Lab of
New Generation Network
Techonology & Applications

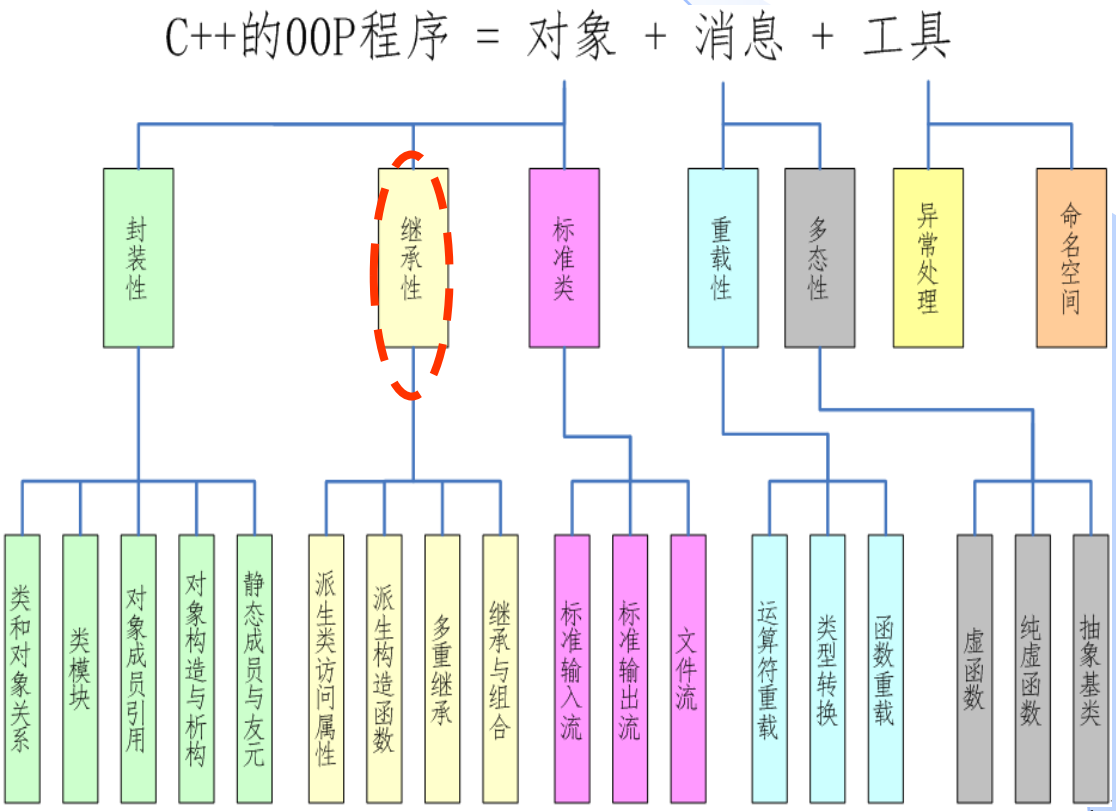
第7讲 继承与派生（下）

7.1 派生类的构造函数

7.2 多重继承

7.3 继承与组合

参考教材第16章内容



回顾类的构造函数

2.1 构造函数

◆ 用参数初始化表对数据成员初始化

```
#include <iostream>
using namespace std;
class Box
{public:
  Box(int h,int w ,int len):height(h),width(w),length(len){}
  int volume();
 private:
  int height;
  int width;
  int length;
};

 int Box::volume()
{return(height*width*length);
}

int main()
{
 Box box2(15,30,25);
 cout<<"The volume of box2 is "<<box2.volume()<<endl;
 return 0;
}
```

```
C:\WINDOWS\system32\cmd.exe
The volume of box2 is 11250
请按任意键继续. . .
```

```
#include <iostream>
using namespace std;
class  A
{
public:
   A(int i ,int j)
   {a1=i;
   a2=j;}
void print()
{cout<<a1<<","<<a2<<endl;}
private:
   int a1,a2; };

class B
{public:
B(int i,  int j, int k): a(i,j), b(k){}
void print();
private:
 A a;
 int b;};

void B:: print()
{ a.print();
  cout<<b<<endl;}
void main()
{ B b (6,7,8);
  b.print();}
```

```
C:\WINDOWS\system32\cmd.exe
6 ,7
8
请按任意键继续. . .
```

7.1 派生类的构造函数和析构函数

- 派生类的构造函数不能继承；
- 其构造函数不仅要考虑所增加数据成员的初始化，还应当考虑基类继承数据成员初始化；
- 方法：通过派生类的构造函数采用初始化参数表来调用基类的构造函数，对基类数据成员初始化；
- 执行顺序：①先调用基类构造函数；②再执行派生类构造函数本身；即“长辈优先”原则；
- 在派生类对象释放时，先执行派生类析构函数~Student1( )，再执行其基类析构函数~Student( )；即“身先士卒”原则。

```
#include <iostream>
#include<string>
using namespace std;
class Student//声明基类Student
 {public:
  Student(int n,string nam,char s)          //基类构造函数
   {num=n;
    name=nam;
    sex=s; }
   ~Student( ){ }                           //基类析构函数
  protected:                                //保护部分
   int num;
   string name;
   char sex ; };

class Student1: public Student      //声明派生类Student1
 {public:                           //派生类的公用部分
  Student1(int n,string nam,char s,int a,string ad):Student(n,nam,s) //派生类构造函数
   {age=a;          //在函数体中只对派生类新增的数据成员初始化
    addr=ad;}
  void show( )
   {cout<<"num:"<<num<<endl;
    cout<<"name: "<<name<<endl;
    cout<<"sex:"<<sex<<endl;
    cout<<"age:"<<age<<endl;
    cout<<"address: "<<addr<<endl<<endl;}
  ~Student1( ){ }   //派生类析构函数
 private:           //派生类的私有部分
  int age;
  string addr;};

int main( )
 {Student1 stud1(10010,"Wang-li",'f',19,"115 Beijing Road,Shanghai");
  Student1 stud2(10011,"Zhang-fun",'m',21,"213 Shanghai Road,Beijing");
  stud1.show( );                    //输出第一个学生的数据
  stud2.show( );                    //输出第二个学生的数据
  return 0;}
```

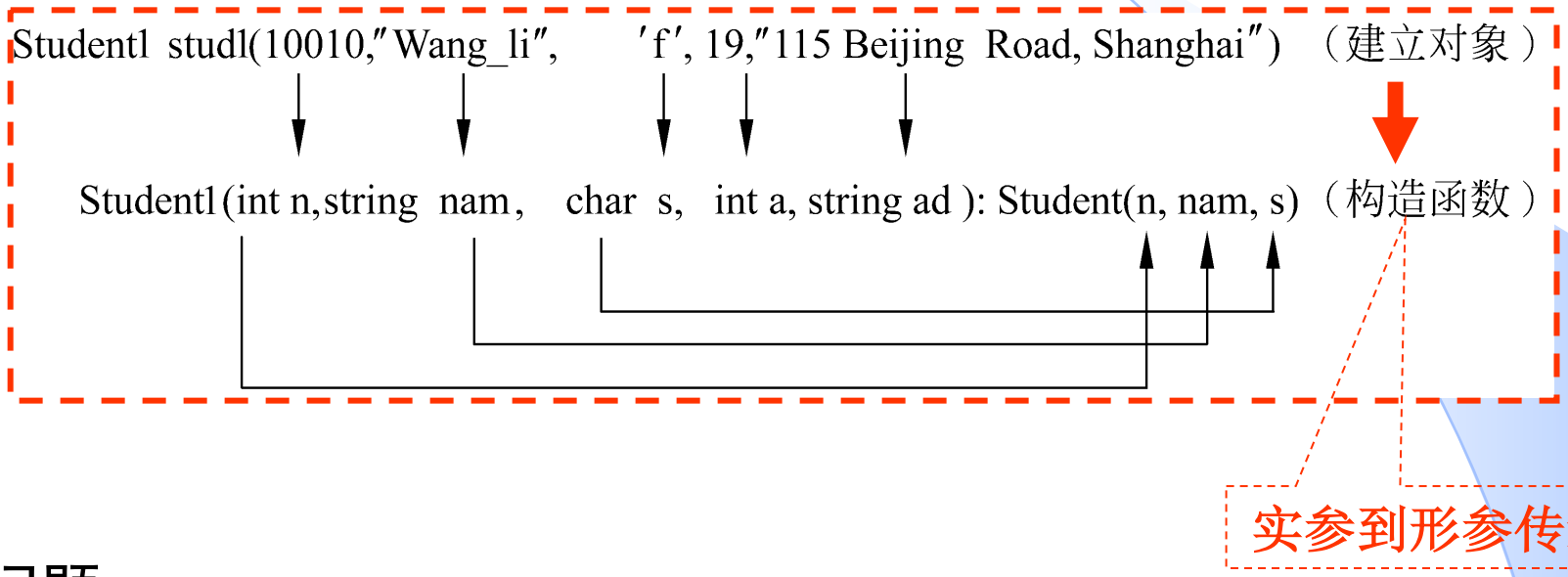
基类函数调用

```
C:\WINDOWS\system32\cmd.exe
num:10010
name: Wang-li
sex:f
age:19
address: 115 Beijing Road,Shanghai

num:10011
name: Zhang-fun
sex:m
age:21
address: 213 Shanghai Road,Beijing

请按任意键继续. . .
```

