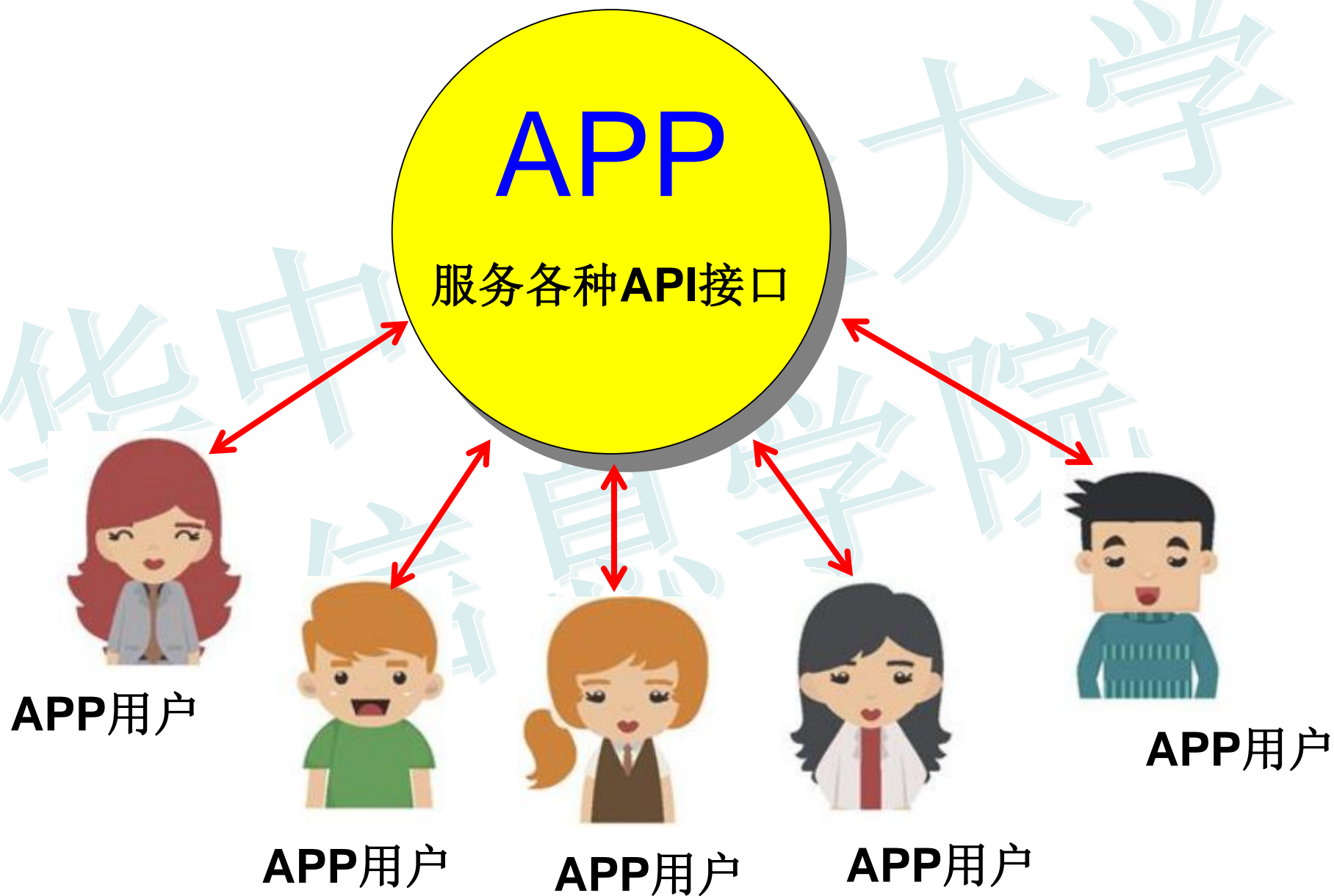


C/C++程序设计案例实战

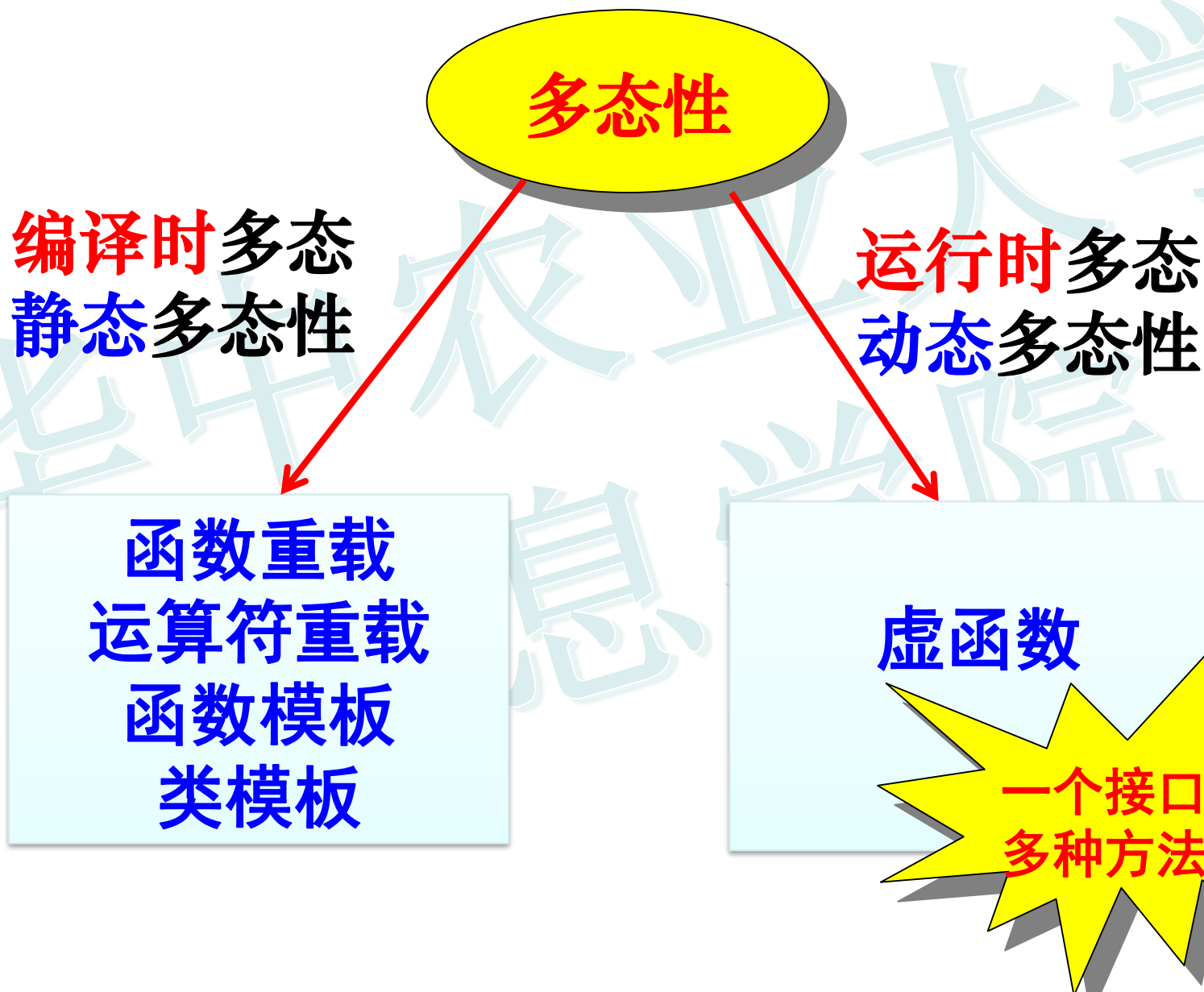
—— 一个接口多种方法

华中农业大学信息学院 章 英

问题引入



问题引入



虚函数

- 用**virtual**关键字说明的成员函数
- 虚函数是实现**运行时多态性**基础
- C++中的虚函数是**动态绑定**的函数
- 虚函数必须是非静态的成员函数，虚函数经过**派生**之后，就可以实现运行过程中的多态。

