移动平台应用软件开发
C++面向对象基础

继承

主讲：张齐勋

zhangqx@ss.pku.edu.cn

《移动平台应用软件开发》课程建设小组

北京大学

二零一五年

北京大学

提纲

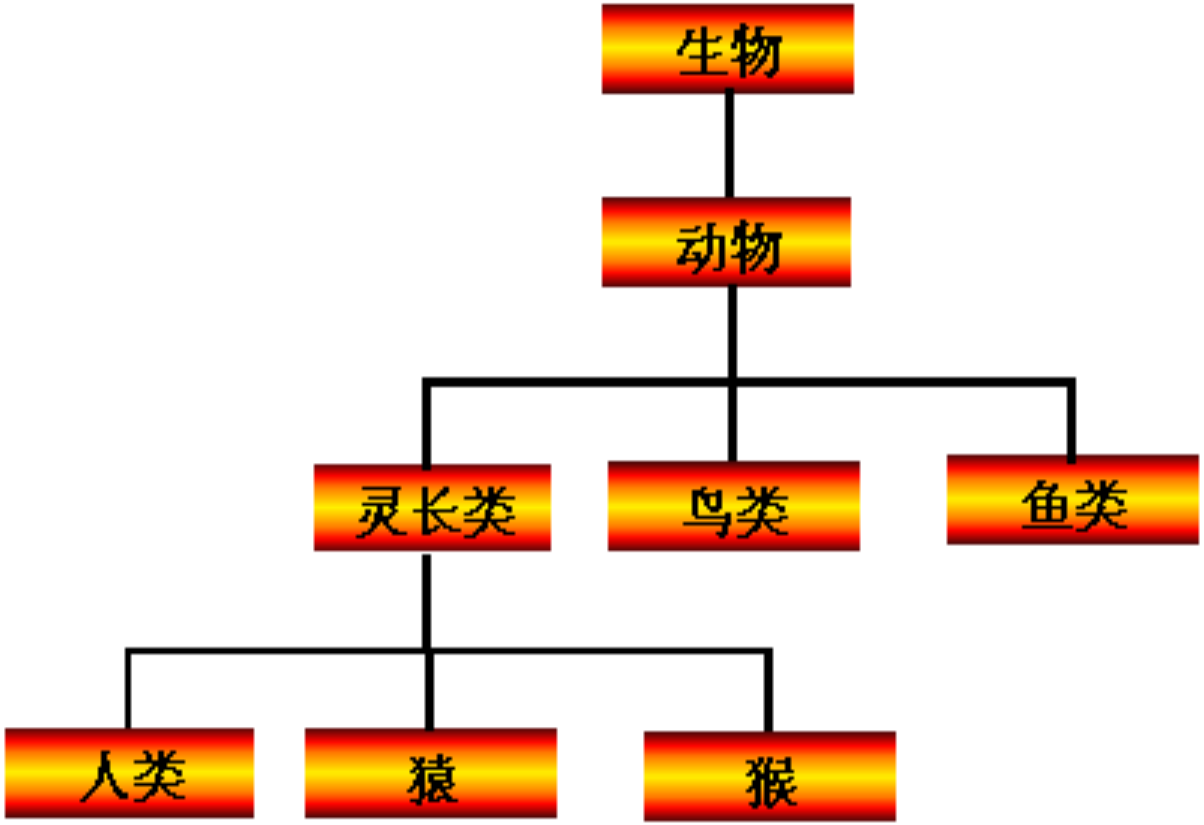
一、基类和派生类

二、单继承

三、多继承

继承

继承是面向对象程序设计的第二个主要特征.通过继承实现了在数据抽象基础上的代码重用.

