第5讲 继承与派生

面向对象程序设计有4个主要特点: 抽象、封装、继承和多态性。

面向对象技术强调软件的可重用性(software reusability) 。C++语言提供了类的继承机制，解决了软件重用问题。

5.1 继承与派生的概念

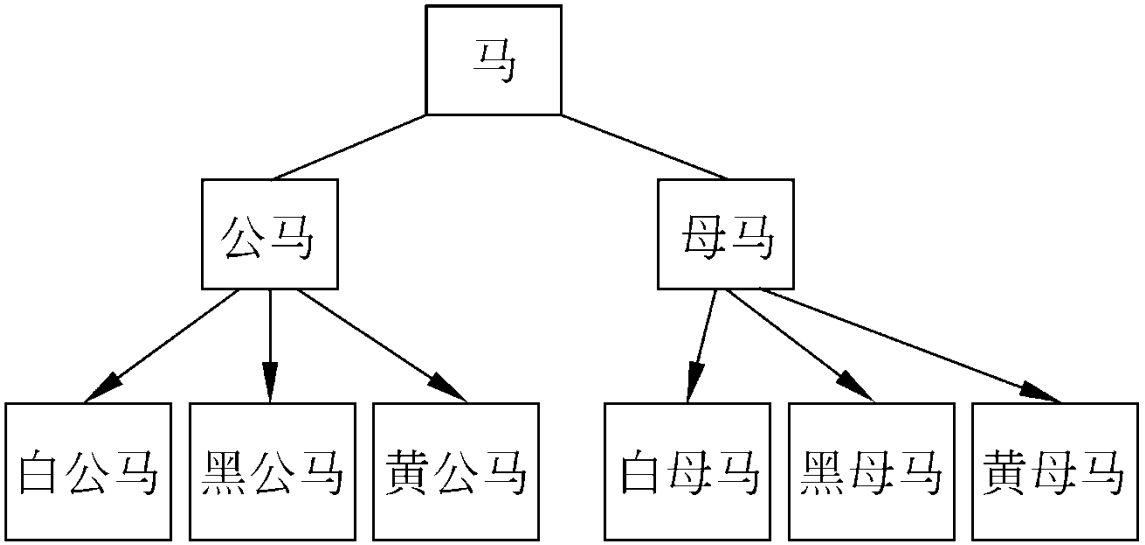
继承(inheritance)是C++的一个重要组成部分。

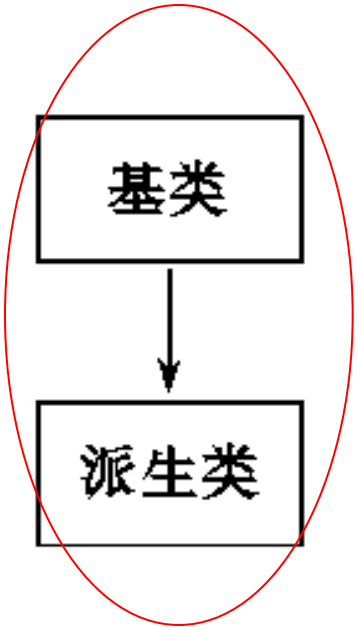
一个类中包含了若干数据成员和成员函数。在不同的类中，数据成员和成员函数是不相同的。

但有时两个类的内容基本相同或有一部分相同。利用原来声明的类作为基础，再加上新的内容即可，以减少重复的工作量。 C++提供的继承机制就是为了解决这个问题。

在C++中，所谓“继承”就是在一个已存在的类的基础上建立一个新的类。

已存在的类(例如“马”)称为“基类(base class)”或“父类(father class)”。新建立的类(例如“公马”)称为“派生类(derived class)”或“子类(son class)”。见下图。



