

## C++ 程序设计

## 5.4 同名的成员函数

1. 重载成员函数：与普通函数的重载类同，成员函数的重载是在**一个类作用域内**，多个函数体可以使用相同的函数名，这类成员函数称为**重载成员函数**。这类成员函数或者参数的个数不同，或者至少有一个参数的类型不同。实际选择调用哪一个成员函数的匹配规则与普通函数类同。

```
class Counter {
public:
    Counter() { v = 0; }
    Counter & operator ++( );
        //Prefix increment operator
    Counter operator ++(int);
        //Suffix increment operator
    void Print( );
private:
    unsigned v;
};
```

， 华中科技大学人工智能与自动化学院

面向对象程序设计

黎云 王卓

## C++ 程序设计

如前所述，几乎每个类都定义了多个构造函数，以适应创建对象时，对象的参数具有不同个数的情况，即定义了**重载构造函数**。

## 2. 基类和派生类的同名成员函数

在**类层次结构**中，基类的成员函数在派生类中重新定义，即在基类和派生类中有同名的成员函数。它们是在**编译时加以区分**，即编译系统确定调用哪一个类的成员函数，有如下区分方法：

， 华中科技大学人工智能与自动化学院

面向对象程序设计

黎云 王卓

## C++ 程序设计

(1) 根据**类对象**加以区分。即使基类和派生类的同名成员函数具有不同参数时，仍然是**基类对象调用基类的成员函数**，**派生类对象调用派生类的成员函数**。

```
#include <iostream>
using namespace std;
class A {
public:
    int foo(int i) { return i + 1; }
};
class B : public A {
public:
    float foo(float f) { return f + 10; }
};
```

， 华中科技大学人工智能与自动化学院

面向对象程序设计

黎云 王卓

## C++ 程序设计

```
void main( )
{ A a; B b;
  cout << "(1) 基类对象调用基类的成员函数
  foo( ), 其结果为 : " << a.foo(5) << endl;
  cout << "(2) 派生类对象调用派生类的成员
  函数foo( ), 其结果为 : " << b.foo(2)
  << endl;
  cout << "(3) 派生类对象调用基类的成员函
  数foo( ), 其结果为 : " << b.A::foo(6)
  << endl;
}
```

该程序的输出结果：

- (1) 基类对象调用基类的成员函数foo( )，其结果为：6
- (2) 派生类对象调用派生类的成员函数foo( )，其结果为：12
- (3) 派生类对象调用基类的成员函数foo( )，其结果为：7

， 华中科技大学人工智能与自动化学院

面向对象程序设计

黎云 王卓

## C++ 程序设计

上例中，由于派生类B中定义的同名成员函数foo(float f)与从基类A中继承的成员函数foo(int i)的作用域不同，虽然它们的形式参数不同，但在类B的作用域中**不发生重载**。仍然是基类对象a调用基类的成员函数A: :foo()，派生类对象b调用派生类的成员函数B: :foo()。

(2) 在派生类的成员函数体内，需要访问基类中的同名成员函数，则必须使用作用域分辨符加以区分，否则将产生**无穷循环**的递归调用，其格式为：

**基类名::成员函数名(参数表);**

例如：

```
void Derived::print()
{
    Base::print();
    ...
}
```

## C++ 程序设计

公有继承方式时，派生类对象访问基类中的同名成员函数，也必须使用作用域分辨符加以区分，其格式为：派生类对象.基类名::成员函数名(参数表)，例如：

**int i=b.A::foo(2);**

(3) 在派生类中重定义的成员函数**掩盖 (override)**了基类中**所有**同名成员函数，即使基类中的同名成员函数的参数表与派生类完全不同，在派生类作用域内基类的同名成员函数也只有用“**基类名::**”的格式指明才能访问到它们。

## C++ 程序设计

3. 基类指针和派生类对象：虽然在类层次结构中，基类指针可用来指向公有派生类的对象，但是当基类和派生类中具有同名的成员函数时，编程者即使让基类指针指向派生类对象，然而由于“**静态联编**”机制的特点，编译系统只能根据基类指针定义时的类型说明，确定**基类类型的指针总是指向基类的(函数)**。

```
#include <iostream>
using namespace std;
class A{
public:
    void Display( ) { puts("Class A"); }
};
class B : public A {
public:
    void Display( ) { puts("Class B"); }
};
void Show(A * p) { p -> Display( ); }
//Show( )的形参是基类的对象指针
```

## C++ 程序设计

```
void main( )
{
    A * pa = new A;
    B * pb = new B;
    pa -> Display( );
    //多接口界面，基类动态对象pa调用A::Display( )
    pb -> Display( );
    //多接口界面，派生类动态对象pb调用B::Display( )。
    Show(pa);
    /* 通过Show( )的单接口界面，用基类动态对象pa作
       实参，调用的是A::Display( )。*/
    Show(pb);
    /* 通过Show( )的单接口界面，用派生类动态对象pb
       作实参，仍然是调用A::Display( )。*/
}
```



