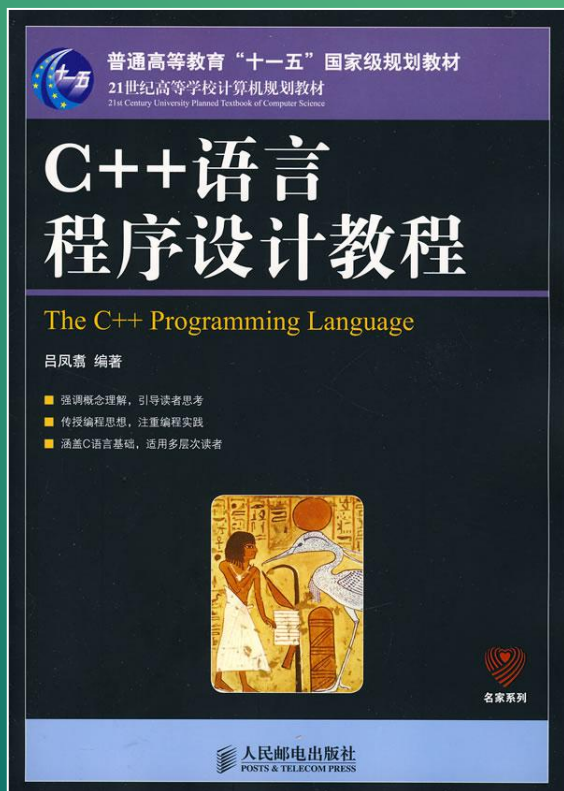


■ The C++ Programming Language



面向对象的程序设计

C++ 程序设计

第8讲 继承性和派生类





- 1 继承的概念
- 2 单重继承
- 3 多重继承



§ 8.1 继承的概念



THE C++ PROGRAMMING LANGUAGE

```
class animal
{
public:
    void eat()
    {
        cout<<"animal eat"<<endl;
    }
    void sleep()
    {
        cout<<"animal sleep"<<endl;
    }
    void breathe()
    {
        cout<<"animal breathe"<<endl;
    }
};
```





```
class fish
{
public:
    void eat()
    {
        cout<<"fish eat"<<endl;
    }
    void sleep()
    {
        cout<<"fish sleep"<<endl;
    }
    void breathe()
    {
        cout<<"fish breathe"<<endl;
    }
};
```





```
#include <iostream.h>
```

```
class animal
```

```
{
```

```
public:
```

```
    void eat()
```

```
    {
```

```
        cout<<"animal eat"<<endl;
```

```
    }
```

```
    void sleep()
```

```
    {
```

```
        cout<<"animal sleep"<<endl;
```

```
    }
```

```
    void breathe()
```

```
    {
```

```
        cout<<"animal breathe"<<endl;
```

```
    }
```

```
};
```

```
class fish:public animal
```

```
{
```

```
};
```

```
void main()
```

```
{
```

```
    animal an;
```


```
    fish fh;
```

```
    an.eat();
```

```
    fh.eat();
```

```
}
```





```
#include <iostream.h>
class animal
{
public:
    animal()
    {
        cout<<"animal construct"<<endl;
    }
    ~animal()
    {
        cout<<"animal destruct"<<endl;
    }
    void eat()
    {
        cout<<"animal eat"<<endl;
    }
    void sleep()
    {
        cout<<"animal sleep"<<endl;
    }
    void breathe()
    {
        cout<<"animal breathe"<<endl;
    }
};
```





```
class fish:public animal
```

```
{
```

```
public:
```

```
    fish()
```

```
    {
```

```
        cout<<"fish construct"<<endl;
```

```
    }
```

```
    ~fish()
```

```
    {
```

```
        cout<<"fish destruct"<<endl;
```

```
    }
```

```
};
```

```
void main()
```

```
{
```

```
    fish fh;
```

```
}
```

animal construct

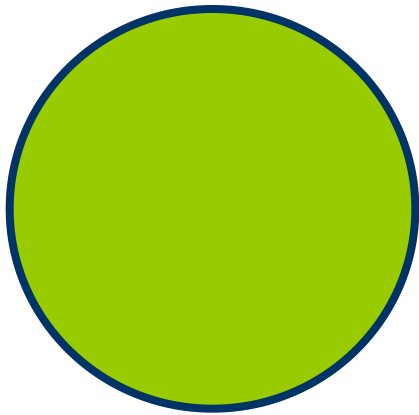
fish construct

fish destruct

animal destruct



为以下图形分别建立类



Circle



Rectangle

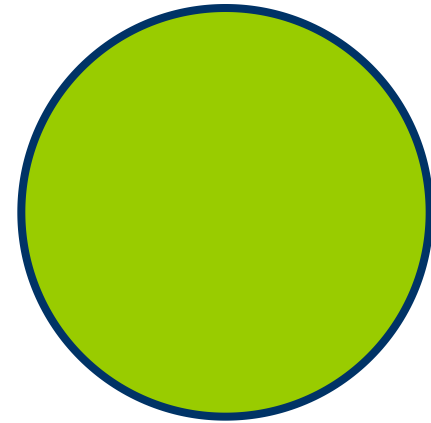


Square





```
class Circle  
{  int fillColor;  
    int borderColor;  
    double radius;  
public:  
    Circle();  
    double Area();  
    double Draw();  
    void SetFillCol(int type);  
    void GetFillCol() ;  
    void SetborderCol(int type);  
    void GetborderCol() ;  
    double GetRadius();  
};
```



