C++语言程序设计
第五章 类的继承

王焦乐

http://faculty.hitsz.edu.cn/jlwang

哈尔滨工业大学（深圳）

机电工程与自动化学院

本章主要内容

- 继承与派生
- 派生类的声明范式
- 派生类的构成
- 派生类成员的访问属性
- 派生类的构造函数和析构函数
- 多重继承
- 基类与派生类的转换
- 继承与组合
- 继承在软件开发中的重要意义

0、继承与派生

- 抽象
- 封装
- 继承（inheritance）
  - 面向对象程序设计的最重要的特征
  - 软件的可重用性（software reusability）
- 多态

0、继承与派生

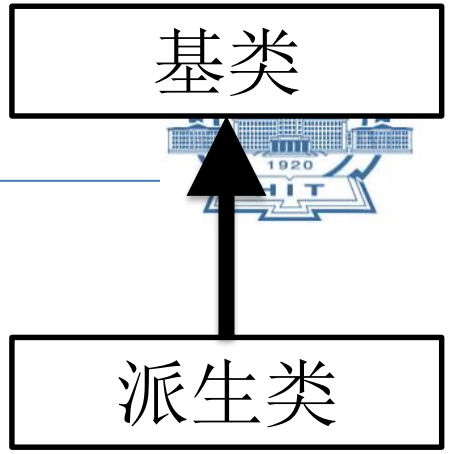
□ 利用继承机制，减少重复的工作量

```
class Student
{
private:
  int num;
  string name;
  char sex;

public:
  void display()
  {
    cout << "num: " << num << endl;
    cout << "name: " << name << endl;
    cout << "sex: " << sex << endl;
  }
};
```

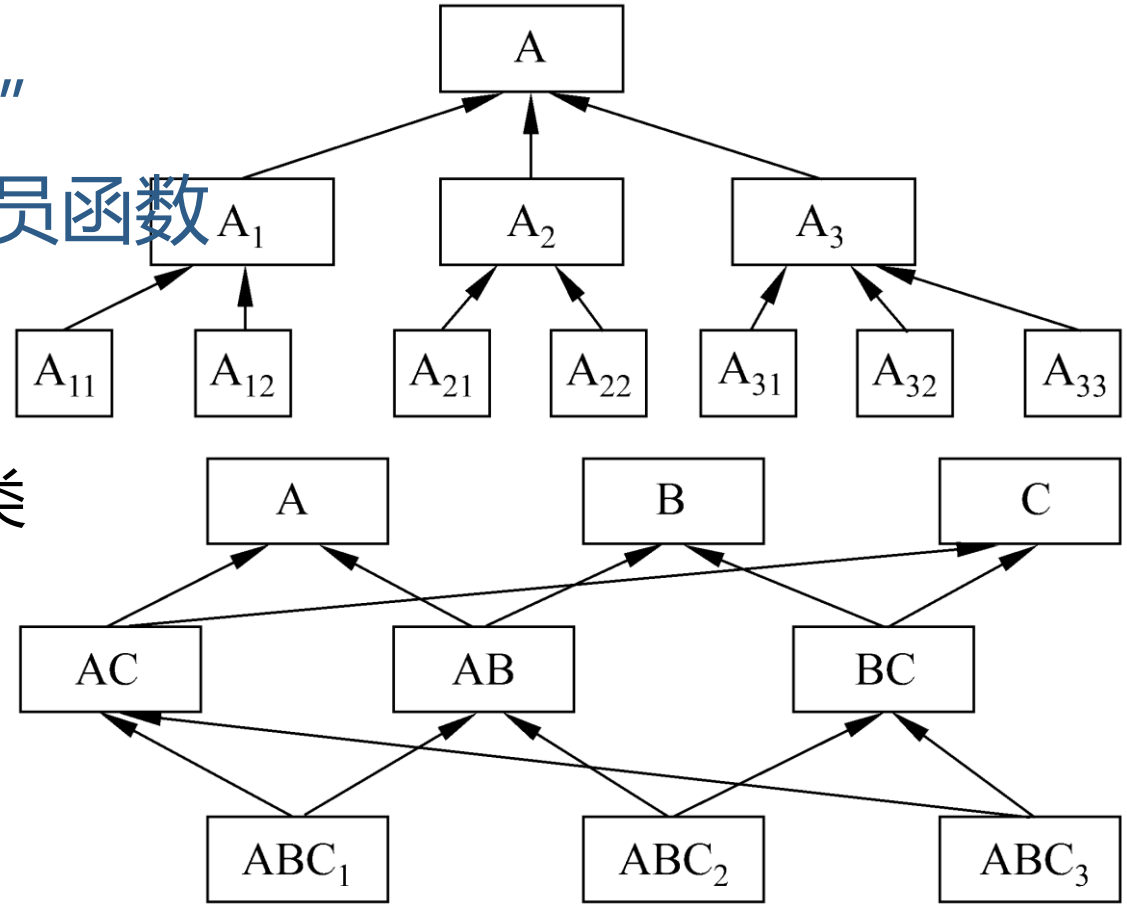
```
class Student1
{
private:
  int num;
  string name;
  char sex;
  int age;
  char addr[20];
public:
  void display()
  {
    cout << "num: " << num << endl;
    cout << "name: " << name << endl;
    cout << "sex: " << sex << endl;
    cout << "age: " << age << endl;
    cout << "address:" << addr << endl;
  }
};
```

新增成员

0、继承与派生



- **继承**：一个新类从已有的类那里获得其已有的特性
- 在一个已存在的类的基础上建立一个新的类
- “基类”（base class）或“父类”
- “派生类”（derived class）或”子类”
- 派生类继承了基类的所有数据成员和成员函数
- 一个基类可以派生出多个派生类
- 继承层次结构：
  - 每个派生类可以作为基类再派生出新的子类
- 单继承VS多重继承
- 派生类是基类的具体化
- 基类是派生类的抽象



1、派生类的声明方式

使用派生类要先声明，声明的格式为

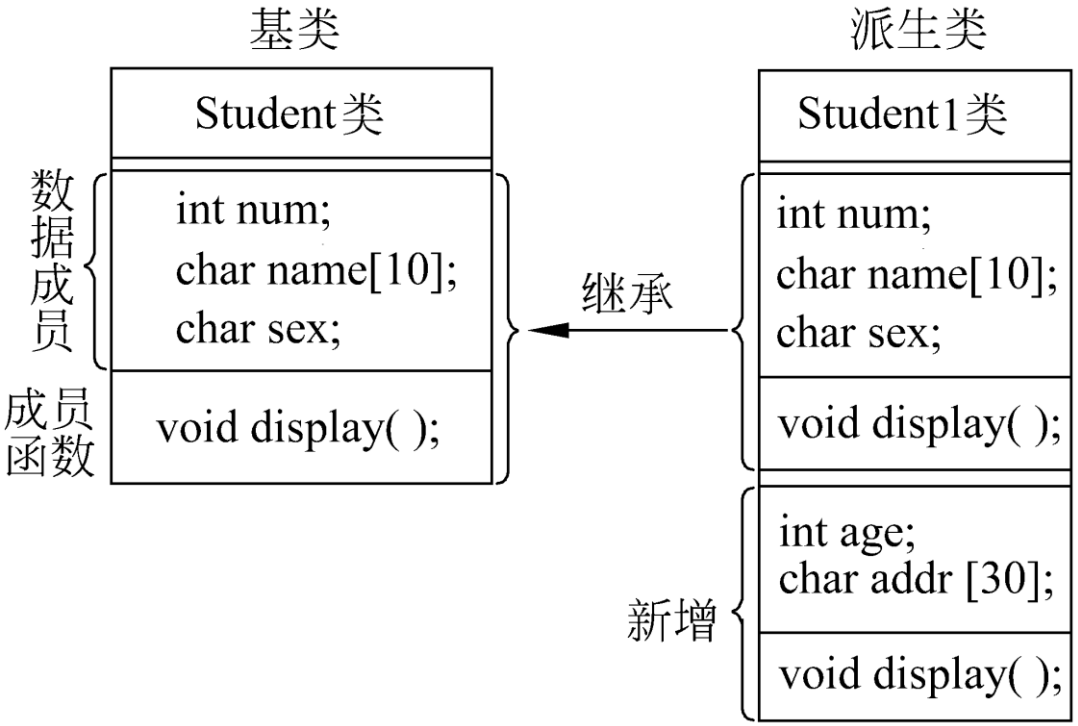
class 派生类名: [继承方式] 基类名
{ 派生类新增成员声明 };