案例分析

基类：点类

(x, y)



派生类：矩形类

(x, y)

w

h



案例——代码实现

Point类——基类

```
class Point
{ public:
    Point(double i, double j)
        {x=i; y=j;}
    double Area() const
        { return 0.0;}
private:
    double x, y;
};
```

案例——代码实现

Rectangle类——派生类

```
class Rectangle:public Point
{ public:
    Rectangle(double i, double j,
double k, double l);
    double Area() const
        {return w*h;}
private:
    double w, h;
};


Rectangle::Rectangle(double i, double
j, double k, double l) :Point(i,j)
{ w=k; h=l; }
```

案例——代码实现

静态多态性

形参：基类对象的引用

```
void fun(Point &s)  
{  
    cout<<"Area="<<s.Area()<<endl; }  
  
int main()  
{  
    Rectangle rec(3.0, 5.2, 15.0, 25.0);  
    fun(rec); 实参：派生类对象  
    return 0;  
}
```

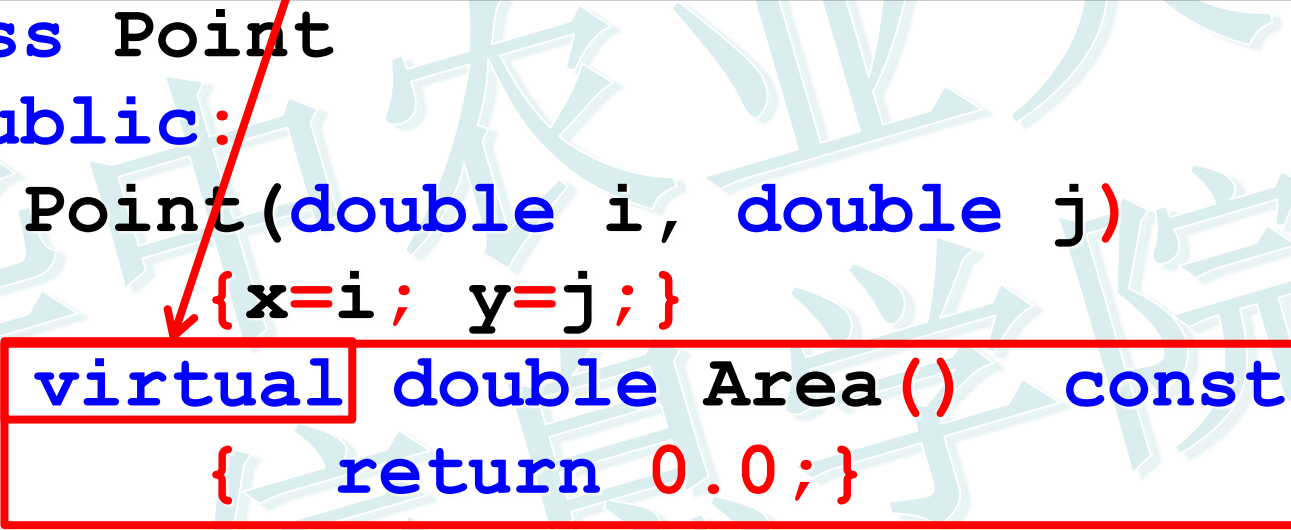


运行结果：Area=0

进阶案例——代码实现

virtual 虚函数

```
class Point
{ public:
    Point(double i, double j)
        {x=i; y=j;}
    virtual double Area() const
        { return 0.0;}
private:
    double x, y;
};
```



进阶案例——代码实现

virtual 虚函数

```
class Rectangle:public Point
{ public:
    Rectangle(double i, double j,
double k, double l);
    virtual double Area() const
    {return w*h;}
private:
    double w, h;
};

Rectangle::Rectangle(double i, double
j, double k, double l) :Point(i,j)
{ w=k; h=l; }
```


进阶案例——代码实现

动态多态性

形参：基类对象的引用

```
void fun(Point &s)
{
    cout<<"Area="<<s.Area()<<endl; }

int main()
{
    Rectangle rec(3.0,5.2,15.0,25.0);
    fun(rec); 实参：派生类对象
    return 0;
}
```



运行结果：Area=375

虚函数

- C++中引入了虚函数的机制在派生类中可以对基类中的成员函数进行覆盖（重定义）。
- 虚函数的声明形式

virtual 函数类型 函数名（形参表）

{

函数体

}

类

数据成员

构造函数

成员函数1

成员函数2

...