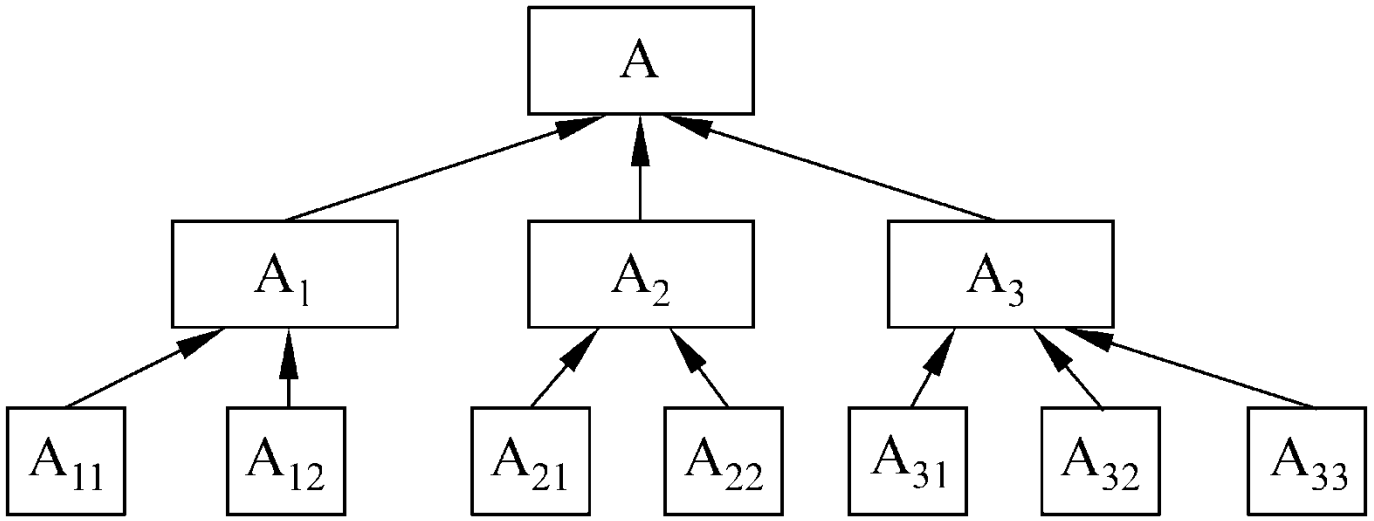


类的继承：一个新建子类从已有的父类那里获得父类的特性。

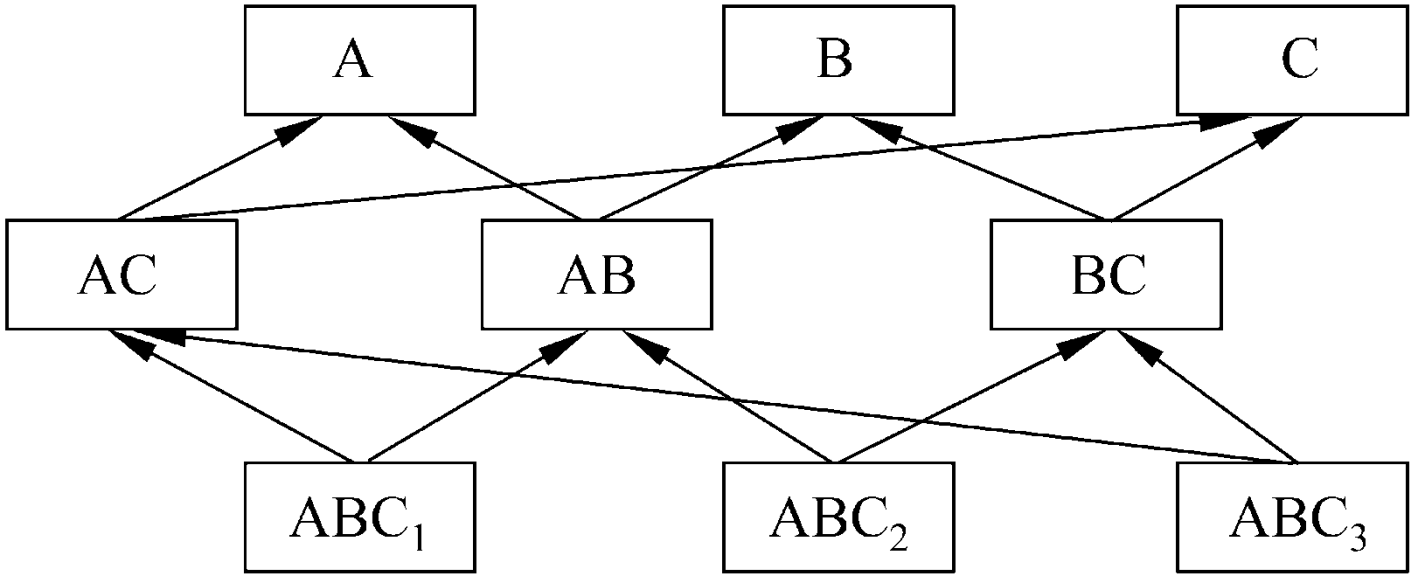
类的派生：从已有的类(父类)产生一个新的子类。

