C++程序设计精要教程

华中科技大学

第9章　多继承与虚基类

◆9.1　多继承类

- 单继承是多继承的一种特例，多继承派生类具有更强的类型表达能力。
- 多继承派生类有多个基类或虚基类。
- 派生类继承所有基类的数据成员和成员函数。
- 派生类在继承多个基类时，不同的基类可以采用不同的派生控制。
- 基类之间的成员可能同名，基类与派生类的成员也可能同名。在出现同名时，如面向对象的作用域不能解析，应该使用作用域运算符来指明所要访问的类的成员。

第9章　多继承与虚基类

◆通过单继承模拟多继承

- 多继承机制是C++语言所特有的 (Java、C#、SmallTalk等没有)。其他面向对象语言需要描述多继承类的对象时，常常通过对象成员或委托代理实现多继承。
- 委托代理在多数情况下能够满足需要，但当对象成员和基类类型相同或存在共同的基类时，就可能对同一个物理对象重复进行初始化(可能是危险的和不需要的)。
- 通常需要重新设计类，使派生类不包含重复的基类或对象成员。

第9章　多继承与虚基类

【例9.1】定义具有水平滚动条和垂直滚动条的窗口类。

```
class Window {
   //...
public:
   Window(int top, int left, int bottom, int right);
   ~Window( );
};
class Hscrollbar {
   //...
public:
   HScrollbar(int top, int left, int bottom, int right);
   ~ HScrollbar( );
};
class Vscrollbar {
   //...
public:
   VScrollbar (int top, int left, int bottom, int right);
   ~ VScrollbar( );
};
```

第9章　多继承与虚基类

```
class ScrollableWind: public Window {
    HScrollbar  hScrollBar;   //委托hScrollBar代理水平滚动
    VScrollbar  vScrollBar;   //委托vScrollBar代理垂直滚动
    //...
public:
    ScrollableWind(int top, int left, int bottom, int right);
    ~ScrollableWind( );
};
ScrollableWind::ScrollableWind (int t, int l, int b, int r):Window(t, l, b, r), hScrollbar(t, r+1,
    b-1, r), vScrollbar(b-1,l-1,b,r+1) {
    //...
};
```

Window、hScrollBar和vScrollBar分别初始化显示端口，则派生类ScrollableWind的对象就会多次初始化显示端口，从而导致显示屏出现多次闪烁。

第9章　多继承与虚基类

用多继承方式定义派生类 ScrollableWind：

```
class ScrollableWind: public Window, public HScrollbar, public Vscrollbar {
    //...
public:
    ScrollableWind(int top, int left, int bottom, int right);
    ~ScrollableWind( );
};
ScrollableWind::ScrollableWind (int t, int l, int b, int r): Window(t, l, b, r),
            HScrollbar(t, r+1, b-1, r), VScrollbar(b-1,l-1,b,r+1) {
    //...
}
```

1. 多继承派生类的定义：

```
class 派生类名:<派生方式> 基类1,<派生方式> 基类2, … {
    <类体>
};
```

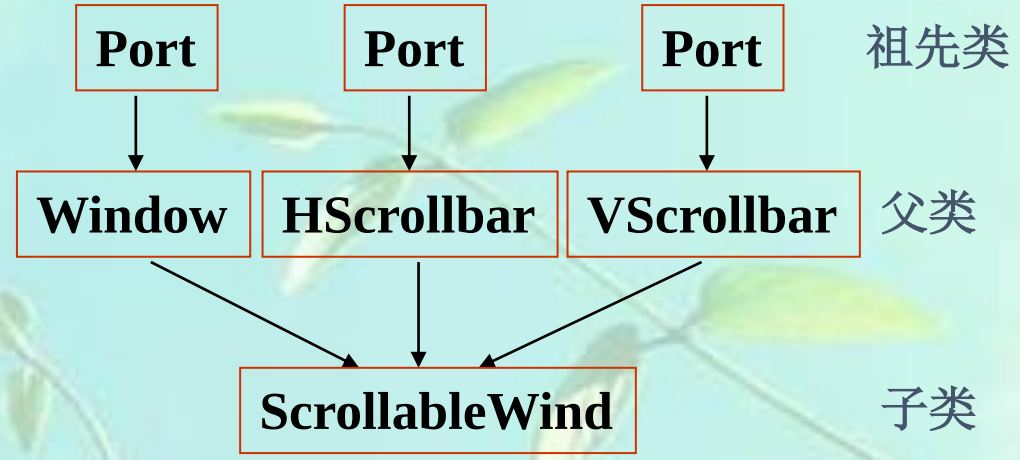
public
protected
private

2. 存在派生类对象多次初始化同一(物理)基类对象问题。

第9章　多继承与虚基类

派生类对象多次初始化同一基类成员问题（多次闪烁）：假设类Window、HScrollbar、VScrollbar都是从基类Port派生，即：

```
class Port { … };
class Window: public Port { … };
class HScrollbar: public Port { … };  class VScrollbar: public Port { … };
class ScrollableWind: public Window, public HScrollbar, public VScrollbar{…};
```