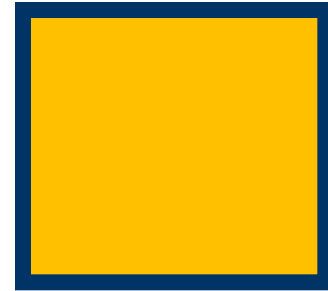


```
class Rectangle  
{  int fillColor;  
    int borderColor;  
    double width;  
    double height;  
public:  
    Rectangle();  
    double Area();  
    double Draw();  
    void SetFillCol(int type);  
    void GetFillCol() ;  
    void SetborderCol(int type);  
    void GetborderCol() ;  
    double GetWidth();  
    double GetHight();  
};
```

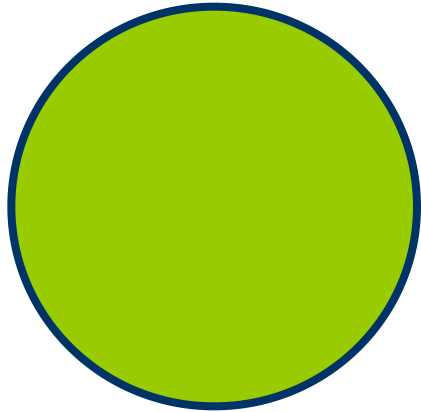




```
class Square  
{  int fillColor;  
    int borderColor;  
    double length;  
public:  
    Square();  
    double Area();  
    double Draw();  
    void SetFillCol(int type);  
    void GetFillCol() ;  
    void SetFillCol(int type);  
    void GetFillCol() ;  
    double GetLength();  
};
```



继承与派生举例



```
class Shape
```

```
{
```

```
int fillColor;
```

```
int borderColor;
```

```
public:
```

```
Shape();
```

```
double Area();
```

```
double Draw();
```

```
void SetFillCol(int type);
```

```
void GetFillCol() ;
```

```
void Set borderColor(int type);
```

```
void Get borderColor() ;
```

```
};
```



```
class Circle: public Shape
```

```
{
```

```
    double radius; // 派生类自己的数据成员
```

```
public:
```

```
    Circle(); // 派生类构造函数
```

```
    double GetRadius(); // 派生类自己的函数成员
```

```
};
```

- 代码减少

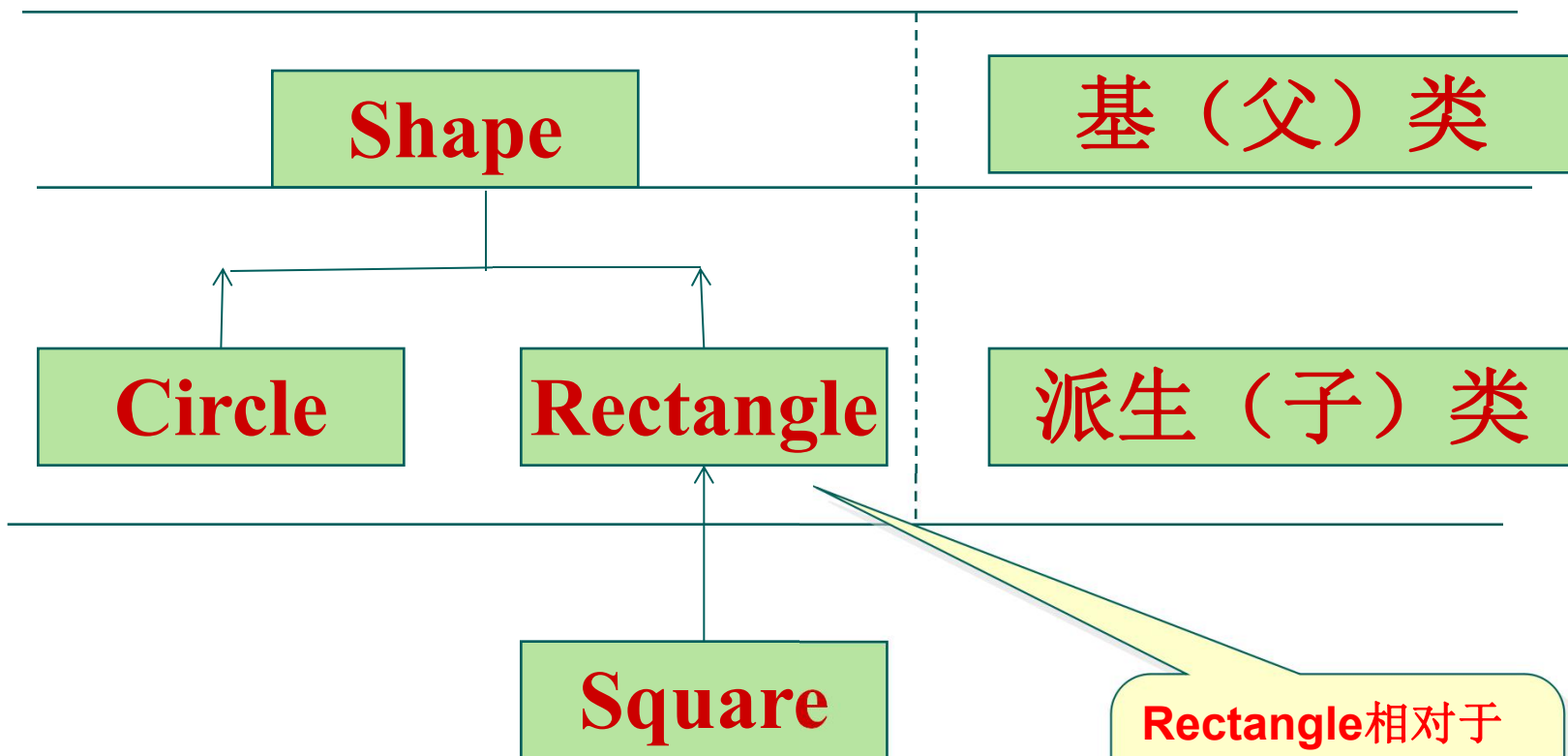




继承与派生问题举例



类的继承与派生



Rectangle相对于
Shape是子类，相
对于**Square**是父类





- ❖ 继承：新类从已有类处得到已有特性的过程
- ❖ 派生：从已有类产生新类的过程
 - 被继承的已有类称为基类（或父类）
 - 派生出的新类称为派生类
- ❖ 继承过程中可以在保持原有特性的基础上，加入自己的新特性。