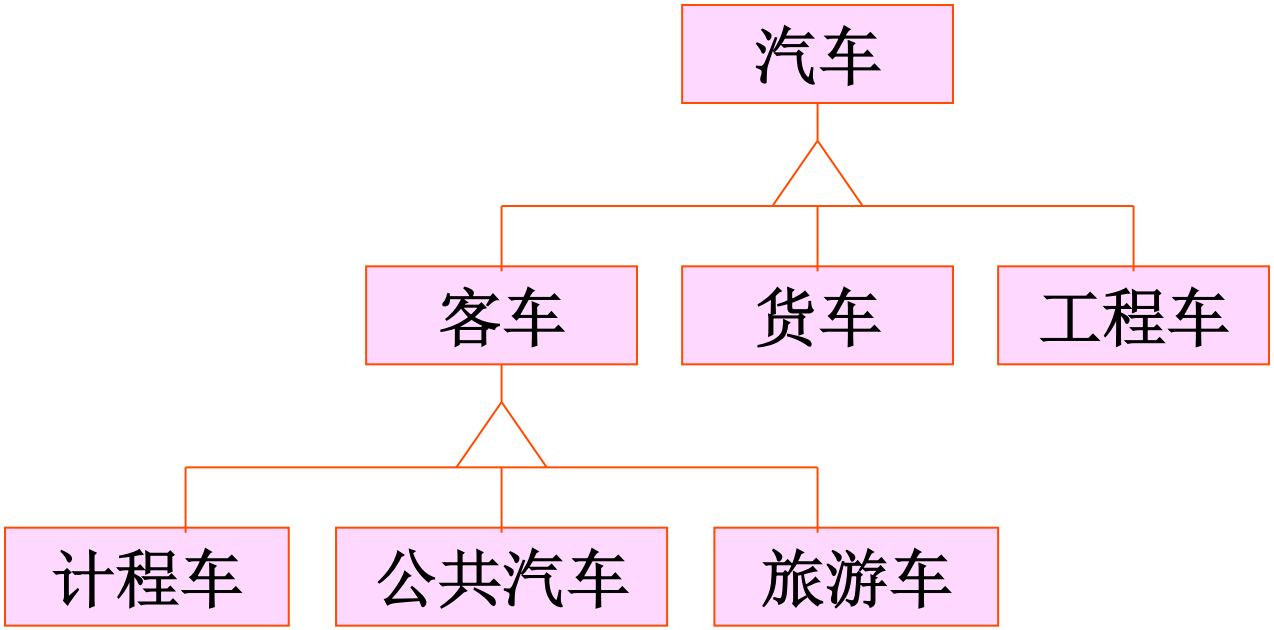


继承的基本概念

继承是一种类与类之间的关系，这种关系允许在既有类的基础上创建新的类。也就是说，定义新类时可以从一个或多个既有类中继承（即拷贝）所有的数据成员和函数成员，然后加上自己的新成员或重新定义由继承得到的成员。

继承形成一种类的层次关系，既有类成为基类或父类，以它为基础建立的新类称为派生类或子类。

继承是C++实现软件重用的主要手段。



交通工具类层次关系图

一、基类和派生类

- 继承是C++面象对象程序设计的重要特性之一，它是指建立一个新的派生类，从一个或多个先前定义的类中继承数据和函数，而且可以重新定义或加进新数据和函数，从而建立了类的层次或等级。

- 通过继承机制，可以利用已有的数据类型来定义新的数据类型。所定义的新的数据类型不仅拥有新定义的成员，而且还同时拥有旧的成员。我们称已存在的用来派生新类的类为基类，又称为父类。由已存在的类派生出的新类称为派生类，又称为子类。

派生(Derivate)类的定义格式

派生类的定义格式为：

```
class 派生类名：  继承方式    基类名
{
        //派生类新增成员定义
        …
};
```

“继承方式”用于规定派生类中由基类继承到的那部分成员在派生类中的访问控制权限。继承方式用下述三个关键字之一来指定：**public**：公有继承；**protected**：保护继承；**private**：私有继承。