两种继承：

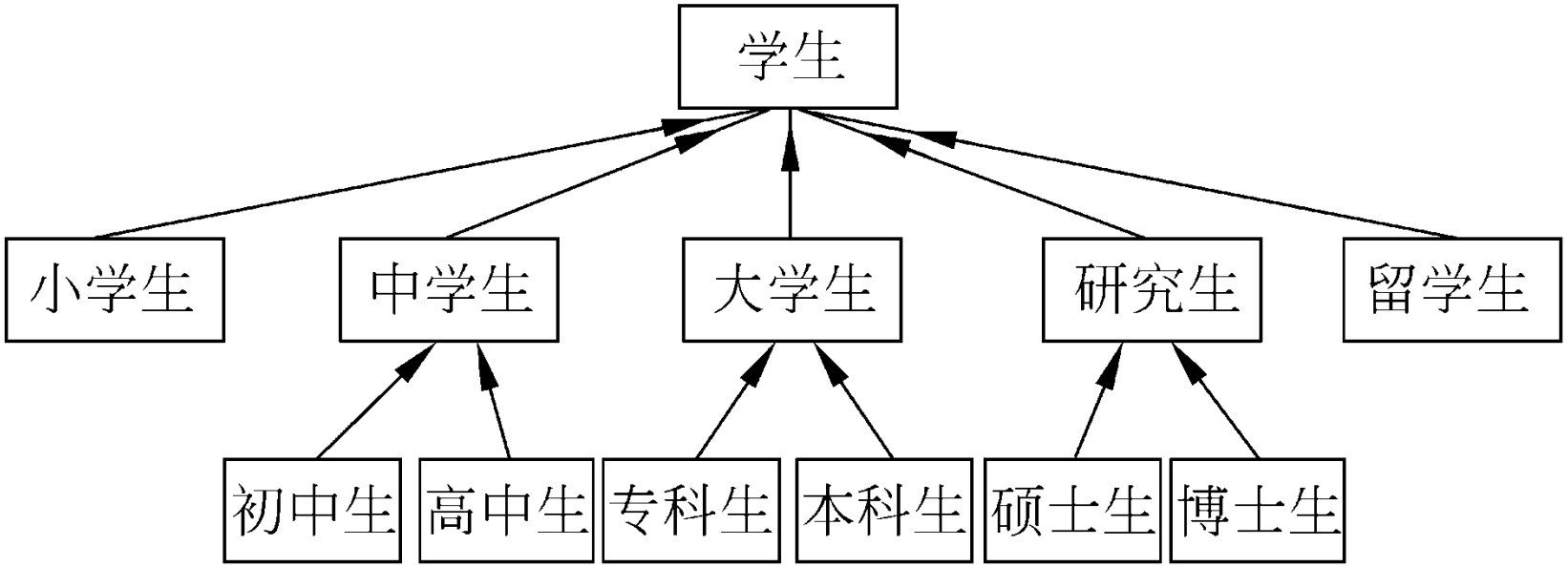
1.单继承(single inheritance)：一个派生类只从一个基类派生，这种继承关系所形成的层次是一个树形结构，如下图。**注意：箭头表示继承的方向，从派生类指向基类。**