❖ 继承的目的：代码重用

❖ 派生的目的：代码扩充

当新的问题出现，原有程序无法解决（或不能完全解决）时，需要对原有程序进行改造。





1. 吸收基类成员（除构造和析构函数外）
2. 添加新的成员（重载和新功能的添加）





class Circle: public Shape

```
{
```

```
};
```

class Shape

```
{
```

```
int fillColor;
```

```
int borderColor;
```

```
public:
```

```
Shape(); //不继承
```

```
double Area(){};
```

```
double Draw(){};
```

```
void SetFillCol(int type);
```

```
void GetFillCol();
```

```
void SetborderCol(int type);
```

```
void GetborderCol() ;
```

```
};
```

❖ // 吸收基类成员

❖ // 除构造函数和析构函数



❖ 不同继承方式的影响主要体现在：

- 派生类成员对基类成员的访问权限
- 通过派生类对象对基类成员的访问权限

❖ 三种继承方式

- 公有继承
- 私有继承
- 保护继承

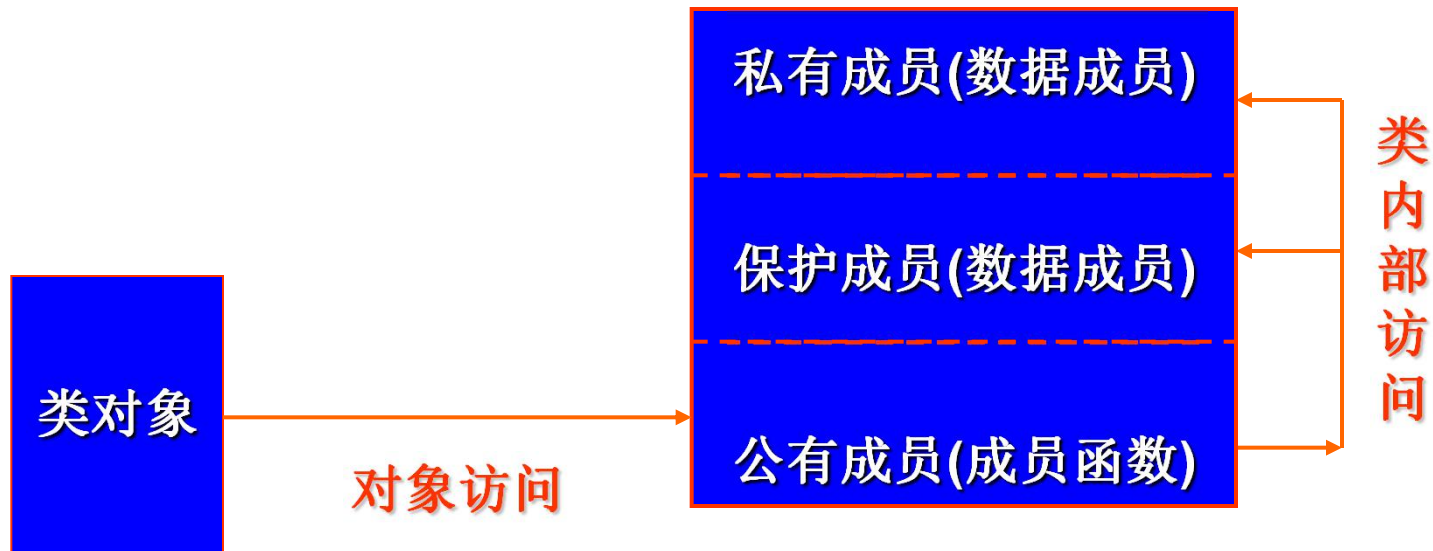
```
class 派生类名: 继承方式 基类名
{
    成员声明;
};
```

// 继承方式: public, protected, private





类体



- 类中的成员函数可以直接访问类中的数据成员（包括私有成员、公有成员、保护成员）；
- 类中的成员函数可以相互访问；
- 类的对象只能访问类的公有成员，不能访问私有成员及保护成员