派生类对象组成

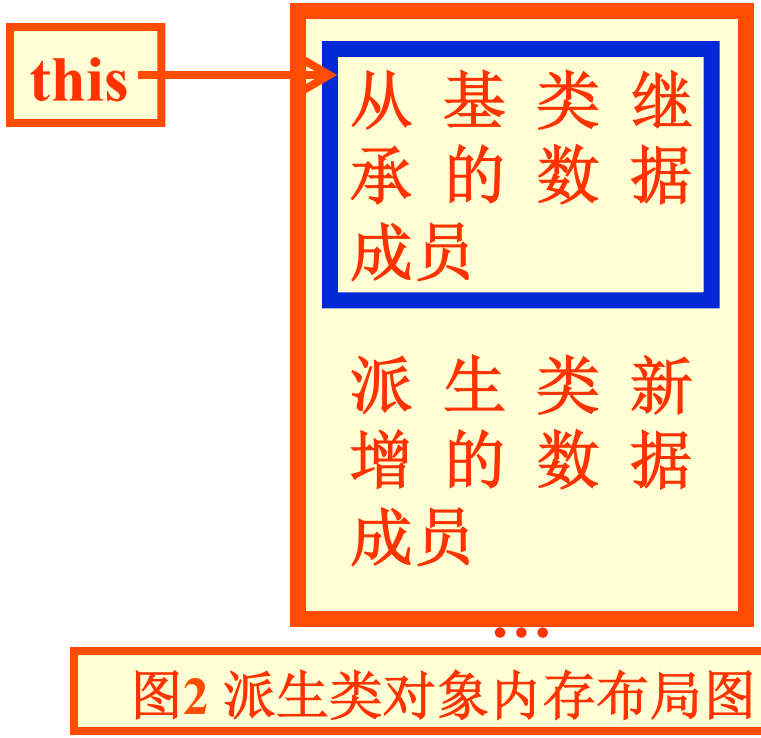


图2 派生类对象内存布局图

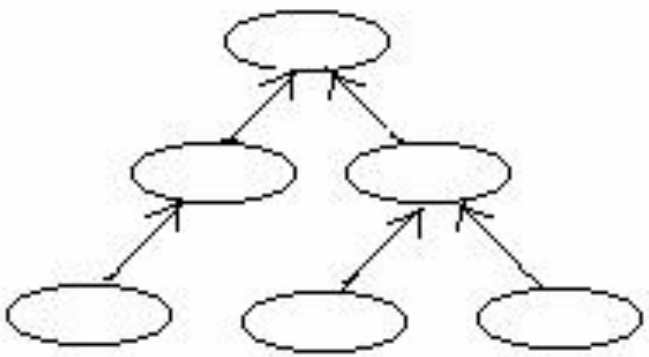
派生类继承了基类的除了构造函数和析构函数之外的所有成员，因此派生类对象由两部分组成：一部分是由基类继承的成员，另一部分是派生类新增加的自己特有的成员。

```
class table{
    public:
        int table_top;
        int table_leg[4];
};

class my_table: public table {
     public:
        int table_cloth;
};

void main(){ }
```

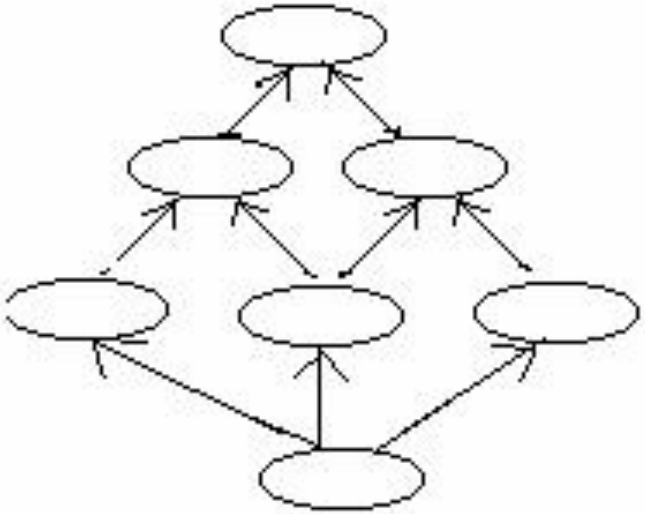
问：此时，my_table有几个成员？

继承方式	基类特性	派生类特性
Public 继承	public	public
	protected	protected
	private	不可访问
Private 继承	public	private
	protected	private
	private	不可访问
Protected 继承	public	protected
	protected	protected
	private	不可访问

- 在C++语言中，一个派生类可以从一个基类派生，也可以从多个基类派生。从一个基类派生的继承称为单继承；从多个基类派生的继承称为多继承。



单继承



多继承

```
class <派生类名>:<继承方式1><基类名1>,<
    继承方式2><基类名2>,...
{
    <派生类新定义成员>
};
```

多继承与单继承的区别从定义格式上看，主要是多继承的基类多于一个。

基类与派生类的关系

任何一个类都可以派生出一个新类，派生类也可以再派生出新类，基类和派生类是相对而言的。

基类与派生类之间的关系可以有如下几种描述:

1. 派生类是基类的具体化

类的层次通常反映了客观世界中某种真实的模型。在这种情况下，不难看出：基类是对若干个派生类的抽象，而派生类是基类的具体化。基类抽取了它的派生类的公共特征，而派生类通过增加行为将抽象类变为某种有用的类型。

2. 派生类是基类定义的延续

先定义一个抽象基类，该基类中有些操作并未实现。然后定义非抽象的派生类，实现抽象基类中定义的操作。

3. 派生类是基类的组合

在多继承时，一个派生类有多于一个的基类，这时派生类将是所有基类行为的组合。

继承的机制将使得在创建新类时，只需说明新类与已有类的区别，从而大量原有的程序代码都可以复用，这便是人们所常说的“可复用的软件构件”。

二、单继承

- 在单继承中，每个类可以有多个派生类，但是每个派生类只能有一个基类，从而形成树形结构。

单继承—成员访问权限的控制

```
class A
   {
   public:
        void f1();
   protected:
        int j1;
   private:
       int i1;
   };
```

```
class B:public A
 {  public:
        void f2();
    protected:
        int j2;
    private:
        int i2;
 };
 class C:public B
 {  public:
        void f3();
 };
```

f2()可以访问f1(),j1,不能访问i1;