继承方式包括：public（公用）、private（私有）、protected（保护）
如果省略，系统默认为**private**。

```
class Student1 : public Student
{
private:
  int age;
  string addr;

public:
  void display_1()
  {
    cout << "age: " << age << endl;
    cout << "address: " << addr << endl;
  }
};
```

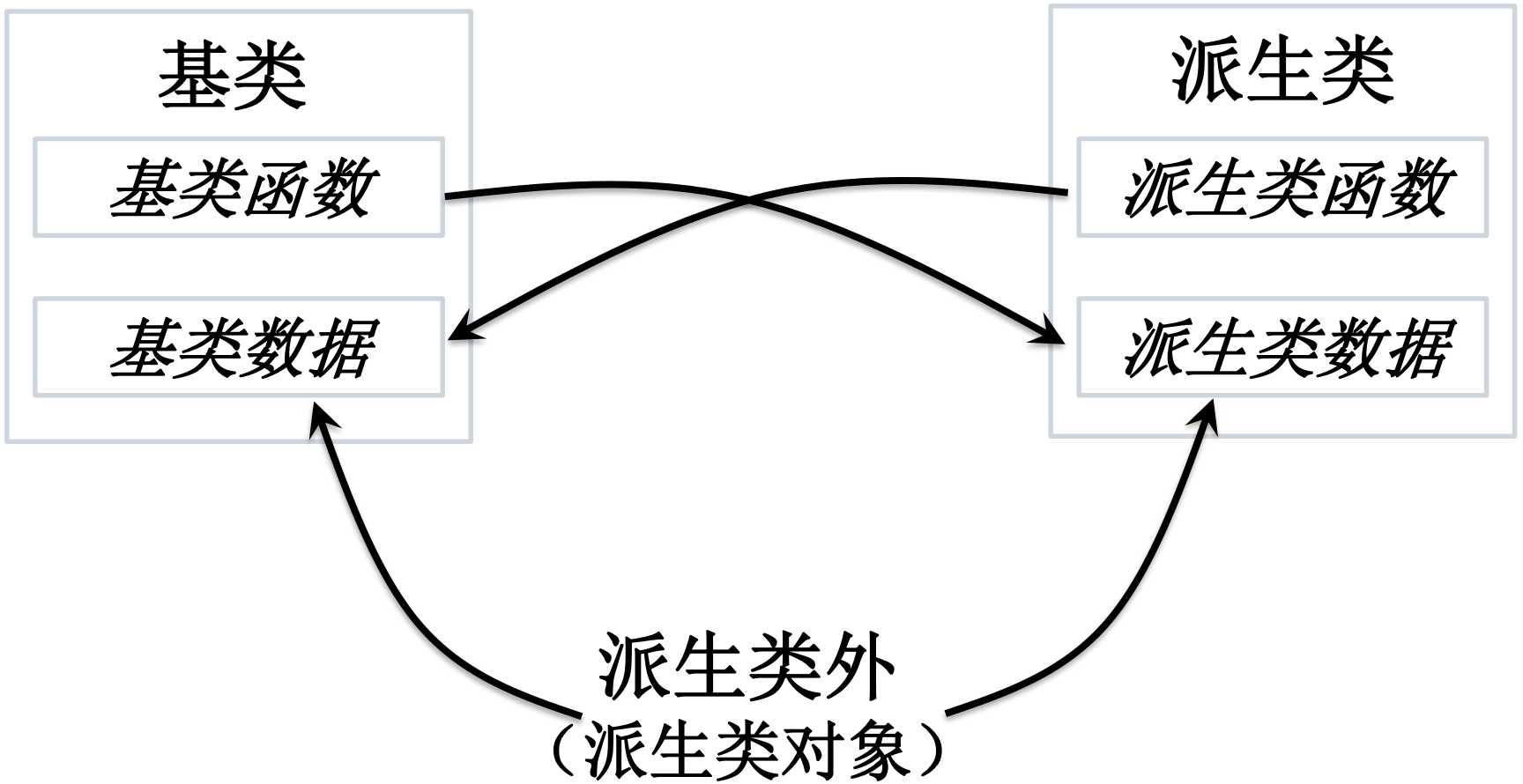
2、派生类的构成

- 从基类接受成员
  - 派生类将基类除构造函数和析构函数外的**所有**成员接收过来
- 调整从基类接受的成员
  - 在派生类中声明一个与基类成员同名的成员**屏蔽**基类的同名成员，注意如是成员函数不仅要函数名相同，而且函数的参数也要相同，屏蔽的含义是用新成员取代旧成员。
- 在声明派生类时增加的成员
  - 体现了派生类对基类功能的扩充。
  - 在声明派生类时，还要自己定义派生类的构造函数、析构函数。