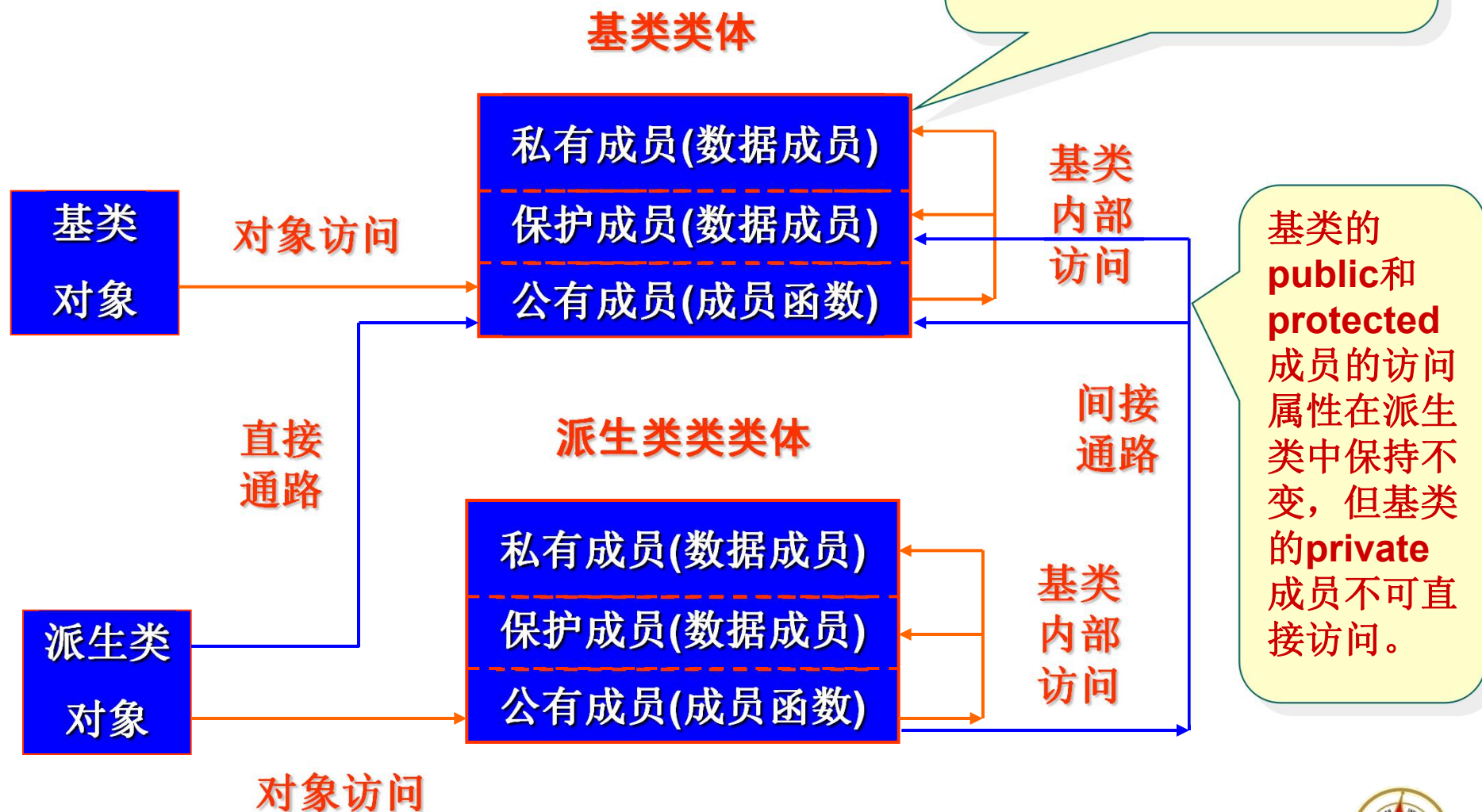


```
class A
{
    protected:
        int x;
}
int main()
{
    A a;
    a.x=5; //错误
}
```



公有继承 (public)

派生类的对象可以通过调用基类的公有成员访问基类的私有成员



```
class A
```

```
{ public: int x; };
```

```
class B: public A
```

```
{
```

```
public:
```

```
void Function()
```

```
{
```

```
    x=5;    // 正确
```

```
}
```

```
}; // 公有成员的公有继承
```

```
int main( )
```

```
{
```

```
    A a;
```

```
    a.x=6;    // 正确
```

```
    B b;
```

```
    b.x=10; // 正确
```

```
    return 0;
```

```
}
```

```
class A
```

```
{ protected: int x; };
```

```
class B: public A
```

```
{
```

```
public:
```

```
void Function()
```

```
{
```

```
    x=5; //正确
```

```
}
```

```
}; // 保护成员的公有继承
```

```
int main( )
```

```
{
```

```
    A a;
```

```
    a.x=6; //错误
```

```
    B b;
```

```
    b.x=10; //错误
```

```
    return 0;
```

```
}
```



```
class A
```

```
{ private: int x; };
```

```
class B: public A
```

```
{
```

```
public:
```

```
void Function()
```

```
{
```

```
    x=5; //错误
```

```
}
```

```
}; //私有成员的公有继承
```

```
int main( )
```

```
{
```

```
    A a;
```

```
    a.x=6; //错误
```