派生类B的对象b1可以访问f1(),不能访问 j1 ,i1;

f3()可以访问f1(),j1,f2(),j2,不能访问i1,i2;派生类C的对象c1只能访问f1(),f2(),不能访问其它的。

在公有继承时，派生类的成员函数可以访问基类中的公有和保护成员，派生类的对象只能访问基类中的公有成员。

公有继承的例子

```
#include<iostream.h>
class A{
private:
        int x;
public:
        void Setx (int i) { x = i ;}
        void Showx ( ){ cout<<x<<endl ;}
};
class B:public A{
private:
        int y ;
public:
        void Sety (int i) { y = i ;}
        void Showy ( )
        {  Showx ( ) ;
           cout<<y<<endl ; }
};
```

```
void main ( )
{
        B  b ;
        b.Setx (10) ;
        b.Sety (20) ;
        b.Showy ( ) ;
}
```

执行该程序，输出结果如下：

```
10
20
```

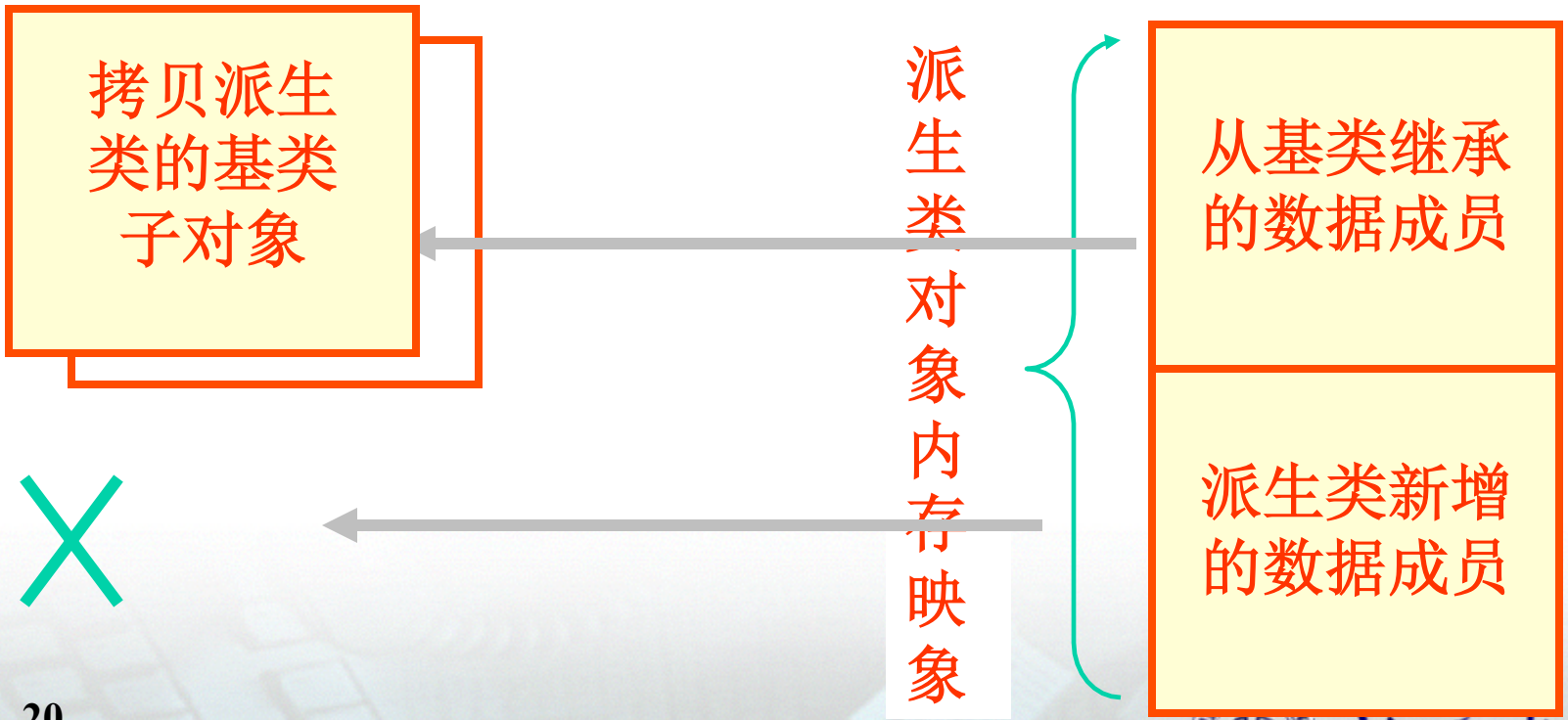
```
#include<iostream.h>
class A{ // 定义基类
        int x ;    // 基类私有成员
public:
        void setx(int i) { x = i ; }              // 基类公有成员函数
        void showx( ) { cout<<x<<endl ;} // 基类公有成员函数
};
class B: private A{          // 定义一个派生类,继承方式为私有继承
        int y ;
public:
        void  setxy( int m,int n)
        {
           setx(m);          //派生类成员函
              y = n ;
        }
        void showxy( )
        { showx( ) ; cout<<y<<endl ; }
};
```

```
void main( ){
  A  a;
  a.setx(5);
  a. showx();
  B  b;// 定义派生类对象b
  b.setxy(10,20);
  b.showx();
//访问派生类的私有成员，非法！
    b.showxy( )
}
```