3、派生类成员的访问属性

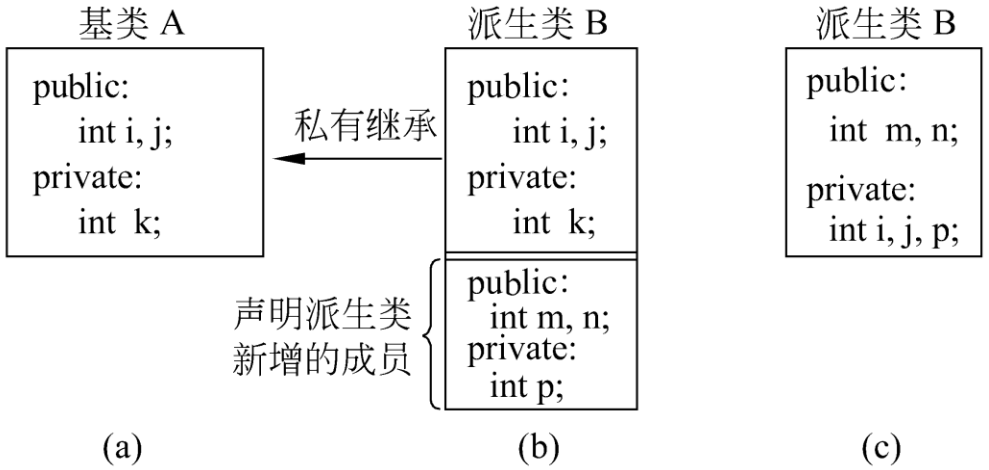
- 需要考虑以下问题



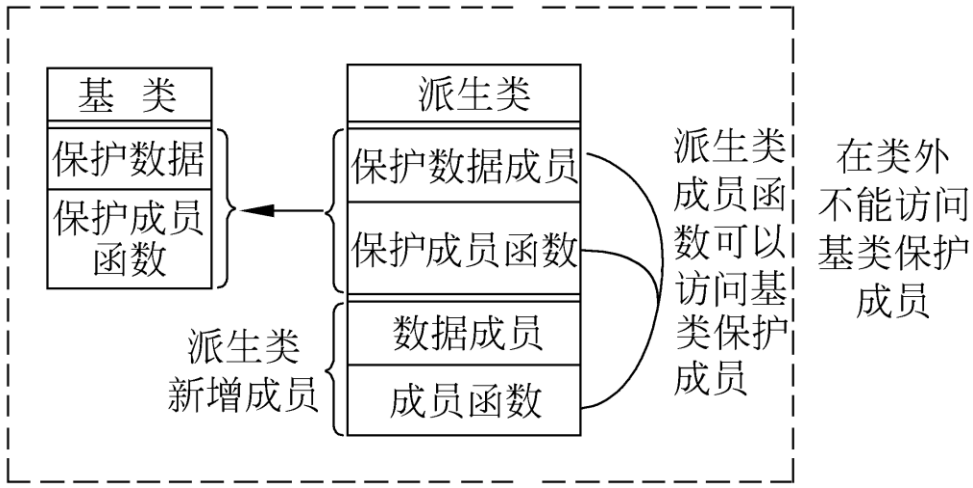
3、派生类成员的访问属性

- 公用继承（public inheritance）
  - `class Student1 : public Student`
- 私有继承（private inheritance）
  - `class Student1 : private Student`
- 受保护的继承（protected inheritance）
  - `class Student1 : protected Student`

私有继承



保护继承



3、派生类成员的访问属性

基类中的访问属性	继承方式	派生类中的访问属性
私有	公用	不可访问
私有	私有	不可访问
私有	保护	不可访问
公用	公用	公用
公用	私有	私有
公用	保护	保护
保护	公用	保护
保护	私有	私有
保护	保护	保护

3、派生类成员的访问属性

派生类中访问属性	在派生类中	在派生类外	在下层公用派生类中
公用	可以	可以	可以
保护	可以	不可以	可以
私有	可以	不可以	不可以
不可访问	不可以	不可以	不可以

3、派生类成员的访问属性

□ 公用继承（public inheritance）

```
class Student
{
private:
  int num;
  string name;
  char sex;

public:
  void set_value()
  {cin>>num>>name>>sex;}

  void display()
  {cout<<"num: "<<num<<endl;
   cout<<"name:
"<<name<<endl;
   cout<<"sex: "<<sex<<endl;}
};
```

```
class Student1: public Student
{private:
    int age;
    string addr;
public:
  void set_value_1()
  {cin>>age>>addr;}
  void display_1()
  {
  co
  cout<<"age: "<<age<<endl;  //  正确
  cout<<"address: "<<addr<<endl;}
};
```

派生类中使用基类私有成员

```
// 方法一：
int main()
{ Student1 stud1;
stud1.display();
stud1.display_1();
return 0;
}

// 方法二：
void display
{display();
cout<<"age: "<<age<<endl;
cout<<"address: "<<addr<<endl;}
```

派生类外调用基类公用成员函数

派生类中调用基类公用成员函数

3、派生类成员的访问属性

□ 私有继承（private inheritance）

```
class Student1: private Student
{private:
  int age;
  string addr;
public:
void display_1()
  {  display();
  cout<<"age: "<<age<<endl;
  cout<<"address: "<<addr<<endl;}
};
```

私有继承

私有派生类中使用基类公用成员

```
int main()
{ Student1 stud1;
  stud1.display_1();
  return 0;}
```

①在main函数中调用派生类的公有成员函数stud1.display_1

②通过该函数调用基类的公有成员函数display

③通过基类的公有成员函数display访问基类的私有数据成员。

3、派生类成员的访问属性

- 受保护的继承（protected inheritance）
- 如果基类只进行了一次派生，则保护继承和私有继承的功能完全相同，但**保护继承可以进一步派生，而私有继承则不可以，两者具有实质性差别**

```
class Student
{protected:
  int num;
  string name;
  char sex;
public:
  void display( );
};
```

基类保护成员

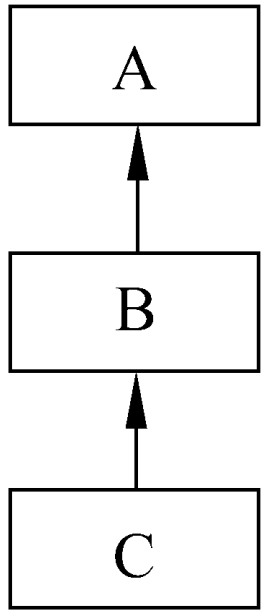
```
class Student1: protected Student
{private:
  int age;
  string addr;
public:
  void display1( )
  {cout<<"num: "<<num<<endl;
  cout<<"name: "<<name<<endl;
  cout<<"sex: "<<sex<<endl;
  cout<<"age: "<<age<<endl;
  cout<<"address: "<<addr<<endl;}
};
```

保护继承

保护派生类中使用基类保护成员

3、派生类成员的访问属性

- 多级派生时的访问属性
- 类B是类A的直接派生类，类C 是类A 的间接派生类
- 类 A是类B的直接基类，是类C的间接基类




```
class A //基类
{
private:
  int ka;
public:
  int ia;
protected:
  void fa();
  int ja;
};
```

```
class B:public A //public派生类
{
private:
  int mb;
public:
  void fb1();
protected:
  void fb2();
};
```

```
class C:protected B //protected派生类
{
private:
  int nc;
public:
  void fc1();
};
```

3、派生类成员的访问属性

```
class A //基类
{
private:
  int ka;
public:
  int ia;
protected:
  void fa();
  int ja;
};
```

```
class B:public A //public派生类
{
private:
  int mb;
public:
  void fb1();
protected:
  void fb2();
};
```

```
class C:protected B //protected派生类
{
private:
  int nc;
public:
  void fc1();
};
```

	ia	fa	ja	ka	fb1	fb2	mb	fc1	nc
基类A	公有	保护	保护	私有					
公用派生类B					公有	保护	私有		
保护派生类C								公有	私有