- 派生类构造函数首行一般格式

  派生类构造函数名（总参数表列）：基类构造函数名（参数表列）

  {派生类中新增数据成员初始化语句}



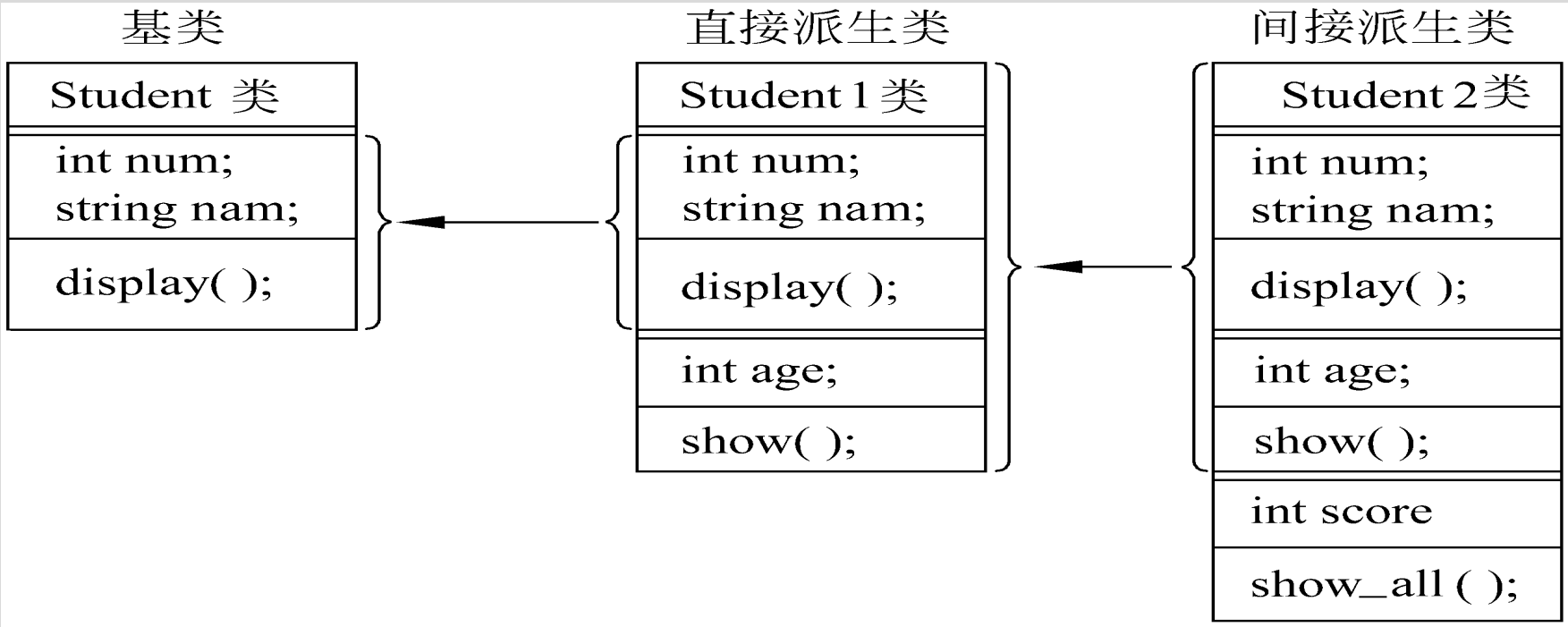
- 问题

  （1）在类外如何定义构造函数？

  （2）在派生类的构造函数列表中，基类构造函数参数为什么没有类型？

◆多层派生时的构造函数

- 基类可以派生出一个派生类，派生类还可以继续派生，形成派生的层次结构
- 多级派生情况下派生类的构造函数



```
#include <iostream>
#include <string>
using namespace std;
class Student//声明基类
 {public:                                  //公用部分
  Student(int n, string nam )              //基类构造函数
   {num=n;
    name=nam;}
  void display( )                          //输出基类数据成
{cout<<"num:"<<num<<endl;
   cout<<"name:"<<name<<endl;}
 protected:                                //保护部分
  int num;                                 //基类有两个数据成员
  string name;};
class Student1: public Student             //声明公用派生类Student1
 {public:
  Student1(int n,string nam,int a):Student(n,nam)//派生类构造函数
   {age=a; }                               //在此处只对派生类新增的数据成员初始化
  void show( )                             //输出num， name和age
   {display( );                            //输出num和name
    cout<<"age:"<<age<<endl;}
  private:                                 //派生类的私有数据
   int age;  };                            //增加一个数据成员
class Student2:public Student1             //声明间接公用派生类Student2
 {public://下面是间接派生类构造函数
  Student2(int n, string nam,int a,int s):Student1(n,nam,a)
{score=s;}
  void show_all( )                         //输出全部数据成员
{show( );                                  //输出num和name
   cout<<"score:"<<score<<endl; }          //输出age
private:
  int score; };                            //增加一个数据成员
 int main( )
 {Student2 stud(10010,"Li",17,89);
  stud.show_all( );                        //输出学生的全部数据
return 0;}
```

```
C:\WINDOWS\system32\cmd.exe
num:10010
name:Li
age:17
score:89
请按任意键继续. . .
```

◆基类和两个派生类的构造函数的写法:

- 基类的构造函数首部: Student(int n, string nam)
- 派生类Student1的构造函数首部:

  Student1(int n, string nam,int a):Student(n,nam)
- 派生类Student2的构造函数首部:

  Student2(int n, string nam,int a,int s): Student1(n,nam,a)
- 定义Student2类对象时执行其构造函数，顺序为:

①调用Student构造函数先初始化基类数据成员num和name

②调用Student1构造函数再初始化Student1的数据成员age

③最后初始化Student2的数据成员score

多层派生构造函数一般形式:

构造函数（参数总表）：直接基类构造函数.