成员函数n

析构函数

virtual

virtual

虚析构函数

- 为什么需要虚析构函数？
 - 可能通过基类指针删除派生类对象；
 - 如果打算允许其他人通过基类指针调用对象的析构函数（通过delete这样做是正常的），就需要让基类的析构函数成为虚函数，否则执行delete的结果是不确定的。

案例——代码实现

运行结果: Base destructor

```
4  class Base {
5      public:
6          ~Base()
7          {   cout<< "Base destructor" << endl; }
8      };
9
10 class Derived: public Base{
11     public:
12         Derived(){ p = new int(0); }
13         ~Derived()
14         {
15             cout << "Derived destructor" << endl;
16             delete p;
17         }
18     private:
19         int *p;
20 };
21
22 void fun(Base *b) { delete b;}
23
24 int main() {
25     Base *b = new Derived();
26     fun(b);
27     return 0;
28 }
```

基类指针

b

派生类对象

数据成员p

整形空间



进阶案例——代码实现

```
4  class Base {  
5      public:  
6          virtual ~Base()  
7              { cout<< "Base destructor" << endl; }  
8  };  
9  
10 class Derived: public Base{  
11     public:  
12         Derived(){ p = new int(0); }  
13         ~Derived()  
14         {  
15             cout << "Derived destructor" << endl;  
16             delete p;  
17         }  
18     private:  
19         int *p;  
20 };  
21  
22 void fun(Base *b) { delete b; }  
23  
24 int main() {  
25     Base *b = new Derived();  
26     fun(b);  
27     return 0;  
28 }
```

运行结果: Derived destructor
Base destructor



比一比



想一想



试一试

问题1：普通成员函数的虚函数与虚析构函数有何不同？

问题2：构造函数能不能是虚函数？为什么？

问题3：静态成员函数能否作为虚函数？

比一比

编译时多态和运行时多态的区别

名称	判断标准	完成时间	类型	实现方式
编译时多态	1、指针类型 2、引用类型	编译时	1、函数重载 2、运算符重载 3、函数模板 4、类模板	具有相同的隐式接口
运行时多态	指针指向的对象类型	运行时	虚函数	1、有基类定义虚函数 2、继承 3、基类指针指向子类对象，直接或间接使用基类指针调用虚函数

虚函数——纯虚函数

- 纯虚函数是一个在基类中声明的虚函数，它在该基类中**没有定义具体的操作内容**，要求各派生类根据实际需要定义自己的版本，纯虚函数的声明格式为：

– virtual 函数类型 函数名(参数表) = 0;

- 带有纯虚函数的类称为抽象类：

```
class 类名  
{
```

```
    virtual 类型 函数名(参数表)=0; //纯虚函数
```

```
    ...
```

```
}
```

抽象类

• 作用

- 抽象类为抽象和设计的目的而声明，将有关的数据和行为组织在一个继承层次结构中，保证派生类具有要求的行为。
- 对于暂时无法实现的函数，可以声明为纯虚函数，留给派生类去实现。

• 注意

- 抽象类只能作为基类来使用。
- 不能声明抽象类的对象。
- 构造函数不能是虚函数，析构函数可以是虚函数。

小结

- (1) 能够编写虚函数实现运行时多态
- (2) 能够说出使用虚析构函数的理由
- (3) 能够写出纯虚函数的定义形式
- (4) 能够区分编译时多态和运行时多态的异同

延伸

请改编案例代码：设计一个抽象点类，然后实现派生的矩形类、圆形类，最后通过一个基类指针求解各个派生类对象的周长和面积。