4、派生类的构造函数与析构函数

- 派生类构造函数：不仅要考虑派生类所增加的数据成员的初始化，还应当考虑基类的数据成员初始化。

- 基本思路：在执行派生类的构造函数时，调用基类的构造函数。

- 派生类的构造函数一般形式：

派生类名::派生类名(基类所需的形参，本类成员所需的形参):基类名(基类参数表)

```
{
    本类成员初始化赋值语句;
};
```

- 用派生类构造函数的形参做基类构造函数的实参。

4、派生类的构造函数与析构函数

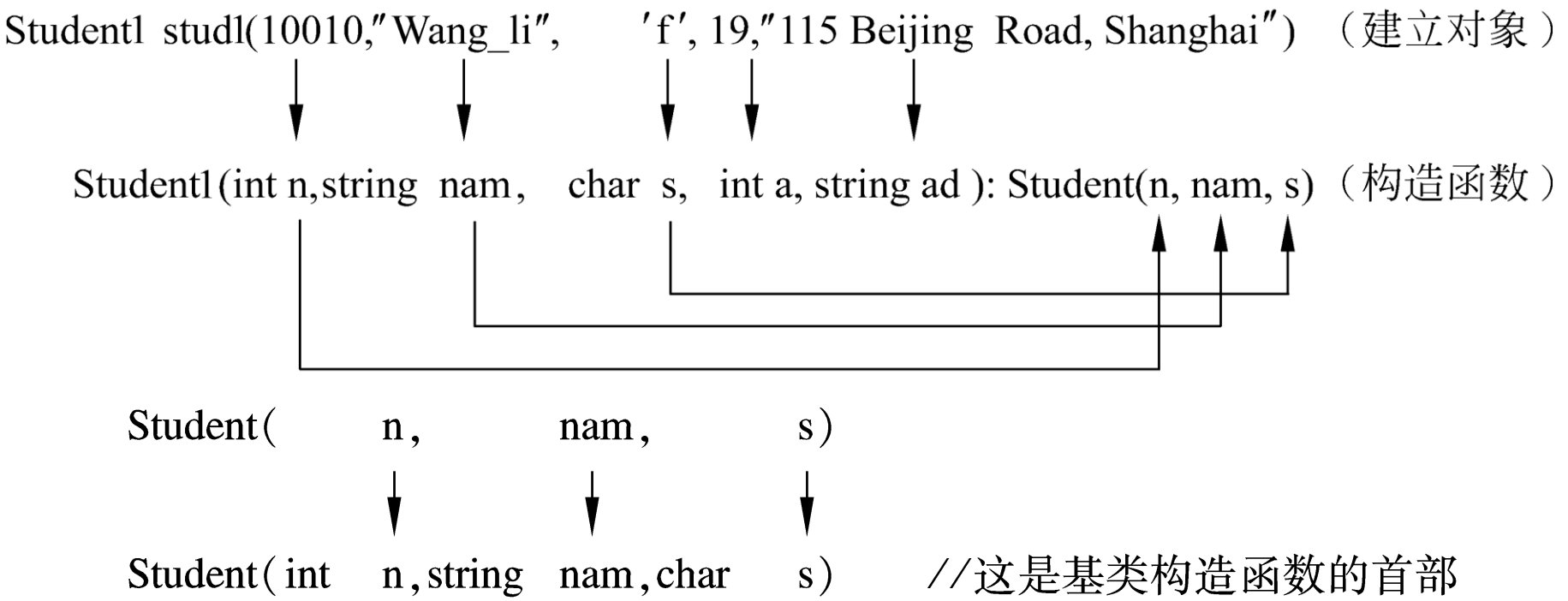
□ 简单的派生类的构造函数

```
class Student //声明基类
{
public: //公用成员
 //基类构造函数
  Student(int n, string nam, char s)
  {
    num = n;
    name = nam;
    sex = s;
  }
  ~Student() {} //基类析构函数
protected: //保护成员
  int num;
  string name;
  char sex;
};
```

```
class Student1:public Student //声明公用派生类
{
public:
  Student1(int n,string nam,char s,int a,char ad[]) :
Student(n, nam, s) //派生类构造函数
  {
    age = a; //  只对派生类新增的数据成员初始化
    addr = ad;
  }
  void show();
private: //  派生类的私有部分
  int age;
  string addr;
};
```

4、派生类的构造函数与析构函数

```
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;
}
```



4、派生类的构造函数与析构函数

- 在建立一个对象时，执行构造函数的顺序是：

①派生类构造函数先调用基类构造函数；

②再执行派生类构造函数本身（即派生类构造函数的函数体）。

- 按上面的例子，先初始化num，name，sex，然后再初始化age和addr。
- 释放派生类对象时，先执行派生类析构函数，再执行其基类的析构函数。

4、派生类的构造函数与析构函数

□ 有子对象的派生类的构造函数

- 类的**数据成员**除了是标准类型或系统提供的类型如string外，**还可以是类类型**，如声明一个类时包含类类型的数据成员：

  Student s1；

  Student 是已声明过的类名，s1是该类的对象。

- 类对象中的内嵌对象称为**子对象（subobject），即对象中的对象**
- 如，除了可以在派生类student1中增加age、address成员外，还可以增加班长一项，而班长本身也是学生，他属于student类型，有学号和姓名等基本数据，班长这项就是派生类中的子对象。