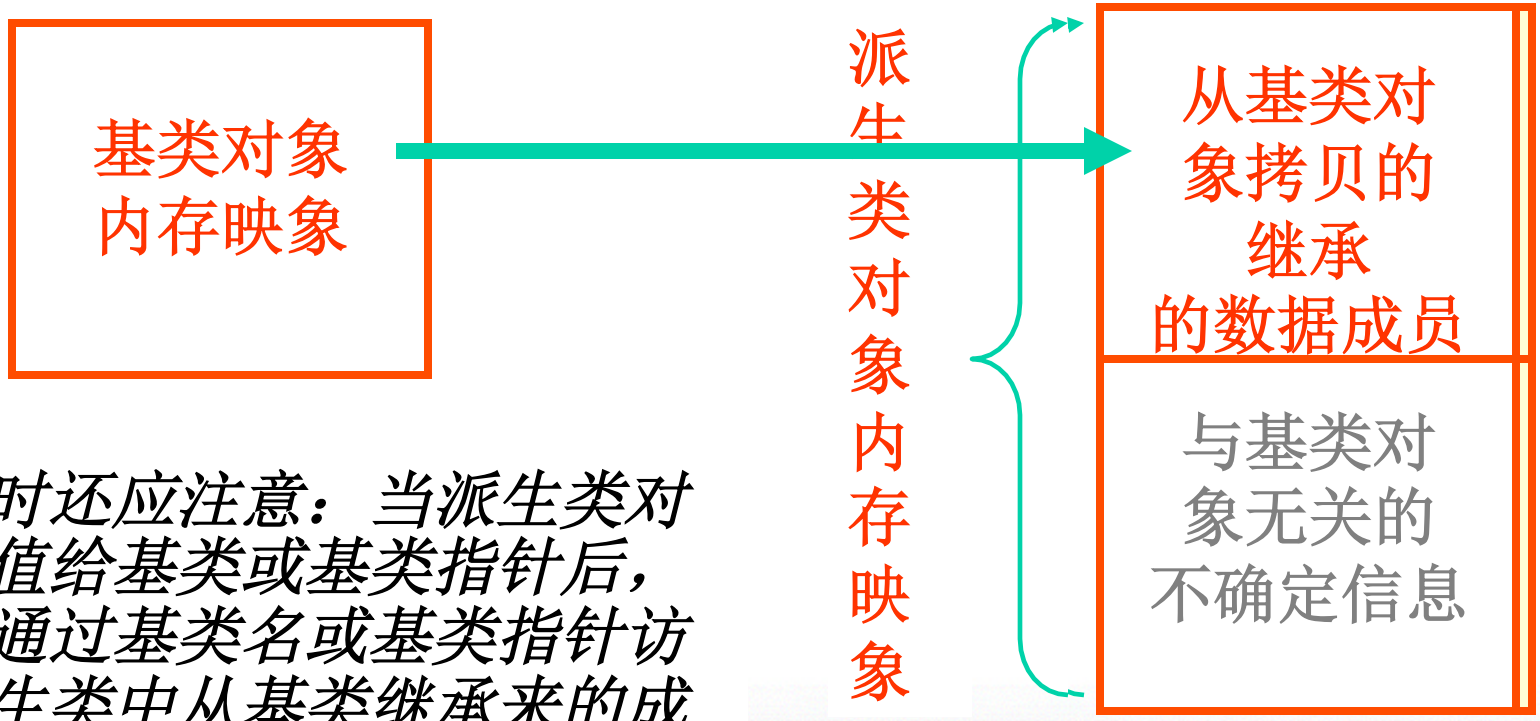
公有继承下的赋值兼容规则

所谓赋值兼容规则指的是不类型的对象间允许相互赋值的规定。面向对象程序设计语言中，在公有派生的情况下，允许将派生类的对象赋值给基类的对象，但反过来却不行，即不允许将基类的对象赋值给派生类的对象。这是因为一个派生类对象的存储空间总是大于它的基类对象的存储空间。若将基类对象赋值给派生类对象，这个派生类对象中将会出现一些未赋值的不确定成员。