◆派生类构造函数的特殊情况

（1）当不需要对派生类新增的成员进行任何初始化操作时，派生类构造函数的函数体可以为空，即构造函数是空函数

```
class Employee {
public: //基类只定义了有参构造函数
    Employee(char * nam, Date& bday, char se, short dep);
};
class Manager : public Employee {
public: //派生类也必须定义构造函数
    Manager(char * nam, Date& bday, char se, short dep);
};
Manager::Manager(char * nam,Date& bday,char se,short dep)
    : Employee(nam, bday, se, dep) //显式调用基类有参构造函数
{ }
```

- 派生类构造函数可以不写基类构造函数调用，前提是：基类没有定义任何构造函数，或者只是定义了无参构造函数，此时C++编译器将自动调用基类的无参构造函数（若无，则调用默认构造函数），无需传递参数

```
class Employee {
public:
    Employee();
};
class Manager : public Employee {
public:
    Manager();
};
Manager::Manager()
//不写基类构造函数调用,将自动调用Employee()
```

```
#include<string>
#include <iostream>
using namespace std;
class Teacher
{public:
 Teacher( )
 {name="huang";
  age=28;
  sex='m'; }
 void display();
 protected:
  string name;
  int age;
  char sex;};

void Teacher::display()
   {cout<<"name:"<<name<<endl;
    cout<<"age"<<age<<endl;
    cout<<"sex:"<<sex<<endl; }
class Teacher_Cadre:public Teacher
{public:
  Teacher_Cadre(float w);
  void show( );
 private:
  float wage;};

Teacher_Cadre::Teacher_Cadre(float w):wage(w){}
void Teacher_Cadre::show( )
  {Teacher::display();
   cout<<"wages:"<<wage<<endl;}

int main( )
{Teacher_Cadre te_ca(45);
 te_ca.show( );
 return 0;}
```

```
C:\WINDOWS\system32\cmd.exe
name:huang
age28
sex:m
wages:45
请按任意键继续. . .
```

空函数

派生类的析构函数

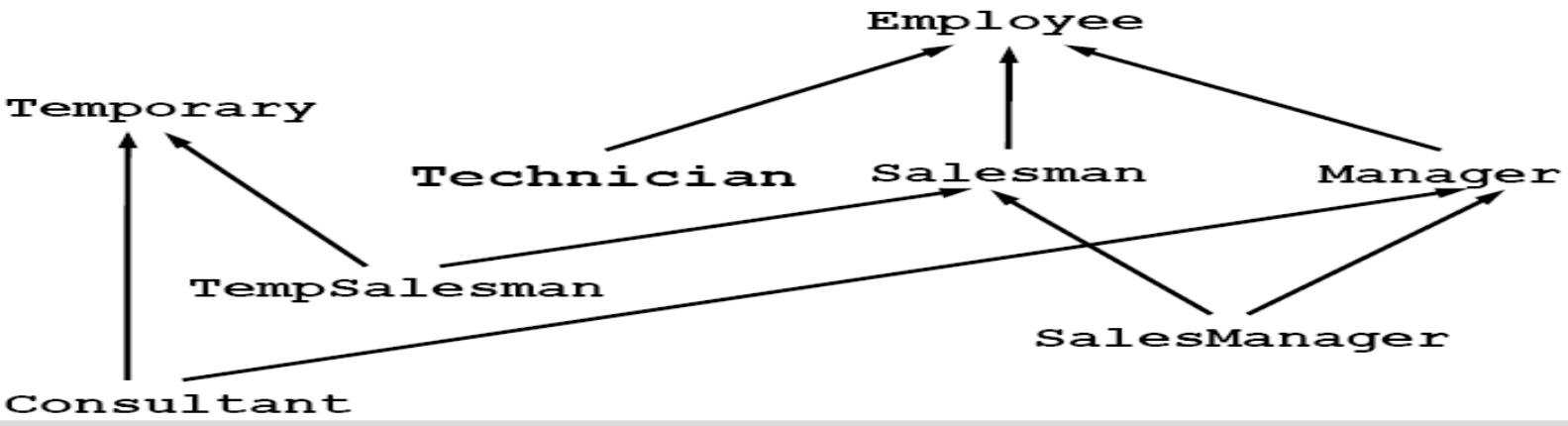
- 派生类是不能继承基类的析构函数，也需要通过派生类的析构函数去调用基类的析构函数；
- 在派生类中可以根据需要定义自己的析构函数，用来对派生类中所增加的成员进行清理工作；
- 基类的清理工作仍然由基类的析构函数负责；
- 在执行派生类的析构函数时，调用的顺序与构造函数正好相反：先执行派生类自己的析构函数，对派生类新增加的成员进行清理，然后调用基类的析构函数，对基类进行清理.

7.2 多重继承*

1. 声明多重继承的方法

- 多重继承(multiple inheritance)：：一个派生类有两个或多个基类，派生类从两个或多个基类中继承所需的属性。
- 一般格式：如果已声明了类A、类B和类C，可以声明多重继承的派生类D:

```
class D:public A, private B,protected C
{类D新增加的成员}
```

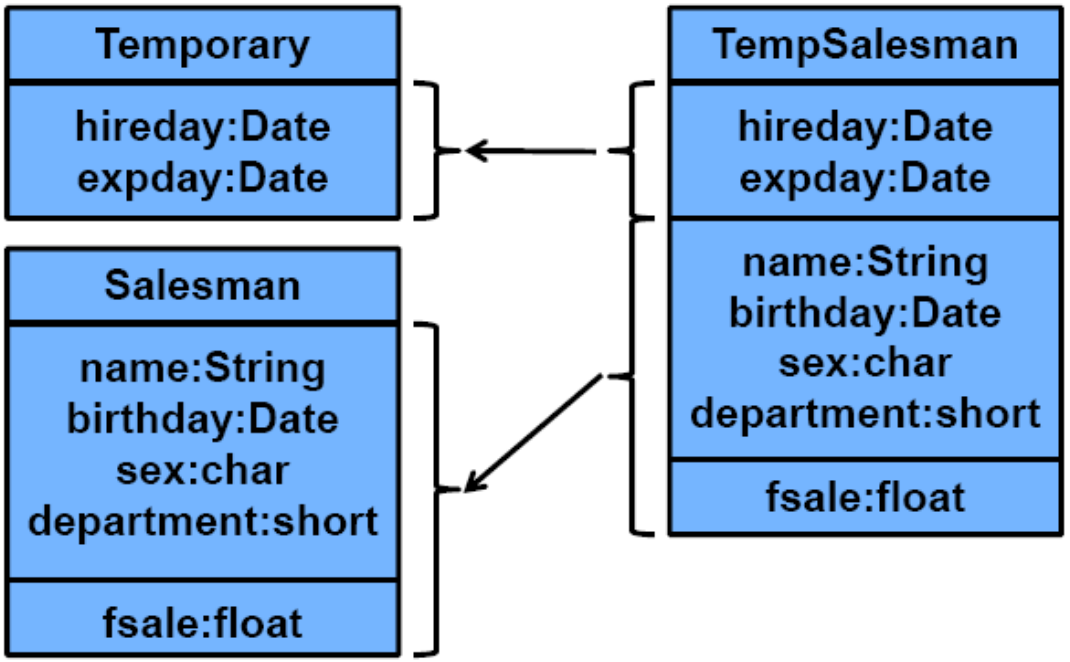


2. 多重继承派生类的构造函数

- 多重继承派生类的构造函数形式：

```
派生类构造函数名(总参数表列)
:  基类1构造函数(参数表列),
   基类2构造函数(参数表列),
   基类3构造函数(参数表列), ...
{  派生类构造函数体  }
```