2.多重继承(multiple inheritance)： 一个派生类有两个或多个基类。这种继承关系所形成的结构如下图。



基类和派生类的关系：派生类是基类的具体化，而基类则是派生类的抽象。



5.2 派生类的声明方式

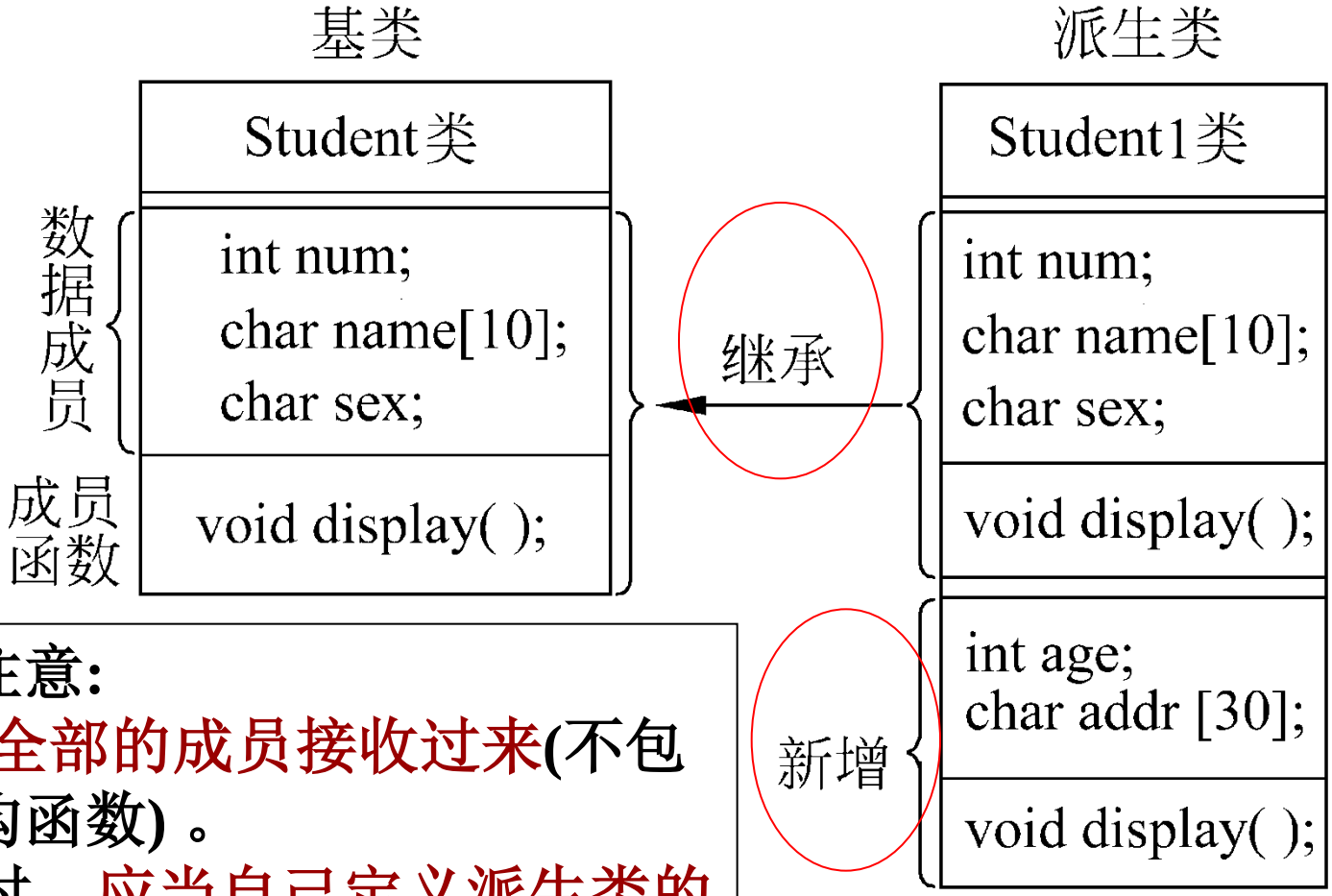
假设已经声明了一个基类Student，在此基础上通过单继承建立一个派生类Student1:

```
class Student1: public Student
//声明基类是Student，public表示继承方式为公有继承
{public:
void display_1( ) //新增加的成员函数
{cout<<"age: "<<age<<endl;
cout<<"address: "<<addr<<endl;}
private:
int age; //新增加的数据成员
string addr; //新增加的数据成员
};
```

继承方式将影响派生类对基类属性的访问

继承方式包括:
1、public(公用的)
2、private(私有的)
3、protected(受保护的)
此项是可选的，如果不写此项，默认为private(私有的)。