允许将派生类的对象赋值给基类的对象。



不允许将基类的对象赋值给派生类的对象。



使用时还应注意：当派生类对象赋值给基类或基类指针后，只能通过基类名或基类指针访问派生类中从基类继承来的成员，不能访问派生类中的其它成员。

构造函数

派生类的对象的数据结构是由基类中说明的数据成员和派生类中说明的数据成员共同构成。将派生类的对象中由基类中说明的数据成员和操作所构成的封装体称为基类子对象，它由基类中的构造函数进行初始化。

构造函数不能够被继承，因此，派生类的构造函数必须通过调用基类的构造函数来初始化基类子对象。所以，在定义派生类的构造函数时除了对自己的数据成员进行初始化外，还必须负责调用基类构造函数使基类数据成员得以初始化。如果派生类中还有子对象时，还应包含对子对象初始化的构造函数。

派生类构造函数的一般格式如下：

<派生类名>(<派生类构造函数总参数表>):<基类构造函数>(参数表1),<子对象名>(<参数表2>)

{

<派生类中数据成员初始化>

};

派生类构造函数的调用顺序如下：

- 基类的构造函数
- 子对象类的构造函数(如果有的话)
- 派生类构造函数

执行的顺序是：先祖先(基类)，再客人(成员对象)，后自己(派生类)。

派生类构造函数实例

```
#include<iostream.h>
class A{                                    //定义基类
private:
        int a ;
public:
   A(int x) { a = x ;cout<<"A's constructor called."<<endl ; }
   void show( ) { cout<<a<<endl;}
};

class B{                                    //定义另一个类
private:
        int b ;
public:
   B(int x) { b = x ;cout<<"B's constructor called."<<endl ; }
   int get( ) { return b;}
};
```

派生类构造函数实例

```
class C : public A{                          //定义派生类
private:
        int c;
        B obj_b ;
public:
        C( int x ,int y , int z):A(x),obj_b(y)// 派生类构造函
        {
                c = z ;
                cout<<"C's constructor called."<<endl ;
        }
         void show( ) {
                A::show( );
                cout<<obj_b.get()<<","<<c<<endl ;
        }
};
```

```
void main( ){
        C  c1(1,2,5),  c2(3,4,7);
        c1.show( ) ;
        c2.show( ) ;
}
```

程序输出如下：

```
A's constructor called.
B's constructor called.
C's constructor called.
A's constructor called.
B's constructor called.
C's constructor called.
1
2,5
3
4,7
```

析构函数

当对象被删除时，派生类的析构函数被执行。由于析构函数也不能被继承，因此在执行派生类的析构函数时，基类的析构函数也将被调用。

在派生类中是否定义析构函数与基类无关。若派生类对象在退出其作用域前,有数据需要做善后工作,就需要定义析构函数。基类的析构函数不会因派生类没有析构函数而得不到执行，它们各自是独立的。 若基类、成员类、派生类都有析构函数，则执行的顺序是：先自己(派生类)，再客人(成员对象)，后祖先(基类)。其顺序与执行构造函数时的顺序正好相反。

派生类析构函数实例

```
#include<iostream.h>
class X{
        int x1,x2 ;
public:
        X (int i,int j) {x1 = i ; x2 = j ;}
        void print ( ) { cout<<x1<<",”<<x2<<endl ; }
        ~X( ) {cout<<”X’s destructor called.”<<endl; }
};
class Y:public X{
        int y ;                 // 派生类Y新增数据成员
public:
        Y( int i , int j , int k) : X(i , j)        { y = k ;}
造函数
        void print( ) {X::print ( ) ;cout<<y<<endl ;}
        ~Y( ) { cout<<”Y’s destructor called.”<<endl ; }
};
```

```
void main( )
{
    Y y1(5,6,7) ,
    y2(2,3,4) ;
    y1.print( ) ;
    y2.print ( ) ;
}
```

程序输出结果如下：

```
5，6
7
2，3
4
Y’s destructor called.
X’s destructor called.
Y’s destructor called.
X’s destructor called.
```

派生类构造函数使用的注意事项

1）派生类构造函数中可以省略对基类构造函数的调用，其条件是在基类中必须有缺省的构造函数，或者根本没有定义构造函数。

2）若基类构造函数有参数，则派生类必须定义构造函数，提供将参数传递给基类构造函数。

继承成员的调整

类的继承机制允许程序员在保持原有类特性的基础上，进行更具体、更详细的类的定义。

派生新类的过程一般包括吸收基类的成员、调整基类成员和添加新成员三个步骤。这里讨论调整和改造基类成员的问题。

对基类成员进行改造、调整，包括两个方面：一是基类成员的访问控制方式，二是对基类成员的重新定义。

恢复访问控制方式

若希望基类某些成员的访问控制方式在private或protected继承方式下，在派生类中的身份不要改变，保持在基类中声明的访问控制方式，这可通过恢复访问控制方式的声明来实现。声明格式如下：