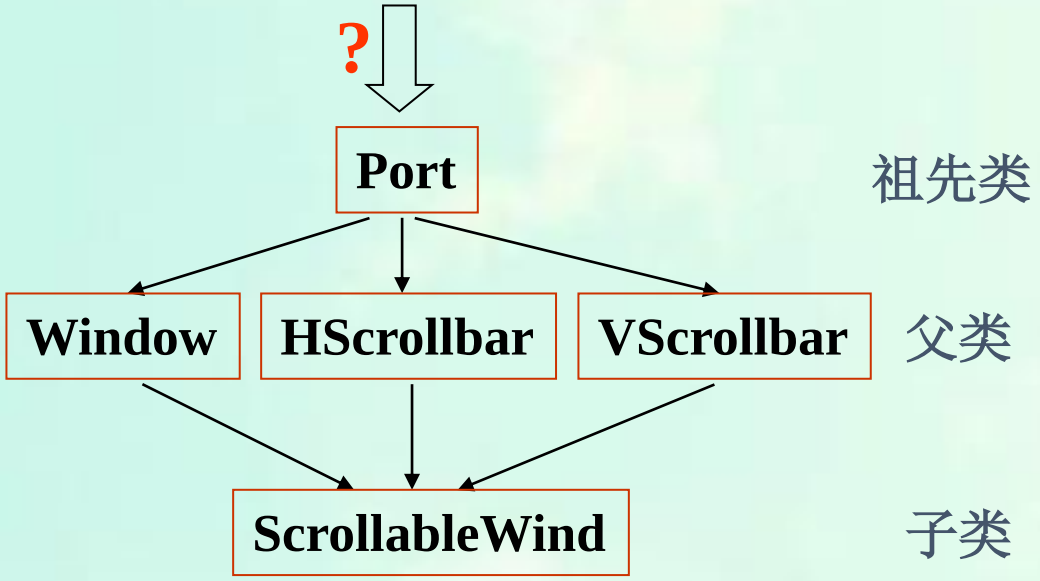
ScrollableWind 派生树

创建ScrollableWind对象时，Port的构造函数通过3条不同的路径，被调用了3次，从而将显示端口初始化3次。即1个子类有3个同名祖先类，而物理显示端口只有一个！

第9章　多继承与虚基类

◆9.2　虚基类

如何实现：创建ScrollableWind对象时，显示端口Port仅被初始化1次？用virtual定义虚基类。



ScrollableWind派生树

```
class Window: virtual public Port  { … };
class HScrollbar: public virtual Port  { … };
class VScrollbar: public virtual Port  { … };
class ScrollableWind: public Window, public HScrollbar, public VScrollbar  { … };
```

第9章　多继承与虚基类

◆9.2　虚基类

- 同一个类不能多次作为某个派生类的直接基类，但可多次作为其间接基类，从而引起存储空间的浪费和其他问题。此时，这些间接基类可定义为虚基类。
- 同一颗派生树中的同名虚基类，共享同一个存储空间；其构造函数和析构函数仅执行1次，且构造函数尽可能最早执行，而析构函数尽可能最晚执行。
- 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，编译程序会对这种二义性访问提出警告。当出现这种情况时，建议：要么将基类说明为对象成员，要么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。

第9章　多继承与虚基类

【例9.3】说明虚基类的二义性访问问题。

```
#include <iostream>
using namespace std;
struct A {
    int a;
    A(int x) { a = x; }
};
struct B: A {   //等于struct B:public A
    B(int x):A(x) { }
};
struct C {
    C( ) { }
};
struct D: virtual A, C {
    D(int x):A(x) { }   //同样调用C( )
};
struct E: B, D {
    E(int x):A(x), B(x+5), D(x+10) { }
};
```

```
void main(void) {
    E  e(0);
//  cout << "a=" << e.a;  //二义性访问
    cout << "a=" << e.B::a;
    cout << "a=" << e.D::a;
}
```