- 多重继承派生类构造函数的执行顺序：先调用各基类的构造函数，再执行派生类构造函数。调用各基类构造函数的顺序是按照声明派生类时基类出现的顺序




```
1.  class TempSalesman: public Temporary, public Salesman {
2.  public:
3.     TempSalesman(char* nam, Date& bday, char se, short dep,
4.                  Date& hday, Date& eday);
5.  };

6.  TempSalesman::TempSalesman(char* nam, Date& bday, char se,
7.     short dep, Date& hday, Date& eday)
8.  : Temporary(hday, eday), Salesman(nam, bday, se, dep)
9.  {
10. }
```

```
#include <iostream>
#include <string>
using namespace std;
class Teacher//声明类Teacher(教师)
 {public:              //公用部分
  Teacher(string nam,int a, string t)      //构造函数
   {name=nam;  age=a;  title=t;}
  void display( )                          //输出教师有关数据
   {cout<<"name:"<<name<<endl; cout<<"age"<<age<<endl;
    cout<<"title:"<<title<<endl;}
 protected:                                //保护部分
  string name;   int age;    string title;};
class Student                              //定义类Student(学生)
 {public:
  Student(string nam,char s,float sco)
   {name1=nam; sex=s; score=sco;}    //构造函数
  void display1( )           //输出学生有关数据
   {cout<<"name:"<<name1<<endl;
    cout<<"sex:"<<sex<<endl;
    cout<<"score:"<<score<<endl;}
 protected:                                //保护部分
  string name1;
  char sex;  float score;};
class Graduate:public Teacher,public Student   //声明多重继承的派生类Graduate
 {public:
   Graduate(string nam,int a,char s, string t,float sco,float w):
  Teacher(nam,a,t),Student(nam,s,sco),wage(w) { }
  void show( )        //输出研究生的有关数据
  {cout<<"name:"<<name<<endl;  cout<<"age:"<<age<<endl;
   cout<<"sex:"<<sex<<endl;     cout<<"score:"<<score<<endl;
   cout<<"title:"<<title<<endl;    cout<<"wages:"<<wage<<endl;}
 private:
  float wage; };
int main( )
 {Graduate grad1("Wang-li",24,'f',"assistant",89.5,1234.5);
  grad1.show( );
  return 0;}
```

```
C:\WINDOWS\system32\cmd.exe
name:Wang-li
age:24
sex:f
score:89.5
title:assistant
wages:1234.5
请按任意键继续. . .
```

```
49      }
50    private:
51      float wage;
52  };
53
54  int main( )
55  {Graduate grad1("W
56   grad1.show( );
57   return 0;
58  }
59
60
```

类视图

<搜索>

- huang
  - 宏和常量
  - 全局函数和变量
  - Graduate
    - 基类型
      - Student
      - Teacher
  - Student
    - 派生类型
      - Graduate
  - Teacher
    - 派生类型
      - Graduate

解决方案资源管理器 | 类视图

自动窗口

名称	值	类型
grad1	{wage=1234.5000 }	Graduat
Teacher	{name="Wang-li" age=24 title="assistant" }	Teacher
name	"Wang-li"	std::bas
age	24	int
title	"assistant"	std::bas
Student	{name1="Wang-li" sex='f' score=89.500000 }	Student
name1	"Wang-li"	std::bas
sex	102 'f'	char
score	89.500000	float
wage	1234.5000	float

自动窗口 | 局部变量 | 线程 | 模块 | 监视 1

```
#include <iostream>
#include <string>
using namespace std;
class Teacher  //声明类Teacher(教师)
{public :
Teacher(string nam,int a, string t)//构造函数
{name=nam,age=a,title=t;}
void display( )  //输出教师有关数据
{cout<<"name:"<<name<<endl;
 cout<<"age"<<age<<endl;
 cout<<"title:"<<title<<endl; }
protected :
 string name; int age;
 string title;};
class Student  //定义类Student(学生)
{public :
Student(string nam,char s,float sco)
{name=nam;
sex=s;
score=sco;}       //构造函数
void display1( )    //输出学生有关数据
{cout<<"name:"<<name<<endl;
cout<<"sex:"<<sex<<endl;
cout<<"score:"<<score<<endl; }
protected :
 string name;
 char sex;
 float score;};
class Graduate_Student: public Teacher, public Student //声明多重继承派生类
{public:
Graduate_Student(string  nam, string nam1,int a, char s, string t, float sco, float w):Teacher(nam,a,t),Student(nam1, s, sco),wage(w) {
void show( ) //输出研究生的有关数据
{cout<<"name:"<<name<<endl;
 cout<<"age:"<<age<<endl;cout<<"sex:"<<sex<<endl;
 cout<<"score:"<<score<<endl;cout<<"title:"<<title<<endl;
 cout<<"wages:"<<wage<<endl;}
private: float wage;  };
int main( )
{ Graduate_Student grad1("Wang-san","huang",25,'f',"assistant",90.5,1233.5);
  grad1.show( );
  return 0;}
```

输出

显示输出来源(S): 生成

```
删除项目“huangtest”(配置“Debug|Win32”)的中间文件和输出文件
编译...
test.cpp
学文档\程序设计基础\tsinghua.c\2010xia\练习\huangtest\huangtest\huangtest.cpp(33) : error C2385: 对“name”的
    可能是“name”(位于基“Teacher”中)
    也可能是“name”(位于基“Student”中)
日志保存在“file://f:\教学文档\程序设计基础\tsinghua.c\2010xia\练习\huangtest\huangtest\Debug\BuildLog.htm”
test - 1 个错误，0 个警告
=== 全部重新生成: 成功 0 个，失败 1 个，跳过 0 个 ==========
```

代码定义窗口 | 调用浏览器 | 输出

```
16  {public :
17  Student(string nam,char s,float sco)
18  {name=nam;
19  sex=s;
20  score=sco;}      //构造函数
21  void display1( )    //输出学生有关数据
22  {cout<<"name:"<<name<<endl;
23  cout<<"sex:"<<sex<<endl;
24  cout<<"score:"<<score<<endl; }
25  protected :
26   string name;
27   char sex;
28   float score;};
29  class Graduate_Student: public Teacher, public Student //声明多重继承派生类
30  {public:
31  Graduate_Student(string  nam, string nam1,int a, char s, string t, float sco, float w):Teacher(nam,a,t),Student(nam1, s, sco),w
32  void show( ) //输出研究生的有关数据
33  {cout<<"name:"<<Teacher::name<<endl;
34   cout<<"age:"<<age<<endl;cout<<"sex:"<<sex<<endl;
35   cout<<"score:"<<score<<endl;cout<<"title:"<<title<<endl;
36   cout<<"wages:"<<wage<<endl;}
37  private: float wage;  };
38  int main( )
39  { Graduate_Student grad1("Wang-san","huang",25,'f',"assistant",90.5,1233.5);
40   grad1.show( );
41   return 0;}
42
```

局部变量

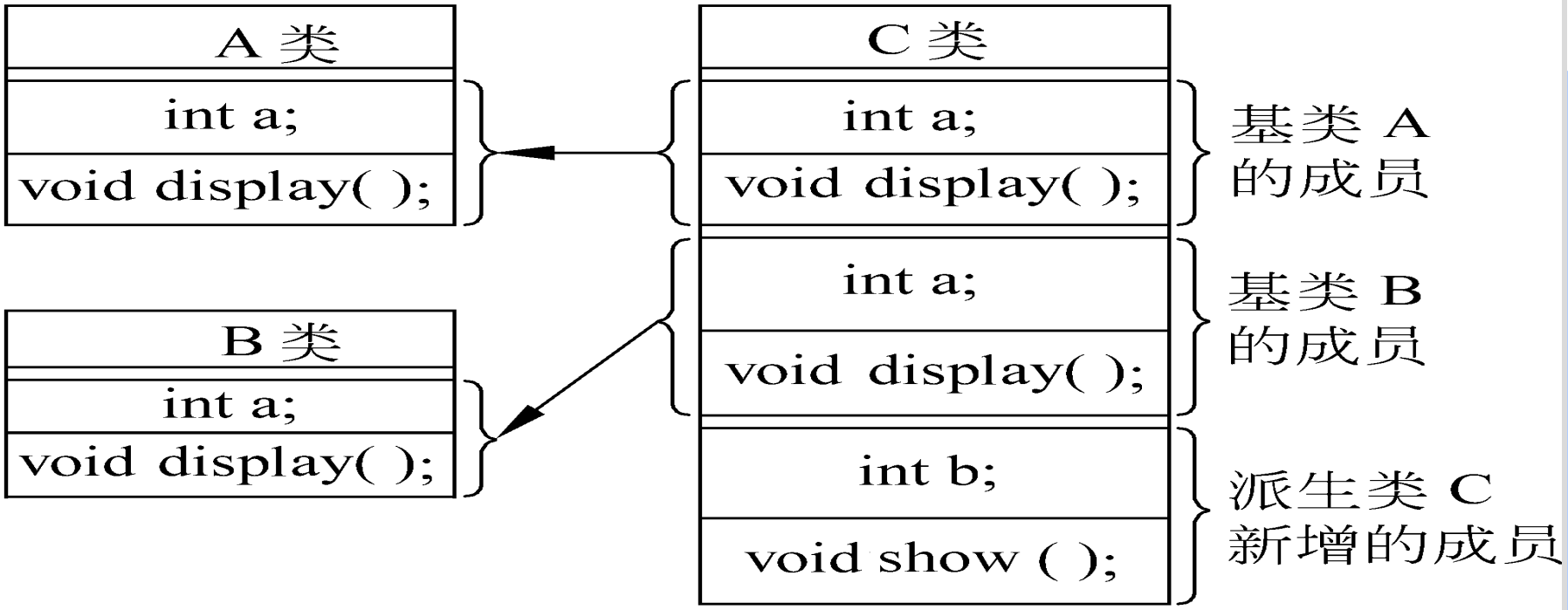
名称	值	类型
grad1	{wage=1233.5000 }	Graduate_Student
Teacher	{name="Wang-san" age=25 title="assistant" }	Teacher
name	"Wang-san"	std::basic_string<
age	25	int
title	"assistant"	std::basic_string<
Student	{name="huang" sex='f' score=90.500000 }	Student
name	"huang"	std::basic_string<
sex	102 'f'	char
score	90.500000	float
wage	1233.5000	float