访问控制方式：

基类名：：基类成员名；

其中，访问控制方式是protected、public之一，基类成员名是要恢复访问控制权限的基类成员名，对数据成员和函数成员都一样，只写成员名即可。这种声明是在派生类中针对基类的某个成员进行的。

例 恢复访问控制方式

```
#include<iostream.h>
class A{
public:
        void f(int i) {cout<<i<<endl ;}
        void g( ) { cout<<”g”<<endl ;}
};
class B: private A{          // 以私有继承方式派生新类B
public:
        void h( ) { cout<<”h”<<endl ;}
        A::f;
// 恢复从基类继承的成员函数f（）的访问控制权限
};
void main( )
{
   B b;
   b.f(6);
//在类B的外部通过派生类对象b访问继承的成员函数f( )
b.h( ) ;
}
```

说明：派生类B中对继承的成员函数f( )进行了恢复访问控制方式的声明，使得该函数在派生类B中仍为公有成员，若没有这种声明，则按继承方式private，在main（）中是不能访问f（）函数的。

继承成员的重命名和重定义

在C++的类继承时，首先是将基类的成员全盘接收。这样，派生类就包含了除了构造函数和析构函数之外的基类所有成员。在程序中有时需要对基类成员进行改造或调整。在派生类中对继承成员可以**重命名（rename）或重定义（override）**。

继承成员的重定义

```
#include<iostream.h>
class Point{          // 定义基类Point
        int x , y ;
public:
        Point( int xx , int yy ) {x = xx
        void move(int m , int n) { x+=
        int area( ) { return 0 ;}
        int Getx( ) { return x ;}
        int Gety( ) { return y ; }
};
class Rectangle: public Point{
        int w , h ;                    //
public:
        Rectangle ( int a , int b , int c
        { w = c ; h = d ;}
        void shift ( int i , int j)
        { Point::move(i , j) ; }       //
        int area( ) { return w*h ; }   //
        int Getw( ) { return w ; }  //派
        int Geth( ) { return h ; }
};
```

```
void main ( ){
    Rectangle rect(2,3,20,10);
    rect.shift(3,2);
    cout<<"The data of rect(x,y,w,h):"<<endl ;
    cout<<rect.Getx( )<<","<<rect.Gety( )<<"," ;
    cout<<rect.Getw ( )<<","<<rect.Geth( )<<endl;
    cout<<"The area of that rect is :"
    cout <<rect.area( )<<endl;
}
```

执行该程序，输出结果如下：

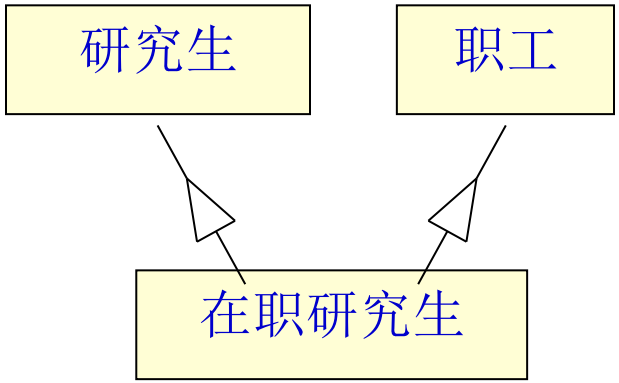
```
The data of rect(x,y,w,h):
5,5,20,10
The area of that rect is :200
```

由此例可见，重命名使派生类可以用不同的名字调用基类中定义的函数。而重定义使派生类与基类的同名函数有不同的执行版本。在调用派生类的公有成员函数时，首先在派生类新定义的成员中寻求匹配，若派生类新定义成员中没有请求的函数，则从直接基类开始在上层基类中寻求匹配。直到找到该函数为止。