## C++ 程序设计

该程序的输出结果:

```
Class A
Class B
Class A
Class A
例程说明:
```

(1) 因此, 如果调用了基类A中的成员函数A::Display() 则在CRT屏幕上出现Class A而调用了派生类B中的成员函数B::Display() 则出现Class B。编程者的意图是在main() 函数内, 第一次调用Show(pa), 实参采用基类A的指针pa, 第二次调用Show(pb), 实参采用派生类B的指针pb, 在CRT屏幕上似乎应该得到Class A Class B的结果, 但其实并非如此。

华中科技大学人工智能与自动化学院 面向对象程序设计 黎云 王卓

## C++ 程序设计

(2) Show() 函数对基类A和派生类B中的两个同名成员函数Display() 的访问都是通过指向基类A的指针\*p实现的。

然而main() 函数内, 第二次调用Show(pb), 在实参传递给形参的过程中, main() 函数内的B类指针pb自动转换成A类指针赋给Show() 函数的形参(A \*p), 因此“Show(b);”语句理所当然地调用A::Display()。由此可知, Show() 函数总只能访问基类A中的成员函数A::Display(), 而无法访问派生类B中的同名成员函数B::Display()。

在屏幕上显示的结果为:

```
Class A
Class A, 而达不到编程者的目的
```

华中科技大学人工智能与自动化学院 面向对象程序设计 黎云 王卓

## C++ 程序设计

这种情况经常出现在编程者设计“单界面, 多实现版本”的程序中, 正如上例中, 设计一个函数Show(), 使用基类指针“A \*p”作为它的形式参数, 形成了一个“单界面”。再利用“一个指向基类的指针可用来指向从基类公有继承的任何对象”这一重要规则, 在基类和派生类各层中, 编写各种不同实现版本的同名成员函数A::Display()、B::Display()、...。

但是由于“静态联编”的弊病, 使得“在基类和派生类中有同名成员函数的情况下, 编译时基类指针总是指向基类成员”而达不到目的。因此要实现“单界面, 多实现版本”的思想, 必须依靠虚函数和“动态联编”机制。

华中科技大学人工智能与自动化学院 面向对象程序设计 黎云 王卓

## C++ 程序设计

(3) 从程序可知, 编程者是想构造一个“单界面, 多实现版本”的框架, 然而, 采用上述“静态联编”的方法, 却不能实现“单界面, 多实现版本”的框架。于是有人可能提出在main() 函数内编写如下语句:

```
pa -> Display() ;//Call A::Display()
pb -> Display() ;//Call B::Display()
```

上面是在main() 函数内定义了两个动态对象pa和pb。对于自动型对象也可编写如下语句:

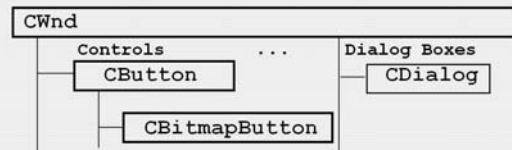
```
void main ()
{
    A a ;
    B b ;
    a.Display() ;//Call A::Display()
    b.Display() ;//Call B::Display()
}
```

华中科技大学人工智能与自动化学院 面向对象程序设计 黎云 王卓

## C++ 程序设计

但是它们都是**多接口界面**。因为A::Display()和B::Display()都是普通的成员函数，则按“静态联编”机制来实现调用，即在编译时按调用成员函数的对象确定好调用哪个类的成员函数。

仍然是基类对象a调用基类的成员函数A::Display()，派生类对象b调用派生类的成员函数B::Display()，但是它**不是单接口界面而是多接口界面**。



15 华中科技大学人工智能与自动化学院 面向对象程序设计 黎云 王卓

## C++ 程序设计

(4) 由于对象引用也是地址传递方式，与对象指针有类似的性质，即基类的对象**引用**可用于从基类公有继承的任何派生类对象”。因此例5.7可改写为：

```

void Show(A & r)
{ r.Display( ); }

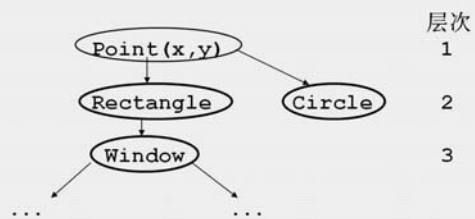
void main( )
{ A a; B b;
  a.Display( ); //use A::Display( )
  b.Display( ); //use B::Display( )
  Show(a); //use A::Display( )
  Show(b); //use A::Display( )
}