自动窗口 | 局部变量 | 线程 | 模块 | 监视 1

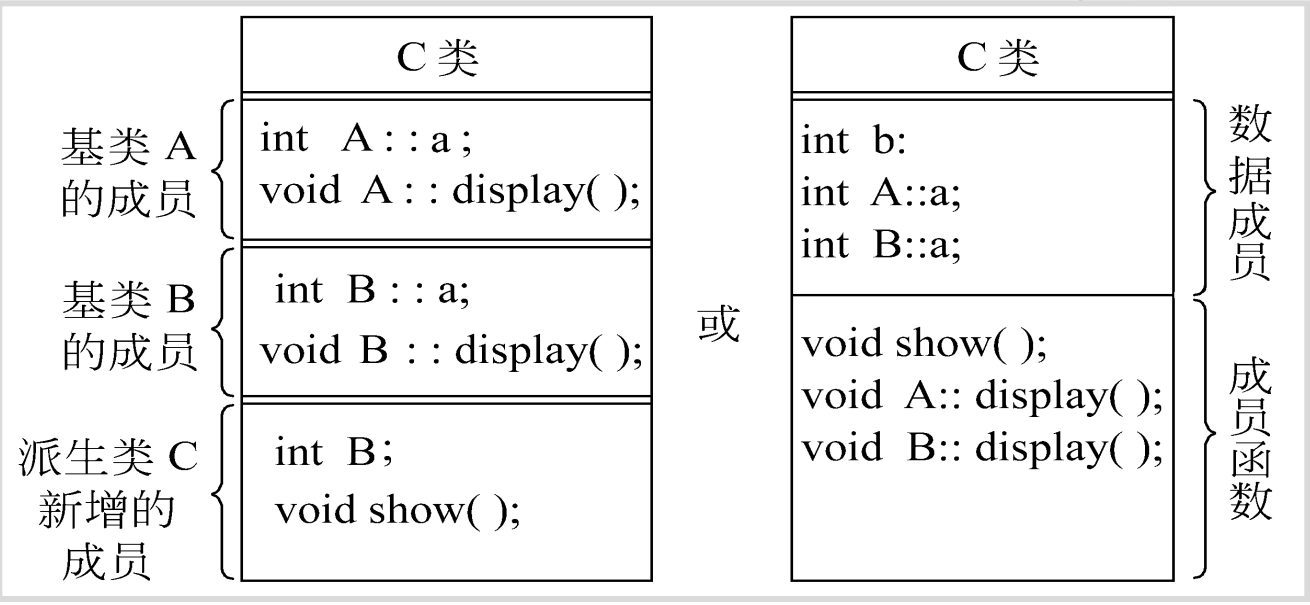
```
f:\教学文档\程序设计基...
name:Wang-san
age:25
sex:f
score:90.5
title:assistant
wages:1233.5
```

- 多重继承引起的二义性问题
  - 多重继承最常见问题是继承的成员同名而产生的二义性(ambiguous)
  - 如果类A和类B中都有成员函数display和数据成员a，类C是类A和类B的直接派生类。分3种情况讨论：

（1）两个基类有同名成员



- 解决方法：用基类名来限定。
  - 类外访问:对象.类名::成员。例，c1.A::a=3;
c1.A::display();
  - 类内访问:类名::成员。例，A::a=3;　A::display( );



- **思考：假设构建C类对象Cobj，则Cobj中的同名成员的存储空间如何分配？会重名吗？为什么？**

(2) 两个基类和派生类三者都有同名成员。

```
class C :public A,public B
{int a;
  void display();};
```

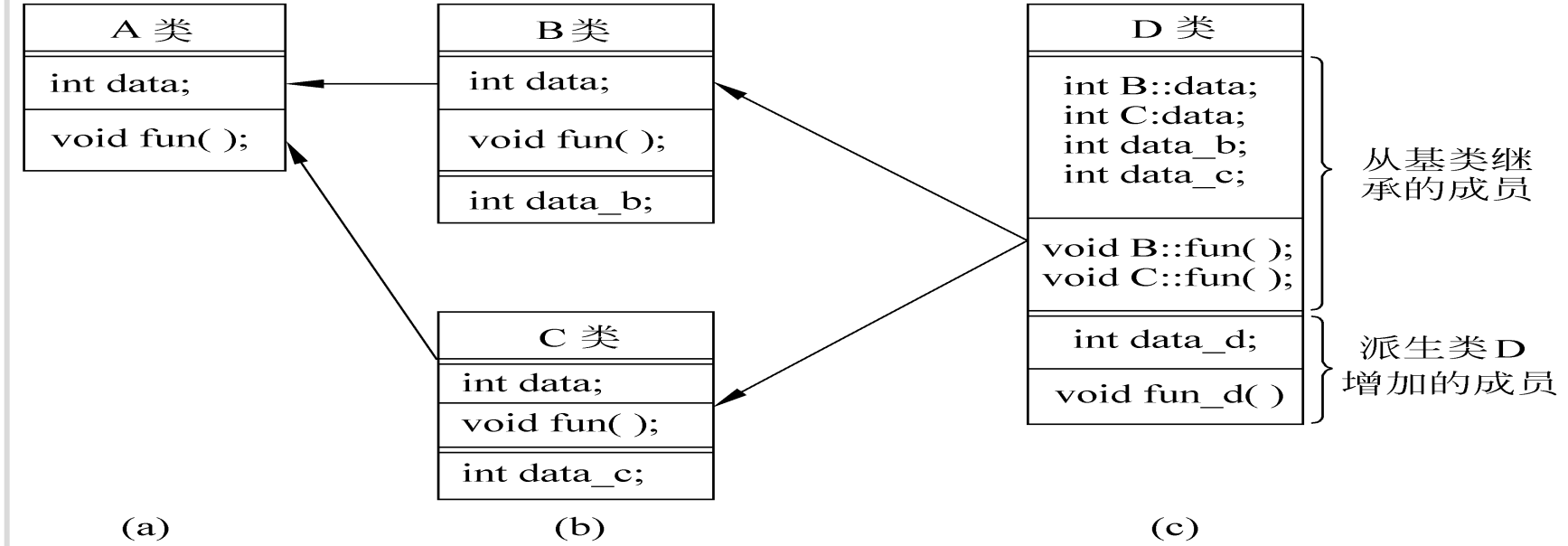
C 类
int a; int A: : a; int B: : a;

void display( );
void A::display( );
void B::display( );

- 如果在main函数中定义C类对象c1，并调用数据成员a和成员函数display。如：C c1; c1.a=3; c1.display( ); **调用的是哪个?**

- 同名覆盖规则:如果在定义派生类对象的模块中通过对象名访问同名成员，则访问的是派生类的成员，即派生类优先原则;
- 注意: 成员函数只有在函数名和参数都相同才发生同名覆盖。只是函数名相同而参数不同为函数重载.