4、派生类的构造函数与析构函数

有子对象的派生类的构造函数

- 不能在声明派生类时对子对象初始化，系统在建立派生类对象时调用派生类构造函数对子对象进行初始化。
- 派生类构造函数的任务包括：

1. 对基类数据成员初始化
2. 对子对象的数据成员初始化
3. 对派生类的数据成员初始化

派生类构造函数一般形式：

派生类名::派生类名 (总参数表):基类名(实参表), 子对象名(参数表)

```
{
        派生类新增成员的初始化语句;
}
```

4、派生类的构造函数与析构函数

- 执行派生类构造函数的顺序是：

①调用基类构造函数，初始化基类数据成员

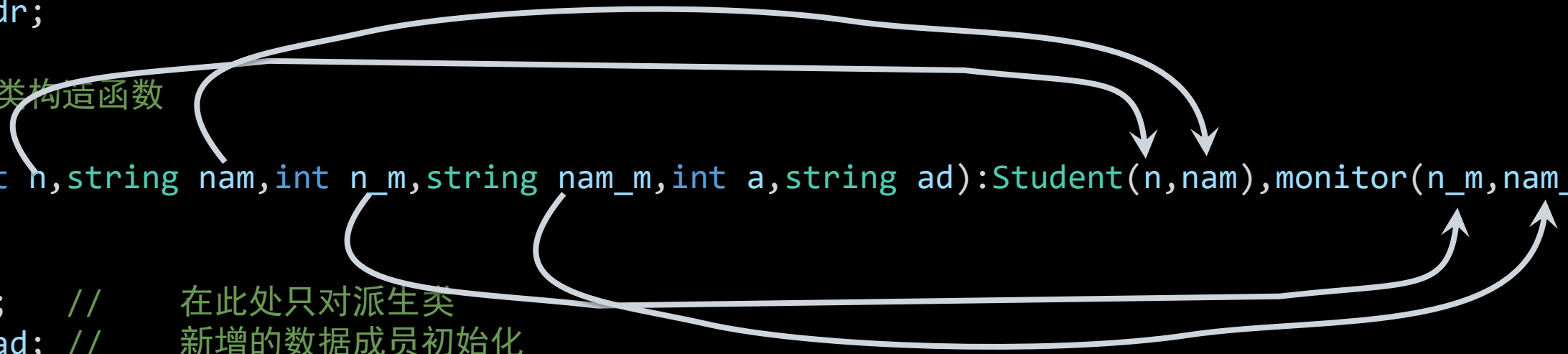
②调用子对象构造函数，初始化子对象数据成员

③执行派生类构造函数，初始化派生类数据成员

- 编译系统在此根据参数名(而不是参数的顺序）决定各参数表中参数之间的传递关系。如有多个子对象，要逐个列出子对象及其参数表。

4、派生类的构造函数与析构函数

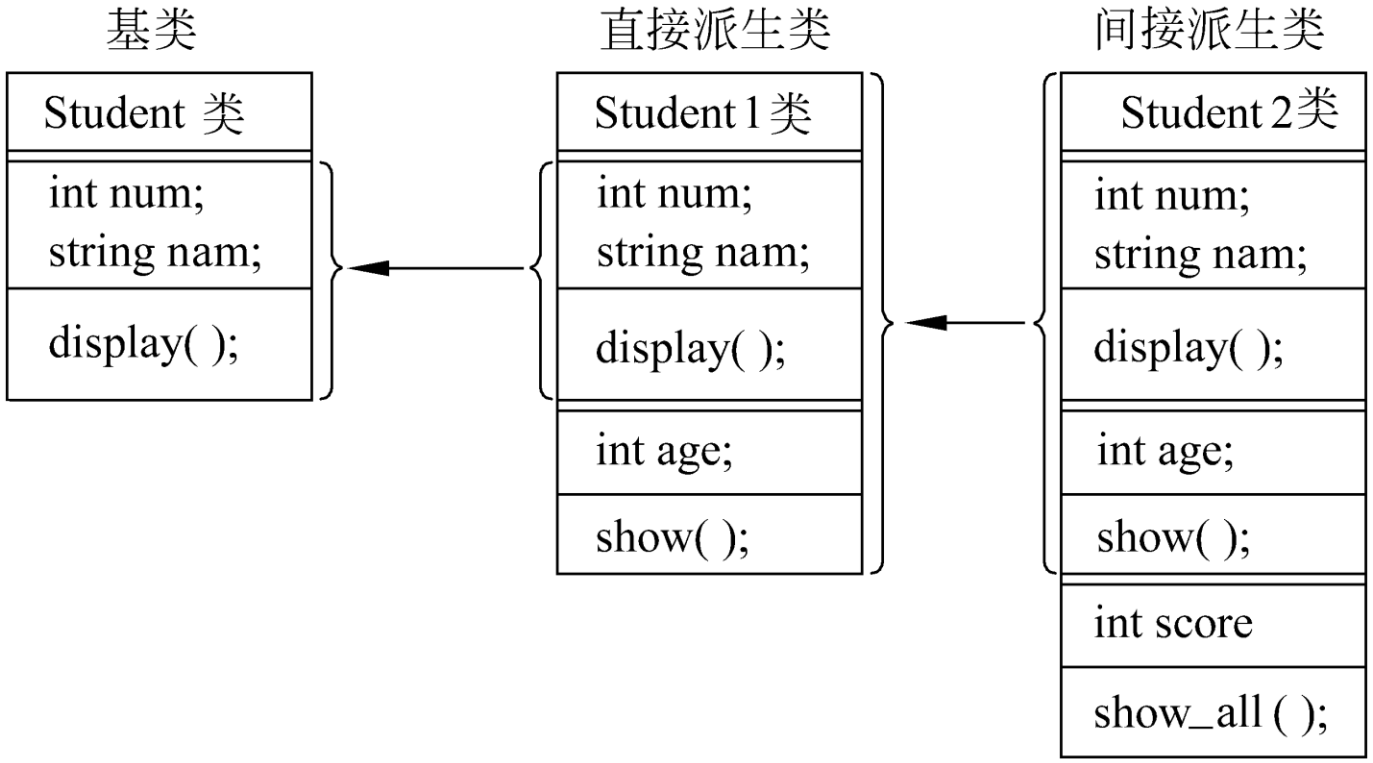
```
class Student1 : public Student //   public继承方式
{
private:              //  派生类的私有数据
  Student monitor; //  定义子对象(班长)
  int age;
  string addr;
public:
//下面是派生类构造函数

Student1(int n,string nam,int n_m,string nam_m,int a,string ad):Student(n,nam),monitor(n_m,nam_m)

  {
    age = a;    //     在此处只对派生类
    addr = ad; //     新增的数据成员初始化
  }
};
```



```
Student1 stud1(10010, "Lisa Wang",
               10001, "Francis Zhang",
               19, "115 Beijing Road, Shanghai");
```

4、派生类的构造函数与析构函数

- 多层派生时的构造函数
- 一个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。多层派生时怎样写派生类的构造函数？现有如图所示的多层派生类：



4、派生类的构造函数与析构函数

□ 多层派生时的构造函数

可以按照前面派生类构造函数的规则逐层写出各个派生类的构造函数。

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);
```

4、派生类的构造函数与析构函数

- 多层派生时的构造函数
- 派生类构造函数的规则：只须调用其直接基类的构造函数即可，不要列出每一层派生类的构造函数。
- 在声明Student2类对象时，调用Student2构造函数，在执行Student2构造函数时，先调用Student1构造函数，
- 在执行Student1构造函数时，先调用基类Student构造函数。

- 初始化的顺序是：
  ①先初始化基类的数据成员num和name
  ②再初始化Student1的数据成员age
  ③最后初始化Student2的数据成员score

4、派生类的构造函数与析构函数

□ 派生类构造函数的特殊形式

(1) 当不需要对派生类新增成员进行初始化时，派生类构造函数的函数体可以为空。

(2) 如果在基类里没有定义构造函数，或定义了没有参数的构造函数，在定义派生类构造函数时可以不写基类构造函数。因为此时派生类构造函数没有向基类构造函数传递参数的任务。在调用派生类构造函数时，系统地自动首先调用基类的默认构造函数。

(3) 如果在基类和子对象的类中都没有定义带参数的构造函数，也不需要对派生类自己的数据成员进行初始化，可以不定义派生类构造函数。

(4) 如果在基类或子对象的类声明里定义了带参数的构造函数，就必须定义派生类构造函数，并在派生类构造函数中写出基类或子对象类的构造函数及其参数表。

(5) 如果在基类构造函数重载，在定义派生类构造函数时，既可以包含基类构造函数及其参数，也可以不包含基类构造函数。

4、派生类的构造函数与析构函数

□ 派生类的析构函数

1. 调用顺序与构造函数正好相反
2. 首先执行派生类自己的析构函数，清理派生类新增加的成员
3. 然后调用子对象类的析构函数清理子对象
4. 最后调用基类析构函数清理基类的成员

5、多重继承

- 多重继承的定义和声明

- 一个派生类同时继承多个基类，这种行为称为多重继承（multiple inheritance）
- 假定已声明了类A，类B和类C，由它们派生出新类D，声明的形式可以是：

```
class D: public A，private B, protected C
{ D类新增的成员声明 }
```

- D以不同继承方式分别继承ABC，需确定各基类成员的访问权限

5、多重继承

- 多重继承派生类的构造函数
- 多重继承派生类的构造函数在初始化表中包括多个基类构造函数，假定派生类有三个基类，它的构造函数形式是：

派生类构造函数名(总参数表)：基类1构造函数(参数表),
基类2构造函数(参数表),
基类3构造函数(参数表)
{ 派生类新增成员初始化语句 }

- 各基类的排列顺序不分先后，系统调用基类构造函数的顺序就是声明派生类时基类的出现顺序

5、多重继承

□ 多重继承派生类的构造函数

```
class Graduate : public Teacher, public Student
{
private:
  float wage; //  工资
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() //  输出人员的有关数据
  {
    cout << "name:" << name << endl;
    cout << "age:" << age << endl;
    cout << "sex:" << sex << endl;
    cout << "score:" << score << endl;
    cout << "title:" << title << endl;
    cout << "wages:" << wage << endl;
  }
};
```

多重继承

构造函数