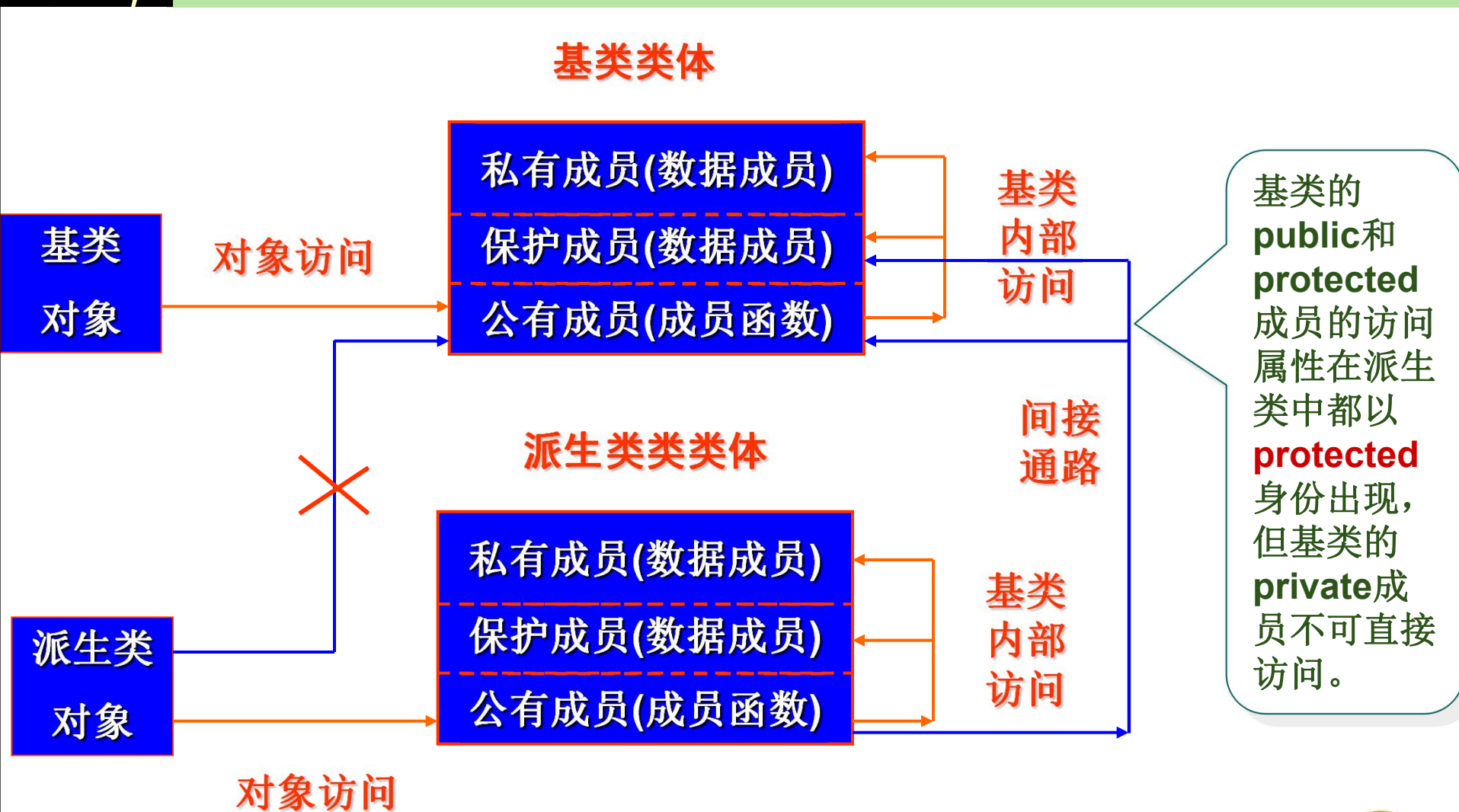
```
    B b;
```

```
    b.x=10; //错误
```

```
    return 0;
```

```
}
```

保护继承 (protected)



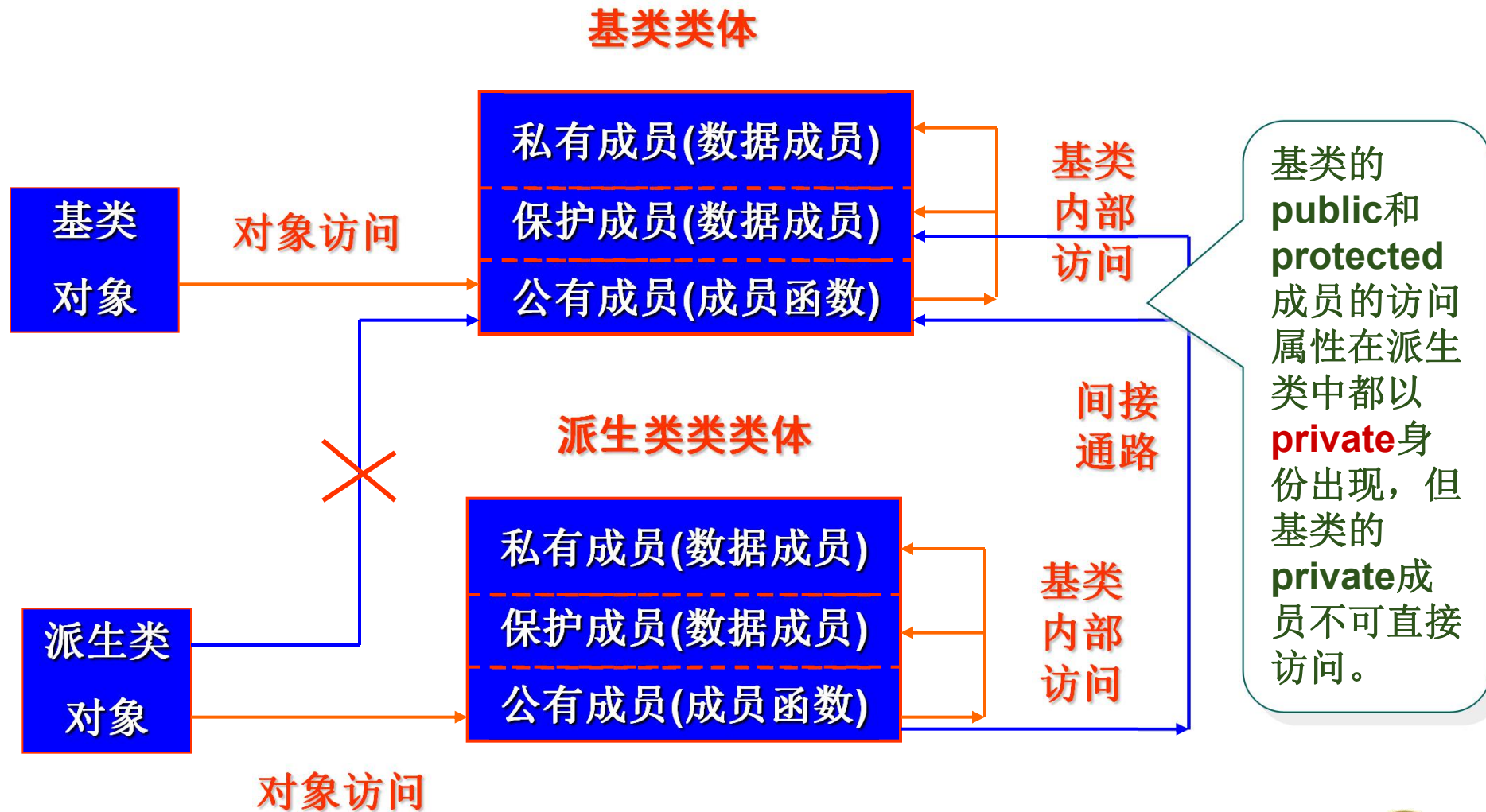
公有成员的保护继承（判断）

```
class A
{ public: int x; };

class B: protected A
{
public:
void Function()
{
    x=5; // 正确
}
};
```

```
int main( )
{
    A a;
    a.x=6; // 正确
    B b;
    b.x=10; // 错误
    return 0;
}
```