- 虚基类* *
- 问题引入：如果一个派生类有多个直接基类，而这些直接基类又有一个共同的基类，则在最终的派生类中会保留该间接共同基类数据成员的多份同名成员，如何处理？
- 方法1:在引用这些同名的成员时，必须在派生类对象名后增加直接基类名，使其唯一标识一个成员；
- 方法2:采用virtual base class方法，使在继承间接相同基类时只保留一份成员，即同名成员在内存中只有1份拷贝。



- 声明虚基类的一般形式为：
  class 派生类名: virtual 继承方式 基类名
- 例：将类A声明为虚基类的方法如下：

```
class A//声明基类A
{...};
class B :virtual public A        //声明类B是类A的公用派生类，A是B的虚基类
{...};
class C :virtual public A        //声明类C是类A的公用派生类，A是C的虚基类
{...};
class D:public B,public C
{....}
```

- 注意：虚基类并不是在声明基类时声明，而是在声明派生类时指定继承方式时声明。作用范围同继承方式；
- 一个基类可以在生成一个派生类时作为虚基类，而在生成另一个派生类时不作为虚基类；
- 经过这样声明后，当基类通过多条派生路径被一个派生类继承时只继承该基类一次。

```
1.  class Salesman: virtual public Employee
2.  { ... };

3.  class Manager: virtual public Employee
4.  { ... };

5.  class SalesManager: public Salesman, public Manager
6.  {  public:
7.     Print() {
8.        cout << "name: " << name << endl;
9.        cout << "birth: " << birthday << endl;
10.       cout << "sex: " << sex << endl;
11.       cout << "department: " << department << endl;
12.       //打印Salesman类成员
13.       cout << "sales income: " << fsale << endl;
14.       //打印Manager类成员
15.       cout << "level: " << level << endl;
16. } .
```

```
1  #include <iostream>
2  using namespace std;
3  class A
4  {public:
5    A(void)
6    {a=10;}
7  void Func(void)
8  {cout<<"Func of A"<<endl;}
9  protected :
10 int a;
11 };
12
13 class B : virtual public A
14 {public:
15   B( void )
16    {a += 10;
17     cout <<"Ba = "<< a <<"\n";}
18 };
19 class C: virtual public A
20 {public:
21   C( void )
22    {a += 10;
23     cout <<"Ca = "<< a <<"\n";}
24 };
25 class D:B,C
26 {public:
27     D( void )
28     {cout <<"a = "<< a << endl;
29     }
30 };
31  void main()
32 { D objD;
33 }
34
```

局部变量

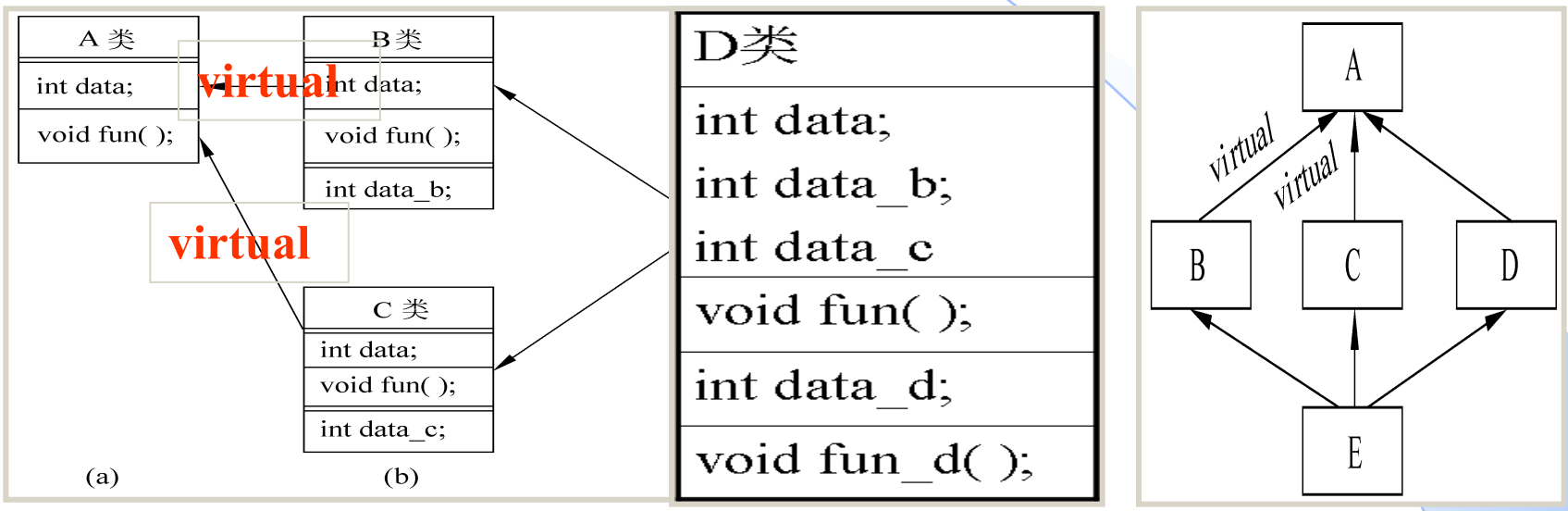
名称	值	类型
objD	{...}	D
B	{...}	B
A	{a=30 }	A
a	30	int
C	{...}	C
A	{a=30 }	A
a	30	int

自动窗口 | 局部变量 | 线程 | 模块 | 监视 1

```
f:\教学文档\程序设计基础\tsinghua.c\2010xia\练习\huangtest\De
Ba = 20
Ca = 30
a = 30
```

虚基类的成员在多重派生类中在内存只有一份拷贝

- 派生类B和C中作虚基类声明，派生类D的成员下图所示



- 注意：为了保证虚基类在派生类中只继承一次，应当在该基类的所有直接派生类中声明为虚基类。否则仍然会出现对基类的多次继承，如右图

◆虚基类的初始化

- 如果在虚基类中定义了带参数的构造函数，则在其所有派生类（包括直接派生或间接派生的派生类）中，通过构造函数的初始化表对虚基类进行初始化：

```
class A {
    A(int i) {}   //基类A的构造函数,有一个参数
    ... };
class B: virtual public A { //A为B的虚基类
    B(int i): A(i) {} //B类构造函数,在初始化表中对虚基类A初始化
    ... };
class C: virtual public A { //A为C的虚基类
    C(int i): A(i) {} //C类构造函数,在初始化表中对虚基类A初始化
    ... };
class D: public B, public C
    D(int i): A(i), B(i), C(i) {} //D类构造函数,在初始化表中
    ...                           //对所有基类作初始化
```