为解决 e.a 产生的二义性，要么将E的基类B说明为对象成员，要么将B的基类A说明为虚基类。若将B的基类A说明为虚基类，则 e.a、e.B::a及e.D::a 都表示虚基类A的成员 a 。

第9章　多继承与虚基类

◆9.3 派生类成员 (类成员同名问题)

- 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。
  - 【例9.2】基类成员间的同名问题。
  - 【例9.4】派生类成员与基类成员同名问题。
  - 【例9.5】虚基类与基类的成员同名问题。

第9章　多继承与虚基类

◆9.4 单重及多重继承的构造与析构

- 在考虑多继承派生类构造函数的执行顺序时，必须注意派生类可能有虚基类、基类、对象成员、const成员以及引用成员。当虚基类、基类和对象成员只有带参数的构造函数时，派生类必须定义自己的构造函数，而不能利用C++提供的缺省构造函数。类有非静态对象成员、const 成员时，也必须定义构造函数。
- 对于虚基类、基类和对象成员来说，如果它们没有定义自己的构造函数，则编译程序就会为它们提供缺省的无参构造函数。对于虚基类、基类和对象成员的无参构造函数，无论它们是自定义的还是由编译程序提供的，可被派生类构造函数按定义顺序自动地调用。

第9章　多继承与虚基类

◆9.4 单重及多重继承的构造与析构

- 派生类对象的构造顺序描述：
  - ① 按定义顺序自左至右、自下而上地构造所有虚基类；
  - ② 按定义顺序构造派生类的所有直接基类；
  - ③ 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；
  - ④ 执行派生类自身的构造函数体；
- 如果虚基类、基类、对象成员、const成员以及引用成员又是派生类对象，重复上述派生类对象的构造过程，但同名虚基类对象在同一棵派生树中仅构造一次。
- 析构派生类对象的顺序同构造逆序。

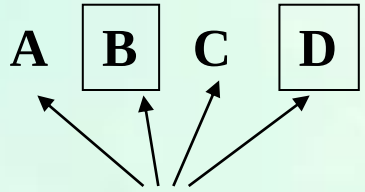
第9章　多继承与虚基类

【例9.6】多继承派生类的构造过程

```
#include <iostream>
using namespace std;
struct A {
    A( ) { cout<<'A'; }
};
struct B {
    B( ) { cout<<'B'; }
};
struct C {
    int a;   int &b;
    const int c;
    C(char d): c(d), b(a)
    { a = d; cout << d; }
};
struct D {
    D( ) { cout << 'D';}
};
```

```
struct E: A, virtual B, C, virtual D {
    A x, y;
    B z;
    E( ): z( ), y( ), C('C')
    {
        cout << 'E';
    }
};

void main(void)
{
    E  e;
}
```



E{ x(A), y(A), z(B)}

输出：

BDACAABE

第9章　多继承与虚基类

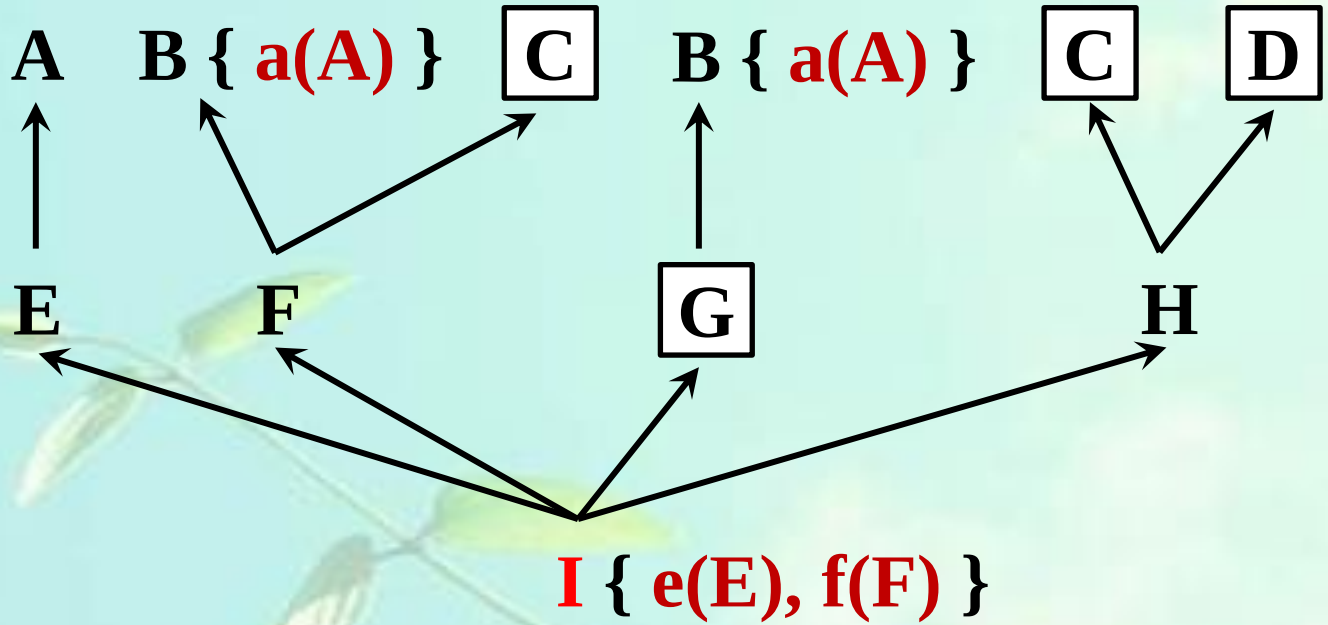
【例9.7】多继承派生类的构造过程。

```
#include <iostream>
using namespace std;
struct A { A( ) { cout << 'A’; } };
struct B { const A a;  B( ) { cout << 'B’; } };
struct C { C( ) { cout << 'C’; } };
struct D { D( ) { cout << 'D’; } };
struct E：  A { E( ) { cout << 'E’; } };
struct F：  B, virtual C { F( ) { cout << 'F’; } };
```