三、多继承

多继承的概念

若一个派生类具有两个或两个以上基类，这种继承称为多继承。



多继承

正如生活中一个人可以同时继承其父母双方的特点一样。

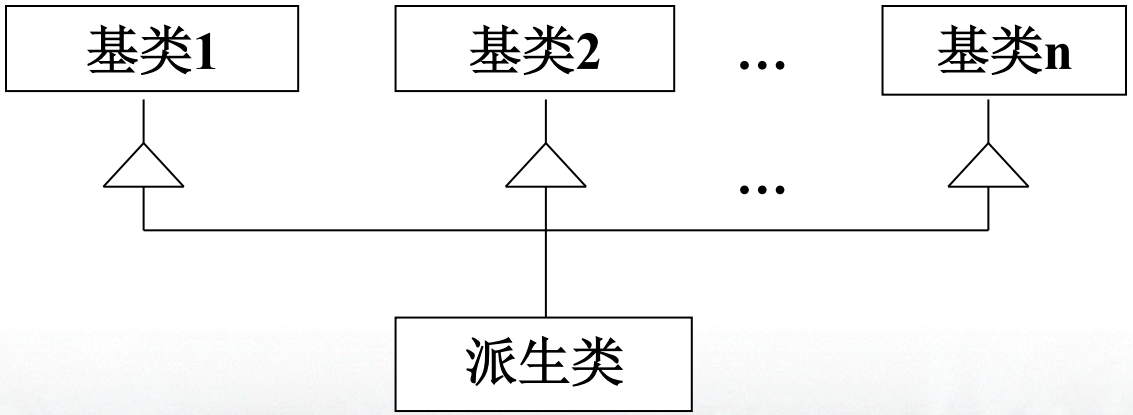
多继承派生类的定义格式

class派生类名：继承方式1 基类名1， 继承方式2 基类名2，…
{ …； //派生类新增成员的定义 };

对每个基类可以采用不同的继承方式，缺省继承方式是private。

例如：

class z：public x，y //类z公有继承了类x，私有继承了类y
{ …； };



多继承的派生类与多个基类的关系

例如:

```
class A
{
   …
};
class B
{
   …
};
class C : public A, public, B
{
   …
};
```

其中，派生类C具有两个基类(类A和类B)，因此，类C是多继承的。按照继承的规定，派生类C的成员包含了基类A,B中成员以及该类本身的成员。

多继承派生类的构造函数

在多继承的情况下，派生类的构造函数格式为：
派生类名：：派生类名（〈总参数表〉）：基类名1（〈参数表1〉），…,基类名n（〈参数表n〉），…,
子对象1（〈参数表n+1〉），…
{
　　…；//派生类构造函数函数体
}

多继承派生类构造函数执行顺序是：
(1)　所有基类的构造函数；多个基类构造函数的执行顺序取决于定义派生类时所指定的顺序，与派生类构造函数中所定义的成员初始化列表的参数顺序无关。
(2)　子对象（如果有的话）类的构造函数；
(3)　派生类本身构造函数的函数代码。

例 多继承派生类的构造函数

```
#include<iostream.h>
class A1{                                   // 定义基类A1
        int a1;
public:
        A1(int i){a1=i;cout<<"constructor A1."<<a1<<endl;}
        void print( ){cout<<a1<<endl;}
};
class A2{                                   // 定义基类A2
        int a2;
public:
        A2(int i){a2=i;cout<<"constructor A2."<<a2<<endl;}
        void print(){cout<<a2<<endl;}
};
class A3{                                   // 定义A3
        int a3;
public:
        A3(int i){a3=i;cout<<"constructor A3."<<a3<<endl;}
        int geta3( ){return a3;}
};
```

```
class B:public A2 , public A1{// 定义派生类 B，基类为A1和A2
        int b;
        A3 obj_a3;  //对象成员
public:
    B(int i,int j,int k,int l):A1(i),A2(j), obj_a3 (k)//派生类构造函数
       { b=l;
        cout<<"constructor B."<<b<<endl;
    }
    void print()
    {   A1::print();
        A2::print();
        cout<<obj_a3. geta3()<<","<<b<<endl;
    }
};
void main( )
{       B bb(1,2,3,4);
        bb.print();
}
```

执行该程序，输出结果如下：

```
constructor A2.2
constructor A1.1
constructor A3.3
constructor B.4
1
2
3,4
```

说明：

(1)程序中派生类B的构造函数有四部分组成，分别完成基类A1、A2，派生类子对象aa，B新增数据成员b的初始化工作。该函数的总参数表中有4个参数，分别是基类A1、A2，派生类子对象obj_a3以及派生类B构造函数的参数。

(2)在构造函数的成员初始化列表中，两个基类顺序是A1在前，A2在后，而定义派生类B时两个基类顺序是A2在前，A1在后。从输出结果中可以看出：先执行A2的构造函数，后执行A1的构造函数。因此，执行基类构造函数的顺序取决于定义派生类时指定的基类的顺序，而派生类构造函数的成员初始化列表中各项顺序可以任意排列。

(3)作用域运算符：：用于解决作用域冲突问题。

在派生类B中的print（）函数体中，使用A1：：print（）；和A2：：print（）；分别指明调用哪一个类中的print（）函数。

(4)派生类对从基类继承的某个函数重新定义后，当在派生类中调用基类的同名函数时会出现二义性，例如，对上述print函数以下面形式调用时：

```
void print(){
        print();      // 对print（）的调用有二义性
        cout<<obj_a3.geta3()<<","<<b<<endl;
}
```

则系统分不清print（）是哪一个基类的。对这种二义性问题应该用作用域运算符：：来解决。

Q&A

本讲结束

北京大学