须由派生类D直接调用间接基类A的有参构造函数，因为B、C调用A的构造函数时可能会出现参数矛盾

按语法，B和C类构造函数调用不能省略，但C++编译器会忽略B和C类中对虚基类A的构造函数的调用

```
#include <iostream>
#include <string>
using namespace std;
class Person  //声明公共基类Person
   {public:
   Person(string nam,char s,int a) //构造函数
   {name=nam;sex=s;age=a;}
   protected:                          //保护成员
   string name;   char sex;   int age;};
class Teacher:virtual public Person     //声明Person的直接派生类Teacher
   {public:
   Teacher(string nam,char s,int a, string t):Person(nam,s,a)//构造函数
    {title=t; }
  protected:  //保护成员
    string title; }; //职称
class Student:virtual public Person            //声明Person为公用继承的虚基类
     {public:
     Student(string nam,char s,int a,float sco)     //构造函数
      :Person(nam,s,a),score(sco){ }              //初始化表
    protected:                                   //保护成员
    float score;  };                             //成绩
class Graduate:public Teacher,public Student   //Teacher和Student为直接基类
    {public:
    Graduate(string nam,char s,int a, string t,float sco,float w)//构造函数
    :Person(nam,s,a),Teacher(nam,s,a,t),Student(nam,s,a,sco),wage(w){}
    void show( )       //输出研究生的有关数据
    {cout<<"name:"<<name<<endl;  cout<<"age:"<<age<<endl;
     cout<<"sex:"<<sex<<endl;        cout<<"score:"<<score<<endl;
     cout<<"title:"<<title<<endl;        cout<<"wages:"<<wage<<endl; }
    private:
    float wage; };                  //工资
int main( )
  {Graduate grad1("Wang-li",'f',24,"assistant",89.5,1234.5);
  grad1.show( );
  return 0;}
```

```
C:\WINDOWS\system32\cmd.exe
name:Wang-li
age:24
sex:f
score:89.5
title:assistant
wages:1234.5
请按任意键继续. . .
```

虚基类的构造函数执行顺序

（1）虚基类的构造函数在非虚基类之前调用

（2）同一层包含多个虚基类，则按申明顺序调用

（3）若虚基类由非虚基类派生而来，则要先调用更高级别的基类构造函数，再遵循（1）和（2）的顺序

例如：

```
Class A;
Class B;
Class C: public A,virtual B
{
 };
```

调用顺序：

```
B()
A()
C()
```

```
#include <iostream>
using namespace std;
class Base1
{public:
   Base1( void )
  {cout <<"class Base1"<<endl;}
};
class Base2
{public:
  Base2( void )
  {cout <<"class Base2"<<endl;}
};
class Level1 : public Base2, virtual public Base1
{public:
 Level1 ( void )
{cout <<"class Level1"<<endl;}
};
class Level2 : public Base2, virtual public Base1
{public:
 Level2 ( void )
{cout <<"class Level2"<<endl;}
};
class Leaf : public Level1, virtual public Level2
{public:
   Leaf ( void )
  {cout << "class Leaf" <<endl;}
};

void main(void)
{
Leaf obj;}
```

◆混合继承：多基类继承与多重继承

```
#include  < IOSTREAM.H >
 // 基类
 class  CBase
{
protected:
    int a;
public:
    CBase(int na)
    {
        a=na;
        cout<<"CBase constructor! ";
    }

    ~CBase(){cout<<"CBase deconstructor! ";}
};
```

◆混合继承：多基类继承与多重继承

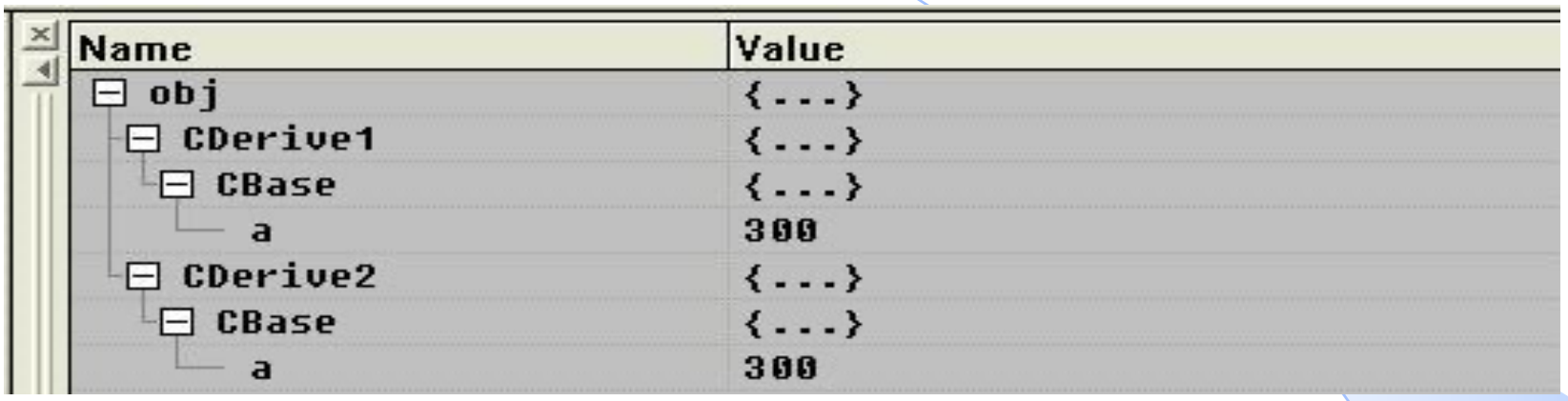
```
// 派生类1(声明CBase为虚基类)
class CDerive1: virtual public CBase
{
public:
    CDerive1(int na):CBase(na)
    {
        cout<<"CDerive1 constructor! ";
    }

    ~CDerive1(){cout<<"CDerive1 deconstructor! ";}

    int GetA(){return a;}
};
```

```
// 派生类2(声明CBase为虚基类)
class CDerive2: virtual public CBase
{
public:
    CDerive2(int na):CBase(na)
    {
        cout<<"CDerive2 constructor! ";
    }
    ~CDerive2(){cout<<"CDerive2 deconstructor! ";}
    int GetA(){return a;}
};
```

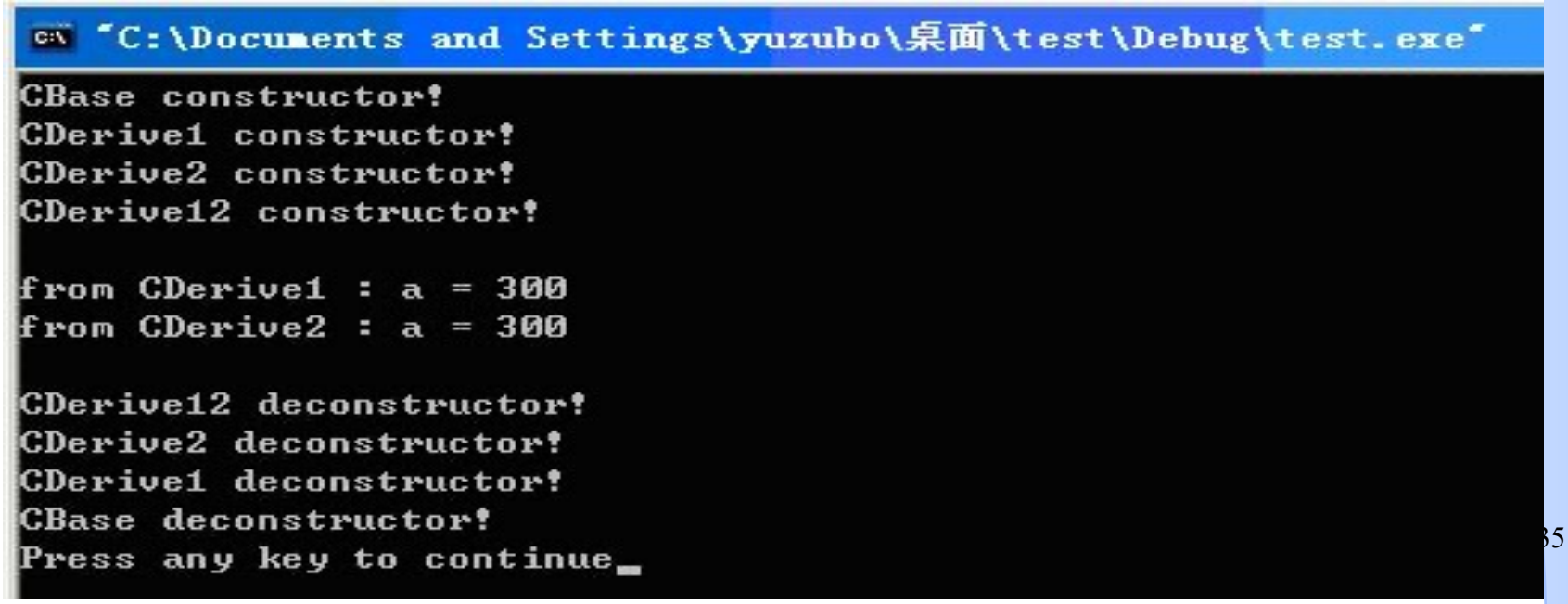
◆混合继承：多基类继承与多重继承

```
// 子派生类
class CDerive12: public CDerive1, public CDerive2
{
public:
    CDerive12(int na1,int na2,int na3):CDerive1(na1),CDerive2(na2),CBase(na3)
    {
        cout<<"CDerive12 constructor! ";
    }
    ~CDerive12(){cout<<"CDerive12 deconstructor! ";}
};
void main()
{
    CDerive12 obj(100,200,300);
    //得到从CDerive1继承的值
    cout<<" from CDerive1 : a = "<<obj.CDerive1::GetA();
    //得到从CDerive2继承的值
    cout<<" from CDerive2 : a = "<<obj.CDerive2::GetA()<<endl<<endl;
}
```

