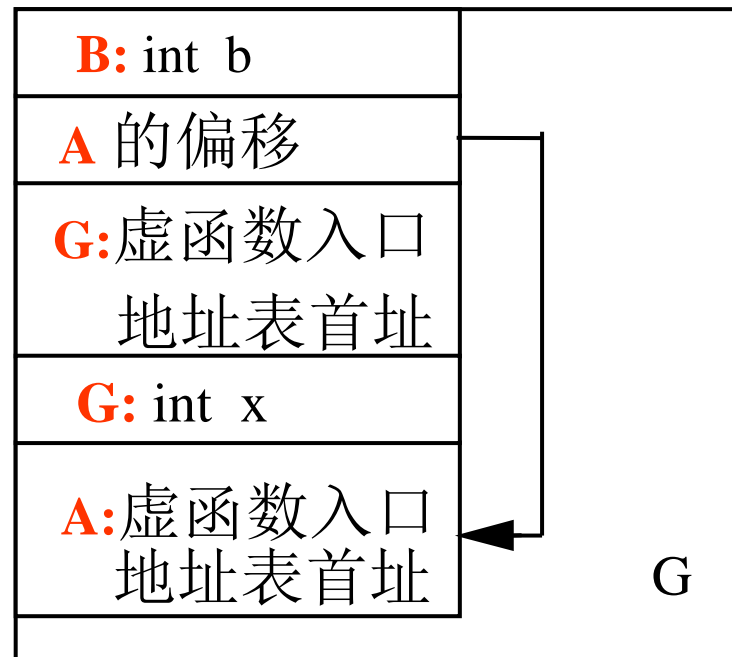
```
struct F:virtual A, virtual B{  
    int x;  
    void ff( ) { };  
};
```



9.5 多继承类的内存布局

```
struct B{
    int b;
    void fb( );
};
```

```
struct G: B, virtual A{
    int x;
    virtual void fg( ) { };
};
```





总结

多继承

语法格式、数据成员的访问
构造与析构顺序、存储结构

虚基类

有虚基类时，构造与析构的顺序
数据成员的访问