5.3 派生类的构成



构造一个派生类注意：

(1) 派生类把基类全部的成员接收过来(不包括构造函数和析构函数)。

(2)在声明派生类时，应当自己定义派生类的构造函数和析构函数，因为构造函数和析构函数是不能从基类继承的。

5.4 派生类成员的访问属性

5.4.1 公用继承

继承方式为public的，称为公用继承。

1、基类的公用成员和保护成员在派生类中仍然保持其公用成员和保护成员的属性。

2、基类的私有成员在派生类中并没有成为派生类的私有成员，它仍然只有基类的成员函数可以引用它。

（对照P160表5.1）

例5.1 访问基类的公有成员。

```
Class Student//声明基类
{public: //基类公用成员
void get_value( )
{cin>>num>>name>>sex;}
void display( )
{cout<<" num: "<<num<<endl;
cout<<" name: "<<name<<endl;
cout<<" sex: "<<sex<<endl;}
private ://基类私有成员
   int num;
   string name;
   char sex;
};
```

```
class Student1: public Student//以public方式声明派生类Student1
{public:
void display_1( )
{display( );
cout<<" num: "<<num<<endl; //引用基类的私有成员，错误
 cout<<" name: "<<name<<endl; //引用基类的私有成员，错误
 cout<<" sex: "<<sex<<endl; //引用基类的私有成员，错误
 cout<<" age: "<<age<<endl; //引用派生类的私有成员，正确
 cout<<" address: "<<addr<<endl;}//引用派生类的私有成员，正确
private:
int age;
string addr;
 };
```

正确：在派生类的display_1函数中调用基类的display函数

```
int main( )
{Student1  stud1;//定义一个Student1类的对象stud1
 stud1.display(); //基类的公用成员在派生类中仍是公有成员
stud1.display_1( ); //Display_1函数是Student1类的公用函数
stud1.num=1001; //错误。外界不能引用基类的私有成员
return 0;
 }
```