◆混合继承：多基类继承与多重继承



Name	Value
obj	{...}
CDerive1	{...}
CBase	{...}
a	300
CDerive2	{...}
CBase	{...}
a	300



```
"C:\Documents and Settings\yuzubo\桌面\test\Debug\test.exe"
CBase constructor!
CDerive1 constructor!
CDerive2 constructor!
CDerive12 constructor!

from CDerive1 : a = 300
from CDerive2 : a = 300

CDerive12 deconstructor!
CDerive2 deconstructor!
CDerive1 deconstructor!
CBase deconstructor!
Press any key to continue
```

7.3 继承与组合

- 子对象：类申明时包含另一个类对象作为数据成员；
- 类的组合(composition)：包含子对象的类申明。类似结构体的嵌套申明；
- 类的组合和继承一样，都是有效地利用已有类的资源。但二者有着本质区别：继承是纵向(同一系族），组合是横向（不同系族）

```
class Teacher//教师类
{public:
  ┆
private:
   int num;
   string name;
   char sex; };
```

```
class BirthDate          //生日类
{ public:
  ┆
private:
int year;
int month;
int day;};
```

```
class Professor:public Teacher      //教授类
{public:
  ┆
   private:
BirthDate birthday; };          //BirthDate类的对象作为数据成员
```

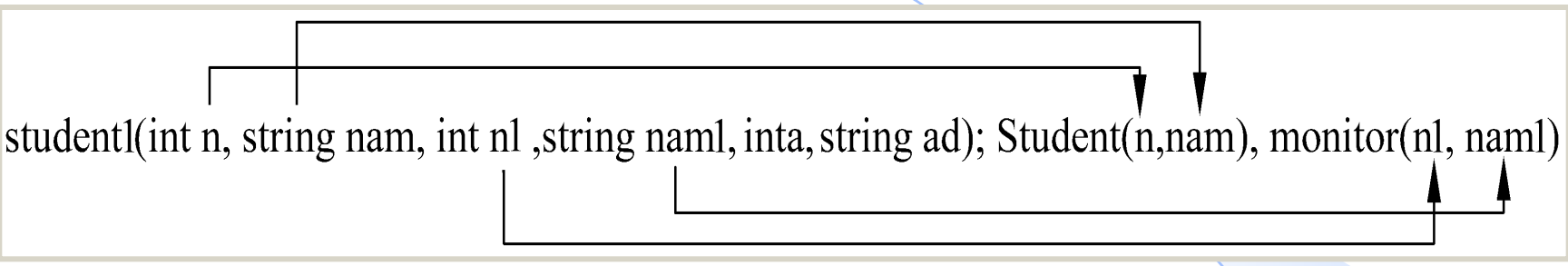
- 继承情况下：派生类包含基类成员，派生类构造函数执行前会自动调用基类构造函数，对基类数据成员进行初始化
- 组合情况下：组合类包含内嵌对象成员，组合类构造函数执行前会自动调用内嵌对象的构造函数，对内嵌对象数据成员进行初始化
- 当类同时存在继承和组合时，构造函数将按照以下顺序调用：
  - 基类构造函数调用，对基类数据成员初始化
  - 内嵌对象构造函数调用，对内嵌对象成员初始化
  - 派生类构造函数体执行，对派生类数据成员初始化

```
#include <iostream>
#include <string>
using namespace std;
class Student//声明基类
 {public:                              //公用部分
Student(int n, string nam )    //基类构造函数
    {num=n;
     name=nam;}
   void display( )   //成员函数，输出基类数据成员
    {cout<<"num:"<<num<<endl<<"name:"<<name<<endl;}
protected:                            //保护部分
int num;
    string name;};

class Student1: public Student          //声明公用派生类Student1
{public:
   Student1(int n, string nam,int n1, string nam1,int a, string ad)
     :Student(n,nam),monitor(n1,nam1)      //派生类构造函数
    {age=a;
     addr=ad;}
   void show( )
    {cout<<"This student is:"<<endl;
     display();                          //输出num和name
     cout<<"age:"<<age<<endl;              //输出age
     cout<<"address:"<<addr<<endl<<endl; }   //输出addr
     void show_monitor( )                  //成员函数，输出子对象
    {cout<<endl<<"Class monitor is:"<<endl;
     monitor.display( ); }                 //调用基类成员函数
    private:                             //派生类的私有数据
    Student monitor;                     //定义子对象
    int age;
    string addr;};
int main( )
{Student1 stud1(10010,"Wang-li",10001,"Li-sun",19,"115 Beijing Road,Shanghai");
   stud1.show( );                        //输出学生的数据
   stud1.show_monitor();                   //输出子对象的数据
return 0;}
```

```
This student is:
num:10010
name:Wang-li
age:19
address:115 Beijing Road,Shanghai


Class monitor is:
num:10001
name:Li-sun
请按任意键继续. . .
```

monitor(n1,nam1)是student(n,nam)实例化

- 注意构造函数中6个形参顺序



- 包含子对象派生类构造函数的一般形式为

  派生类构造函数名（总参数表列）：基类构造函数名（参数表列），子对象名(参数表列)

  {派生类中新增数成员据成员初始化语句}

问题：构造函数中基类与子对象构造顺序可以对换？为什么？

本讲重点分析

- 派生类的构造函数：基类构造函数、多级派生类的构造函数，虚基类的构造函数
- 多重派生时唯一标识问题：同名屏蔽、作用域分辨符、虚基类
- 类的组合、子对象概念

第7次作业

- 要求与前面同，在第9周末交
- 第1题：第1题:教材第16章后面习题中第9题（P509）
- 第2题：在第6次作业的基础上，补充各类（People、student、graduate、teacher和TA）的构造函数和析构函数。

（1）编程测试和分析这些构造函数和析构函数的执行顺序；

（2）要求采用虚基类，重新从teacher类和graduate类派生出TA(助教)类，并对比分析第6次作业没有使用虚基类情况，体会有什么差别？

选做题

第1题： 阅读教材第3章（继承与派生）中综合程序代码，分析其中使用了“多次继承和虚基类方法”来生成哪些新的类？

第2题： 完成下列程序填空

将程序中缺少的部分填上，让程序能正常运行并得到正确结果

```
# include   <iostream>
using namespace std;
class TT
{     public:
                ____________________________________________// 填空 1
                void Print( );
              ____________________________________________ // 填空 2
        private:
              int N, D;
};
TT Div(TT &r1, TT &r2)
{     return TS(r1.N / r2.D + r1.D / r2.N, r1.D / r2.D);
}
TT:: TT (int n, int d)
{     N=n; D=d;          }
void TT::Print( )
{     cout<<"N=" <<N<<" D="<<D<<endl;          }
void main( )
{   TT a(1,2), b(3,4), c;
        c = Div(a, b);
       c.Print( );
}
```