私有继承 (private)



派生类成员的访问权限



公有成员可以被外部访问，也可以被继承；
私有成员不能被外部访问，也不能被继承；
保护成员不能被外部访问，但可以被继承。



公司外部访，也可被继承
私有不能访，也不能继承
保护不能访，却可被继承



第8章 继承性和派生类



THE C++ PROGRAMMING LANGUAGE

1

继承的概念

2

单重继承

3

多重继承





❖ 子对象

- 在一个类中可以使用另一个类的对象作其数据成员，这种对象的数据成员称为子对象。子对象反映两个类之间的包含关系。

```
class A  
{ ... };
```

```
class B  
{ ...  
private:  
    A a;  
    ...  
};
```

子对象





- ❖ **【例7.9】** 分析下列程序的输出结果，熟悉子对象在程序中的应用。

```
#include <iostream.h>
class B
{
public:
    B(int i,int j)    {  b1=i;  b2=j;  }
    void Print()      {  cout<<b1<<', '<<b2<<endl;  }
private:
    int b1,b2;
};
class A
{
public:
    A(int i,int j,int k) : b(i,j)
    {  a = k;  }
    void Print()
    {  b.Print();  cout<<a<<endl;  }
private:
    B b;
    int a;
};
```

调用子对象的成员

b为A的子对象



组合：类以另一个类作为数据成员，称为组合。

继承和组合都有代码重用的作用。



Class Vehicle

```
{  
    //.....  
};  
Class Motor  
{  
    //.....  
};  
Class Car:public Vehicle  
{  
    public:  
        Motor motor;  
};  
Void vehicleFn(Vehicle& v);  
Void motorFn(Motor& m);
```

Void main()

```
{  
    Car c;  
    vehicleFn(c);    //ok  
    motorFn( c);    //error  
    motorFn(c.motor); //ok  
}
```





- ❖ 派生类的构造函数
- ❖ 派生类的构造函数应该包含它的直接基类的构造函数。
 - 先执行基类构造函数；
 - 再执行子对象的构造函数（如有子对象的话）；
 - 最后执行派生类构造函数的函数体。

```
class A // C的基类
{ public: A(); };
class B
{ public: B(); };
class C : public A // A的派生类
{
public: C();
private: B b; // B为C的子对象
};
```

C c;
当用C构建对象c时，ABC各类的构造函数的调用顺序是：
A() → B() → C()



§ 8.2 单重继承



THE C++ PROGRAMMING LANGUAGE

- ❖ 派生类的构造函数
- ❖ 派生类构造函数的成员初始化列表中应该：
 - 显式地包含基类中带参数的构造函数，
 - 或者隐含地包含基类中的默认构造函数。



```
class A // B的基类
{ public: A(int a); };
class B : public A // A的派生类
{
public:
    B(int n, int m) : A(n) { b=m; }
private: int b;
};
```





❖ 派生类的析构函数

❖ 由于析构函数不能继承，因此在派生类的析构函数中要包含它的**直接基类**的析构函数。

- 先执行**派生类**析构函数的函数体。
- 再执行**子对象**所在类的析构函数（如果有子对象的话）
- 最后执行直接**基类**中的析构函数。

```
class A // C的基类
{ public: ~A(); };
class B
{ public: ~B(); };
class C : public A // A的派生类
{
public:   ~C();
private: B b; // B为C的子对象
};
```

C c;

当对象c析构时，ABC各类的析构函数的调用顺序是：

$\sim C() \rightarrow \sim B() \rightarrow \sim A()$



§ 8.2 单重继承



THE C++ PROGRAMMING LANGUAGE

❖ **【例8.7】** 分析下列程序的输出结果，熟悉多层派生类构造函数和析构函数。

```
#include <iostream.h>

class A
{public:
    A()    { a=0; }
    A(int i) { a=i; }
    ~A() { cout<< "In A.\n"; }
    void Print() { cout<<a<<','<<','<<'\n'; }
private:
    int a;
};

class B : public A
{public:
    B() { b1=b2=0; }
    B(int i) { b1=0;b2=i; }
    B(int i,int j,int k) : A(i),b1(j),b2(k) { }
    ~B() { cout<<"In B.\n"; }
    void Print() { A::Print(); cout<<b1<<','<<b2<<','<<'\n'; }
private:
    int b1,b2;
};
```