```
int main()
{
  Graduate grad1("Lisa Wang", 24, 'F',
"Assistant", 89.5, 1234.5);
  grad1.show();
  return 0;
}
```

5、多重继承

- 多重继承引起的二义性问题
- 同名隐藏规则：当派生类与基类中有相同成员时，若未强行指名，则通过派生类对象使用的是派生类中的同名成员
- 如要通过派生类对象访问基类中被覆盖的同名成员，应使用基类名限定。

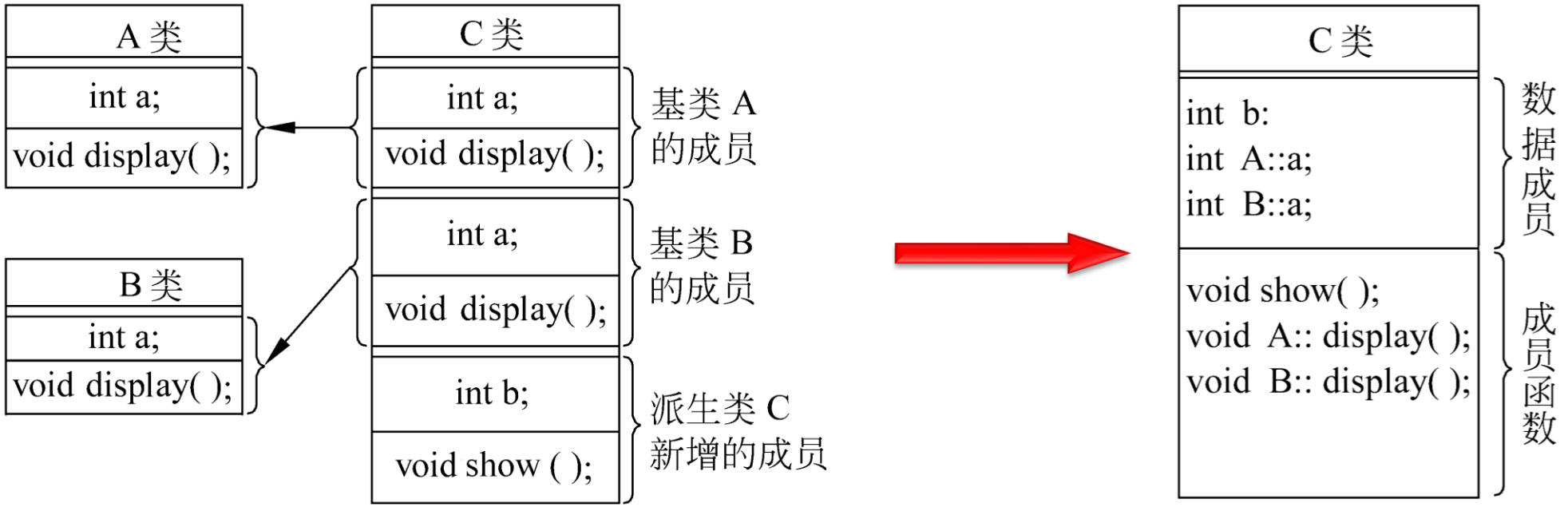
```
class A
{public:
  void f();
};
class B
{public:
  void f();
  void g();
};
```

```
class C : public A, public B
{public:
  void g();
  void h();
};

C c1;
c1.f(); // 有二义
c1.g(); // 无二义，同名覆盖
```

```
//解决方法一：
 c1.A::f();
 c1.B::f();