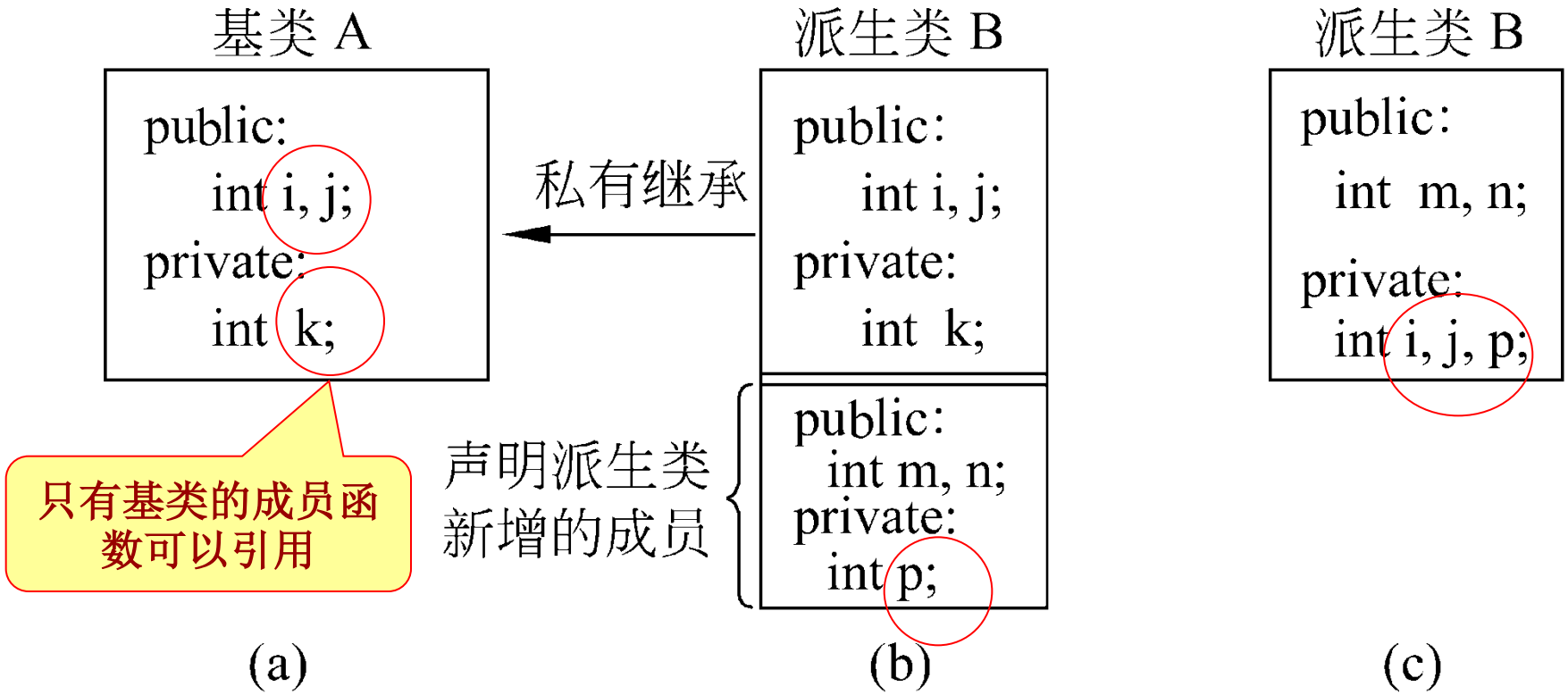
5.4.2 私有继承

继承方式为**private**的，称为私有继承。

1、基类的公用成员和保护成员相当于派生类中的私有成员，只有派生类的成员函数能访问它们。

2、基类的私有成员只有基类的成员函数可以引用它们。

（对照P163表5.2）


基类 A
public:
int i, j;
private:
int k;
只有基类的成员函数可以引用
(a)
私有继承
派生类 B
public:
int i, j;
private:
int k;
声明派生类新增的成员
public:
int m, n;
private:
int p;
(b)
派生类 B
public:
int m, n;
private:
int i, j, p;
(c)

例5.2 将例5.1中的公用继承方式改为用私有继承。

```
class Student1: private Student//用私有继承方式声明派生类Student1
{public:
void display_1( ) //输出两个数据成员的值
{cout<<"age: "<<age<<endl; //引用自己的私有成员，正确
cout<<"address: "<<addr<<endl;} //引用自己的私有成员，正确
private:
int age;
string addr;
};
int main( )
{Student1  stud1;//定义一个Student1类的对象stud1
 stud1.display(); //错误，基类的公用成员函数在派生类中是私有函数
stud1.display_1( ); //正确。Display_1函数是Student1类的公用函数
stud1.num=1001; //错误。外界不能引用基类的私有成员
return 0;
 }
```

私有继承

可以考虑通过派生类的公有成员函数调用基类的公有成员函数，以引用基类的私有数据成员

要正确地引用私有基类的私有成员，作如下改写：

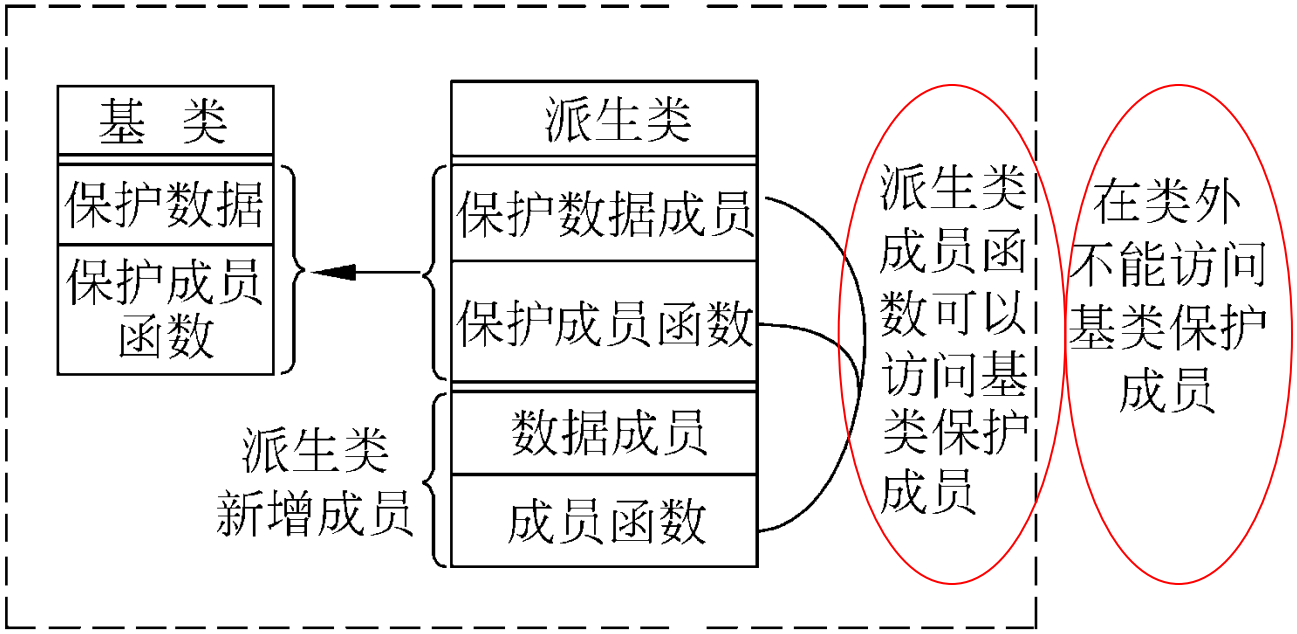
```
void display_1( )//派生类的公有成员函数
{display( );  //调用基类的公用成员函数，引用基类的私有成员
 cout<<"age: "<<age<<endl; //输出派生类的私有数据成员
cout<<"address: "<<addr<<endl;}  //输出派生类的私有成员

int main( )
{Student1 stud1;
 stud1.display_1( );//派生类Student1类的公用函数，可以调用
 return 0;
}
```