§ 8.2 单重继承



THE C++ PROGRAMMING LANGUAGE

```
class C : public B
{
public:
    C() { c = 0; }
    C(int i) { c = i; }
    C(int i, int j, int k, int l)
        : B(i, j, k), c(l) { }
    ~C() { cout<<"In C.\n"; }
    void Print()
    { B::Print(); cout<<c<<endl; }
private:
    int c;
};

void main()
{
    C c1;      a=0,b1=0,b2=0,c=0
    C c2(10);  a=0,b1=0,b2=0,c=10
    C c3(10,20,30,40); a=10,b1=20,b2=30,c=40
    c1.Print();
    c2.Print();
    c3.Print();
}
```

```
0,0,0,0
0,0,0,10
10,20,30,40
In C. } c3的析构
In B. }
In A. }
In C. } c2的析构
In B. }
In A. }
In C. } c1的析构
In B. }
In A. }
```



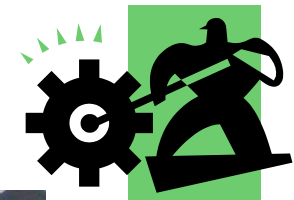
§ 8.2 单重继承



THE C++ PROGRAMMING LANGUAGE

❖ 【例8.5】分析下列程序的输出结果，熟悉派生类构造函数的定义格式。

```
#include <iostream.h>
#include <string.h>
class Student
{
    char name[20];
    int stuno;
    char sex;
public:
    Student(int no, char *str, char s)
    {
        stuno = no;
        strcpy(name, str);
        sex=s;
        cout<<"Constructor called.S\n";
    }
    void Print()
    { cout<<stuno<<'\\t'<<name<<'\\t'<<sex<<'\\t'; }
};
```



§ 8.2 单重继承



THE C++ PROGRAMMING LANGUAGE

```
class Student1:public Student
{
public:
    Student1(int no,
             char *name, char sex,
             int age, int score)
    {
        age=a;
        score=sco;
        cout<<"Constructor called.S1\n";
    }
    void Print()
    {
        Student::Print();
        cout<<age<<"\t"<<score<<endl;
    }
private:
    int age,score;
};
```

调用基类的构造函数

```
void main()
{
    Student1 s1(502001,"Ma Li",'m',20,90),
               s2(502002,"Li Hua",'f',19,88);
    s1.Print();
    s2.Print();
}
```

```
construct called.S
construct called.S1
construct called.S
construct called.S1
502001 Ma Li m 20 90
502002 Li Hua f 19 88
```

同名覆盖，如果没有Student: :，派
生类对象将调用自身的Print()函数



§ 8.2 单重继承



THE C++ PROGRAMMING LANGUAGE

❖ **【例8.6】** 分析下列程序的输出结果，说明派生类构造函数的执行顺序。

```
#include <iostream.h>
class A
{
public:
    A()
    { a=0; cout<< "A Default constructor called." << a << endl; }
    A(int i)
    { a=i;  cout<< "A Constructor called."<<a<<endl;    }
    ~A() { cout<< "A Destructor called."<<a<<endl;  }
    void Print()
    {  cout<<a<<', ' ;  }
    int Geta()    {  return a;  }
private:
    int a;
};
```



§ 8.2 单重继承



THE C++ PROGRAMMING LANGUAGE

```
class B : public A
{
public:
    B()
    { b=0; cout<< "B Default constructor called.\n"; }
    B(int i,int j,int k) : A(i),aa(j)
    { b=k; cout<< "B Constructor called.\n"; }
    ~B() { cout<< "B Destructor called.\n"; }
    void Print()
    { A::Print(); cout<<b<<','<<aa.Geta(); }
private:
    int b;
    A aa;
};

void main()
{
    B bb[2];
    bb[0]=B(7,8,9);
    bb[1]=B(12,13,14);
    for(int i=0;i<2;i++)
        bb[i].Print();
}
```

```
A Default constructor called.0
A Default constructor called.0
B Default constructor called.0
A Default constructor called.0
A Default constructor called.0
B Default constructor called.0
A constructor called.7
A constructor called.8
B constructor called.9
B Destructor called.9
A Destructor called.8
A Destructor called.7
```

注意：无名对象生命期只在表达式中。





关于类的继承，下面哪种说法是正确的？

(**D**)

- A.** 在构建派生类的对象时会先调用基类的构造函数，析构时也是如此。
- B.** 如果派生类有子对象，子对象的构造函数比基类的构造函数优先调用。
- C.** 无论何种继承，在派生类对象中总可以直接访问基类的保护成员。
- D.** 派生类不能访问基类的私有成员，但可通过调用基类有关函数方式间接访问。



定义如下：

```
#include <iostream.h>
class Base
{
    public:
        Base(int x=0) { b=x+3; cout << b++;}
        ~Base() { cout << b;}
    private:
        int b;
};
class Derived : public Base
{
    public:
        Derived(int x=0,int y=0):Base(x)
        { d = y-1; cout << d--;}
        ~Derived() { cout << d; }
    private:
        int d;
};
void main()
{
    Derived obj(2,7);
}
```

运行后的输出结果是：

()



THE C++ PROGRAMMING LANGUAGE





❖ 子类型

- ❖ 当一个类型至少包含了另一个类型的所有行为，则称该类型是另一个类型的子类型。
 - 例如，在公有继承下，派生类是基类的子类型。
 - 如果类型A是类型B的子类型，则称类型A适应于类型B，这时用类型B对象的操作也可以用于类型A的对象。因此，可以说类型A的对象就是类型B的对象。
 - 子类型的关系是不可逆的。