 //解决方法二：
 同名覆盖
```

5、多重继承

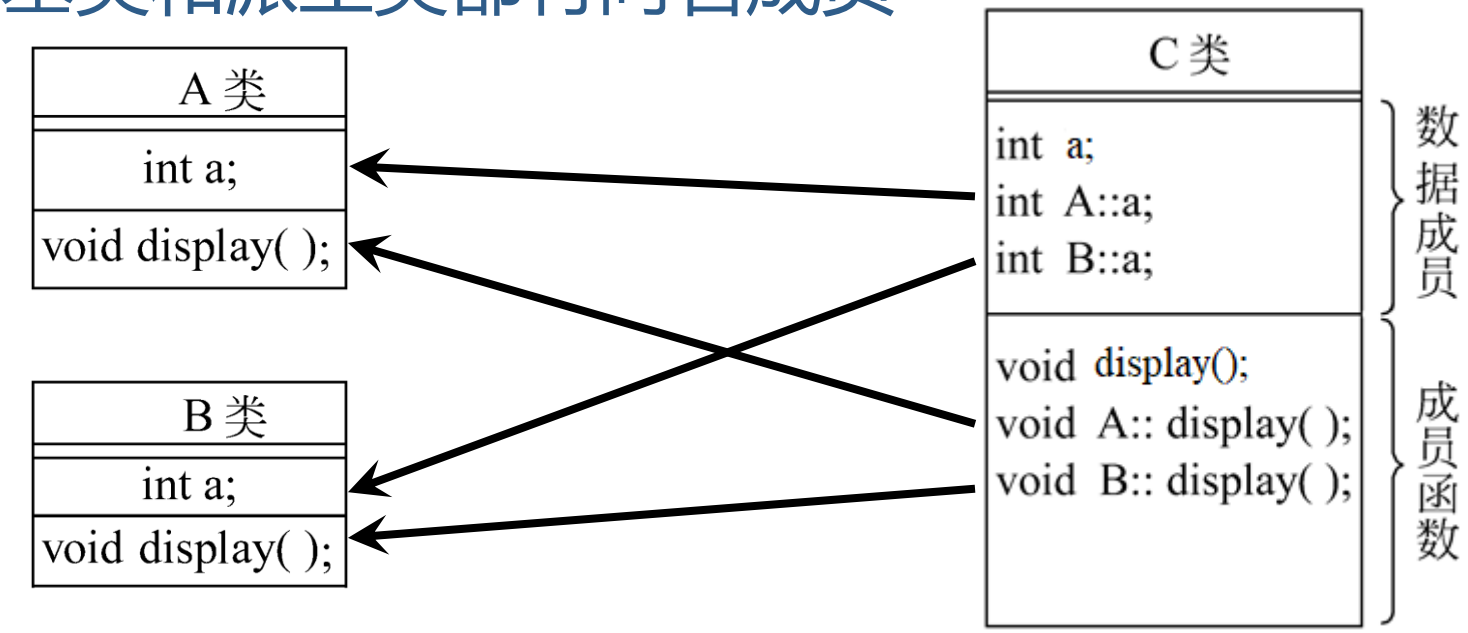
- 多重继承引起的二义性问题
- （1）两个基类有同名成员



- （2）两个基类和派生类都有同名成员
- （3）两个基类由一个共同基类派生

5、多重继承

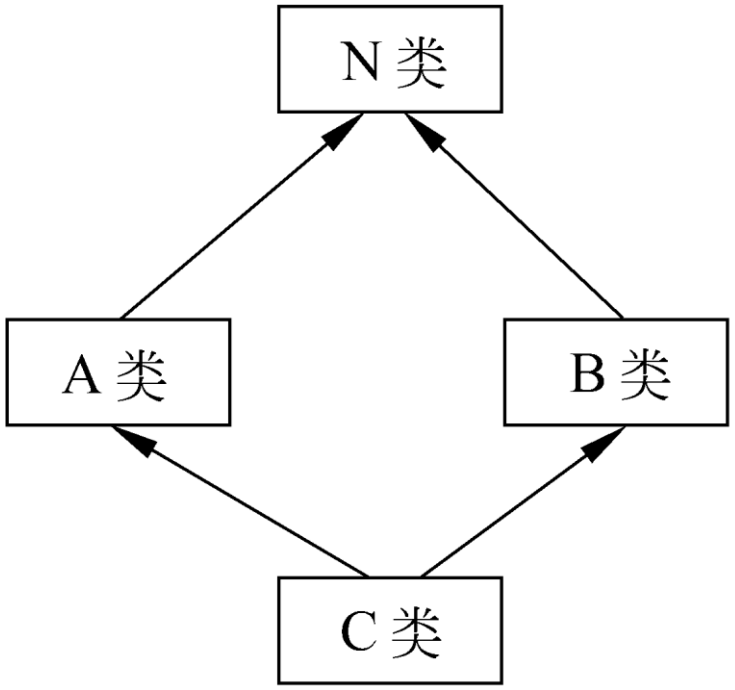
- 多重继承引起的二义性问题
- （1）两个基类有同名成员
- （2）两个基类和派生类都有同名成员

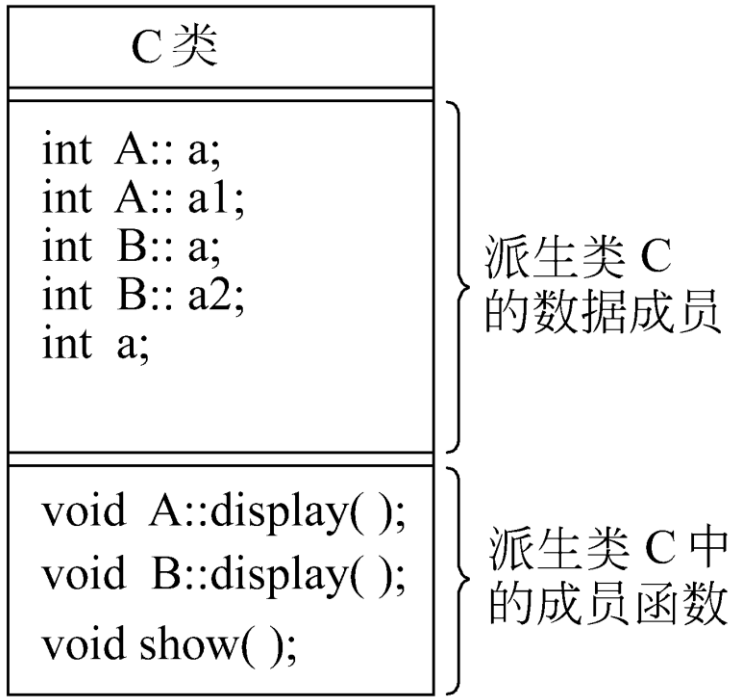


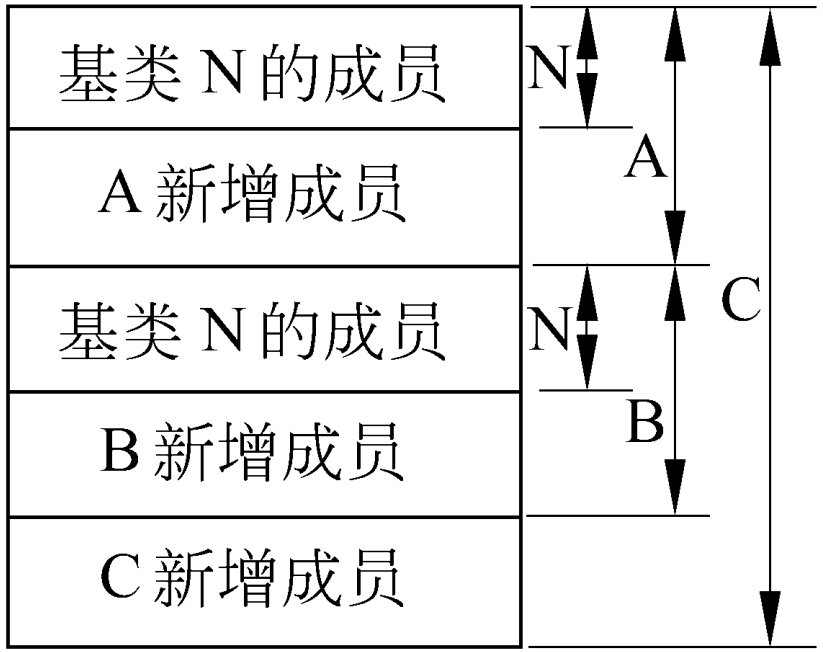
- （3）两个基类由一个共同基类派生

5、多重继承

- 多重继承引起的二义性问题
- （1）两个基类有同名成员
- （2）两个基类和派生类都有同名成员
- （3）两个基类由一个共同基类派生







5、多重继承

- 多重继承引起的二义性问题
- 在多重继承时，基类与派生类之间，或基类之间出现同名成员时，将出现访问时的二义性（不确定性）——采用虚函数或同名隐藏规则来解决。
- 当派生类从多个基类派生，而这些基类又从同一个基类派生，则在访问此共同基类中的成员时，将产生二义性——采用虚基类来解决。

5、多重继承

- 在继承间接共同基类时减少数据冗余——**虚基类**
- 如果不希望在派生类中保留间接共同基类的多个同名成员，C++提供了虚基类的方法，使派生类在继承间接共同基类时**只保留一份成员**。
- 虚基类
  - 用于有共同基类的场合
  - 声明
    - 以virtual修饰说明基类
    - class B1:virtual public B
  - 作用
    - 主要用来解决多层继承时可能发生的对同一基类继承多次而产生的二义性问题
    - 为最低层的派生类提供唯一的基类成员，而不重复产生多次拷贝
  - 注意
    - **在第一级继承时就要将共同基类设计为虚基类**

5、多重继承

□ 在继承间接共同基类时减少数据冗余——**虚基类**

声明虚基类的格式：

class 派生类名：virtual 继承方式 基类名

当基类通过多条派生路径被一个派生类继承时，派生类只继承该基类一次。

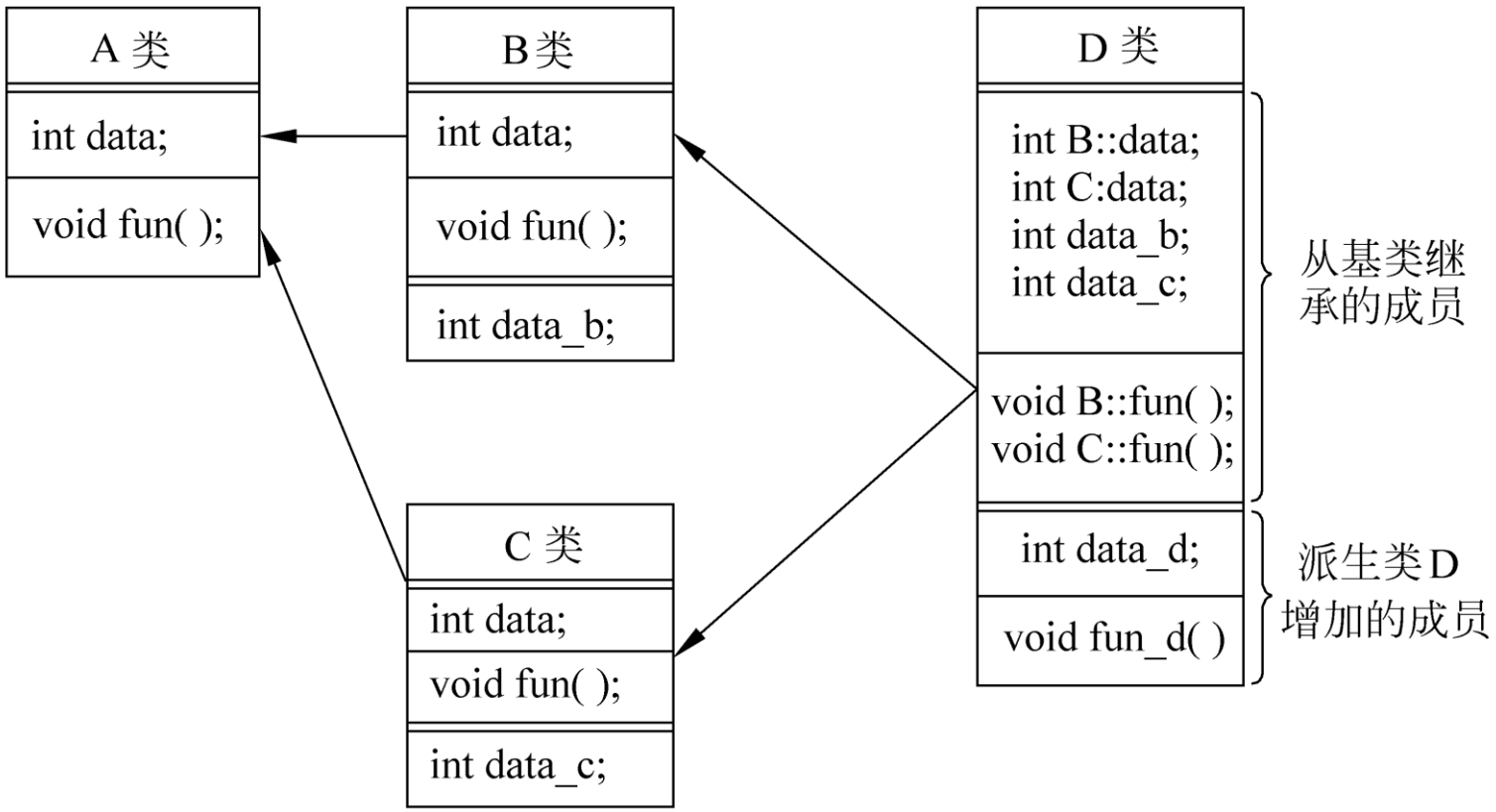
现将A类声明为虚基类：

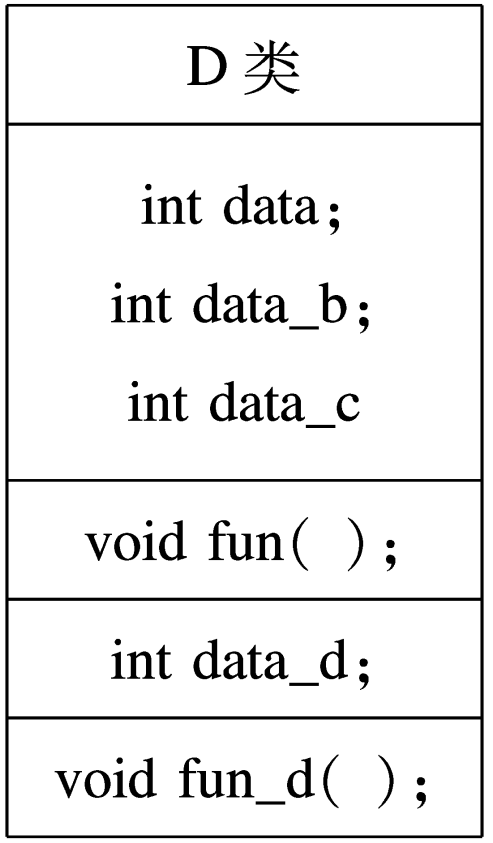
```
class  A
{… … };
class  B: virtual public A
{… … };
class  C: virtual public A
{… … };
```

虚基类是在声明派生类时，指定继承方式时声明的。因为一个基类可以作为一个派生类的虚基类同时也可以作为另一个派生类的非虚基类。

5、多重继承

□ 在继承间接共同基类时减少数据冗余——**虚基类**





5、多重继承

- 在继承间接共同基类时减少数据冗余——**虚基类**
- 如果在虚基类中定义了带参数的构造函数，而且没定义默认构造函数，要求在它的所有派生类（直接和间接）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 k) { } … … };
class  B: virtual public A
    {B (int n ):A(n){ }… … };
class  C: virtual public A
    {C (int n ):A(n){ } … … };
 class  D: public B, public C
    {D (int n ):A(n),B(n),C(n) { } … … };
```