注意：由于私有派生类限制太多，使用不方便，一般不常使用。

5.4.3 保护成员和保护继承

由protected声明的成员称“保护成员”。

如果基类声明了私有成员，那么任何派生类都是不能访问它们的，若希望在派生类中能访问它们，应当把它们声明为保护成员。



继承方式指定为protected的，称为保护继承。

用保护继承方式建立的派生类称为保护派生类，其基类称为保护基类。

也就是把基类的公用成员也保护起来，不让类外任意访问。

保护继承的特点是: 保护基类的公用成员和保护成员在派生类中都成了保护成员，其私有成员仍为基类私有。

（对照P166表5.3，增加了保护继承，并综合了表5.1和5.2的内容）

从表5.3可知: 基类的私有成员被派生类继承后（无论是哪种继承方式）变为不可访问的成员，派生类中的一切成员均无法访问它们。

如果需要在派生类中引用基类的某些成员，应当将基类的这些成员声明为**protected**，而不要声明为**private**。

如果善于利用保护成员，既可实现某些成员的隐蔽（信息隐蔽），又可方便地继承（数据共享），实现代码重用。

例5.3 在派生类中引用保护成员。

```
#include <iostream>
#include <string>
using namespace std;
class Student//声明基类
{public://基类公用成员
    void display( );//成员函数声明
 protected : //基类保护成员
   int num;
   string name;
   char sex;
};
void Student::display( ) //定义基类成员函数
  {cout<<"num: "<<num<<endl;
   cout<<"name: "<<name<<endl;
   cout<<"sex: "<<sex<<endl;
  }
class Student1: protected Student //用保护继承声明派生类
{public:
   void display1( );                    //派生类公用成员函数声明
```

```
private:
    int age; //派生类私有数据成员
    string addr;
};
void Student1::display1( ) //定义派生类公用成员函数
    {cout<<"num: "<<num<<endl; //可引用基类的保护成员
     cout<<"name: "<<name<<endl; //可引用基类的保护成员
    cout<<"sex: "<<sex<<endl; //可引用基类的保护成员
     cout<<"age: "<<age<<endl; //引用派生类的私有成员
     cout<<"address: "<<addr<<endl; //引用派生类的私有成员
    }

int main( )
 {Student1 stud1; //stud1是派生类Student1类的对象
  stud1.display1( ); //display1是派生类中的公用成员函数
  stud1.num=10023; //错误，外界不能访问基类的保护成员
return 0;
 }
```

可以通过派生类的公用成员函数display1去访问基类的保护成员num.name和sex

不可以通过派生类的对象直接去访问基类的保护成员

小提示：

1、无论哪一种继承方式，在派生类中是不能访问基类的私有成员的，私有成员只能被本类的成员函数所访问。

2、如果在多级派生时都采用公用继承方式（A类是B类的公有基类，而B类是C类的公有基类，……），那么直到最后一级派生类都能访问基类的公用成员和保护成员。

在实际中，常用的是公用继承。

5.5 派生类的构造函数和析构函数

构造函数的主要作用是对数据成员初始化。

在设计派生类的构造函数时，要使派生类的数据成员和基类的数据成员同时都被初始化。

5.5.1 简单的派生类的构造函数

例5.5 简单的派生类的构造函数。

```
#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)
 //派生类构造函数首行的写法
{    age=a; //新增的数据成员初始化
     addr=ad;
}
   void show( )
    {cout<<"num: "<<num<<endl;//基类的保护成员在派生类中可以引用
     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;
 };
```

总参数列表，依据**参数名称**传递给基类构造函数

基类构造函数参数列表

```
int main( )
 {Student1 stud1(    10010,"Wang-li",'f',19,"115 Beijing Road,Shanghai"    );    //实参
  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;
}
```