赋值兼容规则



当类型A是类型B的子类时，则满足下述的赋值兼容规则。**A是B的派生类，B是A的基类。**

① A类的对象可以赋值给B类的对象。**派生类的对象可以赋值给基类类型的对象。**

② A类的对象可以给B类对象引用赋值。**派生类对象可以给基类的对象引用赋值。**

③ A类的对象地址值可以给B类对象指针赋值。**派生类对象指针可以给基类对象指针赋值，即基类对象指针可以指向派生类的对象。**

例如：

```
class A:public B
A a;B b;A *pa;B *pb;
b = a;
B &rb = a;
pb =&a; pb = pa;
```

例8.8 分析下列程序的输出结果。

```
#include<iostream.h>

class A
{public: A()      {a=0;}
      A(int i)  {a=i;}
      void Print()      {cout<<a<<endl;}
      int Geta()      {return a;}
private: int a;
};

class B:public A
{public: B()      {b=0;}
      B(int i,int j):A(i),b(j) { }
      void Print()
      {
          cout<<b<<',';
          A::Print();
      }
private: int b;
};

void fun(A &a)
{
    cout<<a.Geta()+2<<',';
    a.Print();
}
```

```
void main()
{
    A a1(10),a2;
    B b(10,20);
    b.Print();
    a2=b;
    a2.Print();
    A *pa=new A(15);
    B *pb=new B(15,25);
    pa=pb;
    pa->Print();
    fun(*pb);
    delete pa;
}
```

20, 10
10
15
17,15



例8.8

```
Print() \Geta()  
int a
```

A

a1: a=10

a2: a=0

a=15

pa:

```
A a1(10), a2;  
B b(10, 20);  
a2=b;
```

```
A *pa= new A(15);  
B *pb= new B(15, 25);  
pa=pb;
```

pb:

```
Print() \Geta()  
int a
```

```
Print()  
int b
```

B

b: a=10

b=20

a=15

b=25



- ❖ 试建立一个类**Worker**用于描述职工，具体要求如下：
- ❖ （1）私有数据成员
- ❖ **unsigned int id:** 职工号。
- ❖ **char name[11]:**姓名
- ❖ **float salary:** 工资。（有一符号常量为工资最低值，设为**2000**）
- ❖ **int level:** 技术等级（**1~20**级），**1**级为最低，每个等级差别**200**元
- ❖ （2）公有成员函数
- ❖ **Worker ():** 构造函数，初始化数据成员为默认值（自动生成职工号，姓名为空，工资默认最低）。
- ❖ **Worker (...):** 构造函数(自己定义参数)，用参数初始化数据成员。参数有各个数据成员。
- ❖ **void setName(...):** 设置职工姓名
- ❖ **void infoList():** 输出职工的各项信息。
- ❖ **void setLevel(...):** 修改技术等级**level**的值。每增加一个等级工作增加**200**元。
- ❖ （3）**static**变量**total**计算工资总数和函数**Average()**计算平均工资。
- ❖ （4）设置友元函数**void setReward(...)**，给工人。
- ❖ （5）在主程序中定义**N(=5)**个**Worker**对象作为测试数据，完成对**Worker**类和程序的测试。
- ❖ （6）在主程序中对其中某个工人发放奖金**500**元。





❖ ===== 作业2

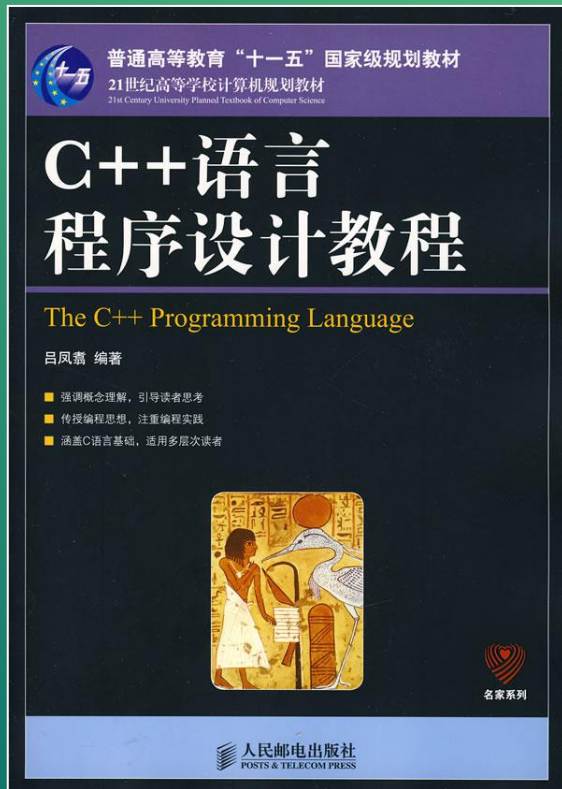
- ❖ 在**Worker**类的基础上声明一个个数为**N(=5)**的对象数组**WorkerArr**,
- ❖ 并设指向该数组的指针**pWorkerArr**, 用该指针为数组输入各成员的数据。
- ❖ 用**for**循环移动该指针, 直到输入完成。

❖ ===== 作业3

- ❖ 在类**Worker**的基础上派生两个类: **PieceWorker**和**HourWorker**
- ❖ 在**PieceWorker**中添加变量**piece** (整数), 自己设定每个**piece**的报酬, 并根据**piece**计算**salary**.
- ❖ 在**HourWorker**中添加变量**hour** (浮点数), 自己设定每个**hour**的报酬, 并根据**hour**计算**salary**.
- ❖ 在**main()**中分别声明不同的对象, 并计算他们各自的**salary**.



■ The C++ Programming Language



Thank You!