5、多重继承

- 在继承间接共同基类时减少数据冗余——**虚基类**
- 最后的派生类不仅要负责对其直接基类初始化，还要负责对虚基类初始化。
- 只执行最后的派生类调用虚基类的构造函数，忽略虚基类其他派生类(如类B和类C)调用虚基类构造函数，保证对虚基类的数据成员只作一次初始化。

```
class Graduate : public Teacher, public Student
//    声明Teacher和Student类为公用继承的直接基类
{
public:
  Graduate(char *nam, char s, int a, char *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)
  {}
};
```

6、基类与派生类的转换

- 在3种继承方式中，只有公用派生类才是基类真正的子类型，它完整的继承了基类的功能。
- 基类与派生类对象之间有**赋值兼容关系**，具体表现为：
- （1）派生类对象可以**向基类对象赋值**
  - 不能用基类对象对子类对象赋值，不同子类对象之间也不能赋值
- （2）派生类对象可以替代基类对象**向基类对象的引用进行赋值或初始化**
- （3）如果函数的参数是基类对象或基类对象的**引用**，相应的**实参可以用子类对象**
- （4）派生类对象的地址可以赋给指向**基类对象的指针变量**
  - 指向基类对象的指针也可以用来指向派生类对象

6、基类与派生类的转换

```
Student(int n, string nam, float s) : num(n), name(nam), score(s) {}
```

```
class Graduate : public Student
{
public:
  Graduate(int n, string nam, float s, float w) : Student(n, nam, s), wage(s) {}
  void display();
private:
  float wage;
};
void Graduate::display()
{
  Student::display();
  cout <<"wage="<<wage<<endl;
}
```

```
int main(){Student stud1(1001,"Li",87.5);
  Graduate grad1(2001,"Wang",98.5,1000);
  Student *pt=&stud1;
  pt->display();
  pt = &grad1;
  pt->display();}
```

无法访问派生类增加的成员

```
num: 1001

name: Li

score: 87.5

num: 2001

name: Wang

score: 98.5
Press any key to continue . . .
```

7、继承与组合

- 类的组合和继承都是软件重用的重要方式
- 类的组合（composition）：一个类中以另一个类的对象作为数据成员
- 继承：建立了派生类与基类的关系，是一种“**是**”的关系
- 继承是**纵向**的
  - 如：“白猫是猫”，“黑人是人”
- 组合：建立了成员类与组合类（或称复合类）的关系，是“**有**”的关系
- 组合是**横向**的
  - 如：教授有一个生日属性

7、继承与组合

- 类的组合和继承都是软件重用的重要方式
- 组合：建立了成员类与组合类（或称复合类）的关系，是“**有**”的关系
  - 如：教授有一个生日属性

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

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

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

8、继承在软件开发中的重要意义

- 继承是C++和C的最重要的区别之一
- 软件重用缩短了开发周期，继承机制使得开发者可以使用类库
- 继承 **VS** 修改已有类的原因
- （1）继承不改变基类本身，**不会影响其他程序**的使用
- （2）使用类库时，无法获得基类的**源代码**，只有接口
- （3）使用类库时，**基类已与其他组件有关联**，不允许修改
- （4）使用类库时，**基类是抽象类**无独立功能
- （5）面向对象程序设计，需要设计类的**层次结构**，是不断从抽象到具体的过程

小结

- 继承与派生
- 派生类的声明范式
- 派生类的构成
- 派生类成员的访问属性
- 派生类的构造函数和析构函数
- 多重继承
- 基类与派生类的转换
- 继承与组合
- 继承在软件开发中的重要意义