第9章　多继承与虚基类

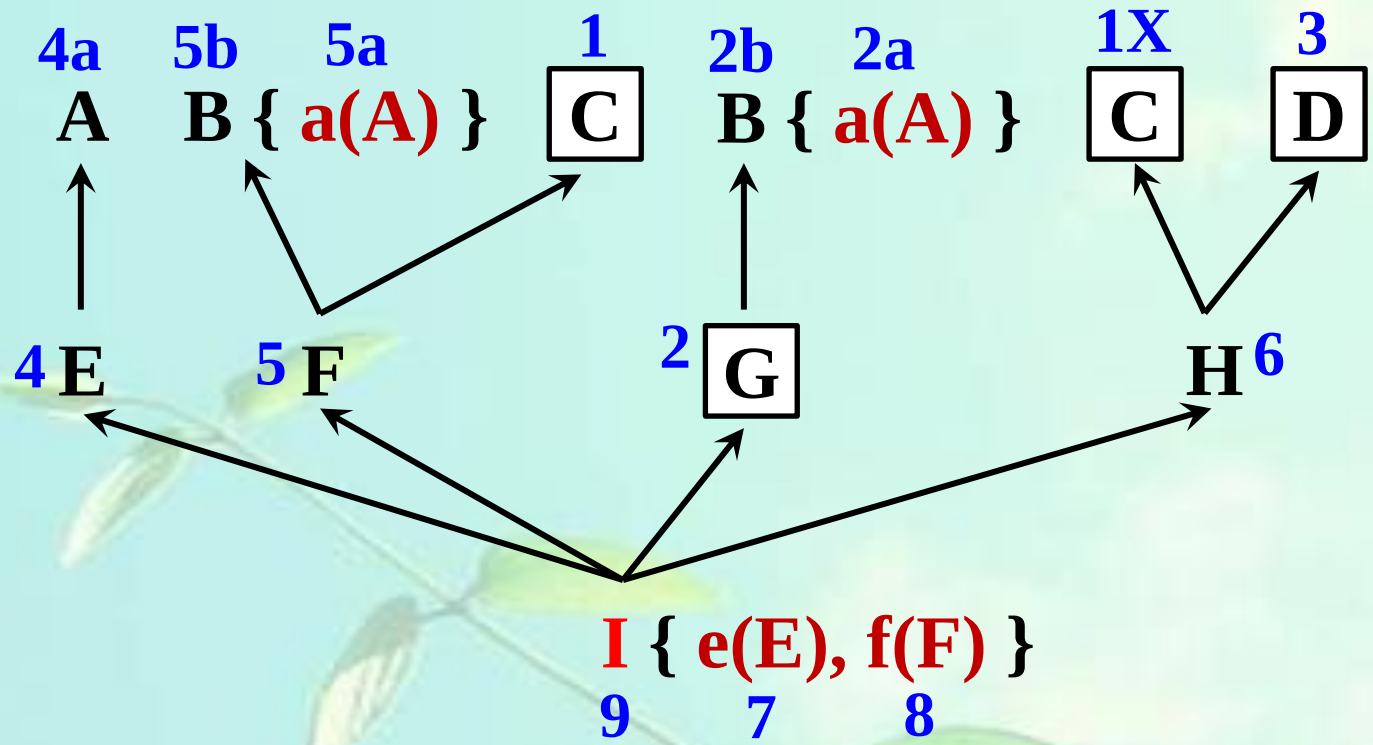
```
struct G：B { G( ) { cout << 'G’; } };
struct H：virtual C, virtual D
{  H( ) { cout << 'H’;  } };
struct I：E, F, virtual G, H {
    E  e;
    F  f;
    I( ): f( ), e( ) , F( ), E( ) { cout << 'I’; }
};
void main(void) { I  i; }
```