若派生类构造函数在类外面定义，在类体中只写该函数的声明:

Student1(int n，string nam，char s，int a，string ad);

在类的外面定义派生类构造函数:

Student1∷Student1(int n，string nam,char s,int a，string ad):Student(n，nam，s) //注意红色部分在函数声明时不写

```
 {age=a;  addr=ad;    }
```

上例中，

1、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。

2、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( )。

5.5.2 有子对象的派生类的构造函数

类的数据成员中还可以包含类对象，例如，在声明一个类时包含这样的数据成员：

Student s1;// Student是已声明的类名，s1是Student类的对象

s1就是类对象中的内嵌对象，称为子对象(subobject)。

例5.6 包含子对象的派生类的构造函数。

```
#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，与例5.5相同
    {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( )//输出学生数据成员
   {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，输出子对象数据成员
   {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(班长)
   int age;
   string addr;
 };
```

基类构造函数参数列表

子对象构造函数参数列表

总参数列表，依据**参数名称**传递给基类和子对象

```
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;
}
```

系统自动执行派生类构造函数的顺序是:

① 调用基类构造函数，对基类数据成员初始化；

② 调用子对象构造函数，对子对象数据成员初始化；

③ 再执行派生类构造函数本身，对派生类数据成员初始化。

5.5.3 派生类的析构函数

调用的顺序与构造函数正好相反：

①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；

②然后调用子对象的析构函数，对子对象进行清理；

③最后调用基类的析构函数，对基类进行清理。

5.4 多重继承

单继承：一个类是从一个基类派生而来的。

多重继承(multiple inheritance)：派生类从两个或多个基类中继承所需的属性。C++允许一个派生类同时继承多个基类。

5.4.1 声明多重继承的方法

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}
```

D是多重继承的派生类，它以公用继承方式继承A类，以私有继承方式继承B类，以保护继承方式继承C类。D按不同的继承方式的规则继承A,B,C的属性，确定各基类的成员在派生类中的访问权限。

5.4.2 多重继承派生类的构造函数

定义形式：

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)

{派生类中新增数据成员初始化语句}

各基类的排列顺序任意。

例5.8　声明一个教师(Teacher)类和一个学生(Student)类，用多重继承的方式声明一个研究生(Graduate)派生类。教师类中包括数据成员name(姓名)、age(年龄)、title(职称)。学生类中包括数据成员name1(姓名)、age(性别)、score(成绩)。在定义派生类对象时给出初始化的数据，然后输出这些数据。

```
#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;                       //职称
};
class Student  //定义类Student(学生)
 {public:
   Student(char nam[],char s,float sco) //构造函数
     {strcpy(name1,nam);
      sex=s;
      score=sco;}
    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;                           //成绩
 };
```

```
class Graduate:    public Teacher, public Student
//声明多重继承的派生类Graduate
 {public://多重继承的派生类构造函数
   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;    //Teacher类保护成员
     cout<<"age:"<<age<<endl; //Teacher类保护成员
     cout<<"sex:"<<sex<<endl;  //Student类保护成员
     cout<<"score:"<<score<<endl; //Student类保护成员
     cout<<"title:"<<title<<endl; //Teacher类保护成员
     cout<<"wages:"<<wage<<endl;//Graduate类私有成员
     }
  private:
    float wage;                    //工资
 };
int main( )
 {Graduate grad1("Wang-li",24,'f',"assistant",89.5,1234.5);
 grad1.show( );
  return 0;
}
```

注意：

在两个基类中，分别用name和name1来代表姓名，其实这是同一个人的名字。

也可在两个基类中都使用同一个数据成员名name，而在show函数中引用数据成员时则应指明其作用域，如

```
cout<<"name:"<<Teacher::name<<endl;
```

不致引起二义性，能通过编译，正常运行

在设计派生类时要细致考虑其数据成员，尽量减少数据冗余。

5.5 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，使软件重用成为可能。

许多厂商开发各类实用的类库。用户将它们作为基类去建立适合于自己的类(即派生类)，并在此基础上设计自己的应用程序。

对类库中类的声明一般放在头文件中，类的实现(函数的定义部分)是单独编译的，以目标代码形式存放在系统某一目录下。用户使用类库时，不需要了解源代码，但必须知道头文件的使用方法和怎样去连接这些目标代码(在哪个子目录下)，以便源程序在编译后与之连接。

作业

P202第9题