第9章　多继承与虚基类



例9.7六棵派生树 (根红色)

第9章　多继承与虚基类



例9.7六棵派生树(根红色)，输出：CABGDAEABFHAECABFI

第9章　多继承与虚基类

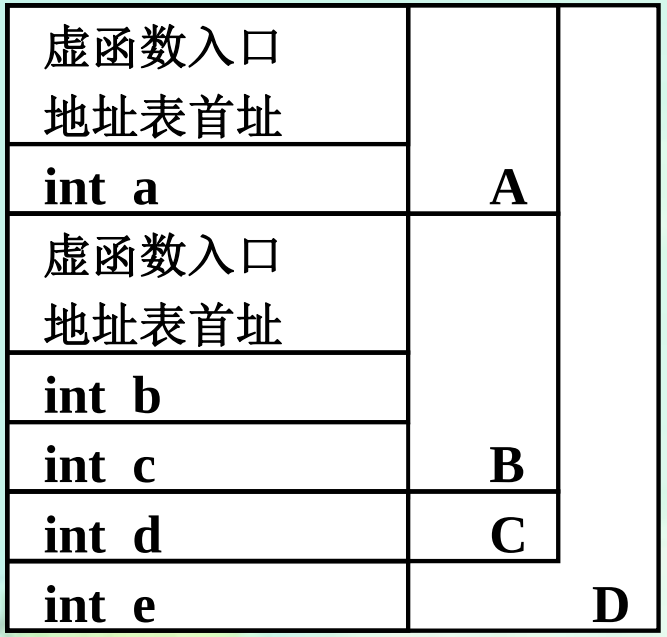
◆9.5 类的存储空间

- **派生类无虚基类的情况下：**
  - 若派生类的第一个基类建立了虚函数入口地址表（VFT），则派生类就共用该表首址所占用的存储单元；
  - 若派生类的第一个基类没有定义虚函数，派生类就在建立完所有基类的存储空间之后，根据派生类中是否定义了新的虚函数，确定是否为VFT表首址分配一个存储单元，然后为新定义的数据成员建立存储空间。
  - 静态数据成员不包括在内。

第9章　多继承与虚基类

【例9.8】无虚基类的多继承派生类存储空间的建立

```
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( )  {  };
};
```



派生类D的存储空间示意图

D的第1个基类A已建立了VFT首址。D共用该表首址所占用的存储单元。

sizeof(D) = sizeof(A) + sizeof(B) + sizeof(C) + sizeof(e)

sizeof(A) = sizeof(void *) + sizeof(a)

sizeof(B) = sizeof(void *) + sizeof(b) + sizeof(c)

sizeof(C) = sizeof(d)

第9章　多继承与虚基类

◆9.5 类的存储空间

- **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。

第9章　多继承与虚基类

◆9.5 类的存储空间

- **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：

  ③派生类依次处理自定义的数据成员，为每个数据成员建立相应的存储空间。

  ④派生类根据虚基类偏移的建立顺序，依次为虚基类建立存储空间，同名虚基类仅在派生类存储空间内建立一次。

  ⑤如果直接基类和虚基类又是派生类，则在派生类的存储空间内重复步骤①至⑤。如果数据成员又为派生类类型，则在数据成员的存储空间内重复步骤①至⑤。

第9章　多继承与虚基类

【例】含有虚基类的多继承派生类存储空间的建立

```
#include <iostream>
using namespace std;
struct A  {
    virtual void fa( )  {  };
};
struct B  {
    int  b;
    void fb( );
};
struct E: virtual A  {
    int  x;
    virtual void fe( )  {  };
};
struct F: virtual A, virtual B  {
    int  x;
    void ff( )  {  };
};
struct G: B, virtual A  {
    int  x;
    virtual void fg( )  {  };
};
```