```

16 华中科技大学人工智能与自动化学院 面向对象程序设计 黎云 王卓

## C++ 程序设计

又如：



15 华中科技大学人工智能与自动化学院 面向对象程序设计 黎云 王卓

## C++ 程序设计

```

#include <iostream>
using namespace std;
class Point {
public :
    Point(double i, double j)
    { x = i; y = j; }
    double Area( ) const { return 0.0; }
private :
    double x, y;
};
class Rectangle : public Point {
public :
    Rectangle(double i, double j,
               double k, double l);
    double Area(void) const
    { return w * h; }
private :
    double w, h;
};

```

16 华中科技大学人工智能与自动化学院 面向对象程序设计 黎云 王卓

## C++ 程序设计

```

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

void fun(Point & s)          //单接口界面
{    cout << "The Area = "
    << s.Area() << endl;    }

void main( )
{    Rectangle  rec(3.0 , 5.2 ,
    15.0 , 25.0);

    fun(rec);
}
该程序的输出结果:
The Area = 0

```

17 华中科技大学人工智能与自动化学院 面向对象程序设计 黎云 王卓

## C++ 程序设计

•程序的输出结果表明，由于采用静态联编，在fun()函数中，由s所引用的对象执行的Area()操作，在程序编译阶段被选择到Point::Area()函数体代码上，从而导致输出不期望的结果，而编程者希望s所引用的对象执行的Area()操作，应选择到Rectangle::Area()函数体代码上，完成计算长方形面积的任务，这是静态联编所达不到的。

18 华中科技大学人工智能与自动化学院 面向对象程序设计 黎云 王卓

## C++ 程序设计

## § 5.5 虚函数

1. 静态联编：所谓“联编”是当函数调用时，链接上相应函数体的代码，这一程序执行过程称为联编。例如：

```

...
print (3.1415926);    //函数调用语句

...
print (double value) //函数头
{ cout << value; }    //函数体

```

19 华中科技大学人工智能与自动化学院 面向对象程序设计 黎云 王卓

## C++ 程序设计

C++有两种函数联编机制，**静态**联编和**动态**联编。静态联编（Static Binding或Early Binding先期联编）是在编译时，编译系统根据对象就能决定调用固定（程序运行期间不能改变）的函数体代码段。例如：编译系统根据浮点数3.1415926这一实例，调用print(double value){ cout << value; }这一函数体代码段，生成完成向CRT屏幕输出浮点数的机器码代码段。

20 华中科技大学人工智能与自动化学院 面向对象程序设计 黎云 王卓



## C++ 程序设计

在静态联编方式下，编译系统在编译该语句时就知  
道函数调用过程中将要使用哪些对象，并能决定调  
用哪一个函数，然后生成能完成语句功能的可执行  
代码。静态联编支持C++中的**运算符重载**和**函数名  
重载**这两种形式的多态性。与动态联编相比，其优  
点是运行开销小（仅传递参数，执行函数调用，清  
除堆栈等），程序执行速度快，缺点是灵活性差。

21 华中科技大学人工智能与自动化学院 面向对象程序设计 黎云 王卓

## C++ 程序设计

**2. 虚函数机制和动态联编技术：**为了实现“单界面，多实现版本”的思想，C++提供了一种解决方法，引用虚函数机制，即动态联编技术。

(1) 虚函数定义：虚函数是基类的**公有部分或保护部分**的某成员函数，在函数头前加上关键字“**virtual**”，其格式为：

```
class A {
public: (或protected : )
    virtual <返回类型> 成员函数名(参数表);
    . . .
};
```

22 华中科技大学人工智能与自动化学院 面向对象程序设计 黎云 王卓

## C++ 程序设计

```
#include <iostream>
using namespace std;
class A{
public:
    virtual void Display( )
    { puts("Class A"); }
};
class B : public A {
public:
    [virtual] void Display( )
    { puts("Class B"); }
};

void Show(A * p)
{ p -> Display( ); }
```

23 华中科技大学人工智能与自动化学院 面向对象程序设计 黎云 王卓

## C++ 程序设计

```
void main( )
{
    A * pa = new A;
    B * pb = new B;
    pa -> Display( ); //use A::Display( )
    pb -> Display( ); //use A::Display( )
    Show(pa);         //use A::Display( )
    Show(pb);         //use B::Display( )
}
```

该程序的输出结果：

```
Class A
Class B
Class A
Class B
```

24 华中科技大学人工智能与自动化学院 面向对象程序设计 黎云 王卓

## C++ 程序设计

①虚函数可以在一个或多个**公有**派生类中被重新定义（即可以有不同的实现版本），但在派生类中重新定义时，该同名虚函数的**原型**，其中包括返回类型、参数的个数和类型以及它们的顺序等，都必须**完全相同**，否则将不产生动态联编。

②一旦在基类中说明了一个虚函数，那么对于所有后续派生类中它**仍然是**虚函数，即使在派生类中没有用“**virtual**”指明。

25 华中科技大学人工智能与自动化学院 面向对象程序设计 黎云 王卓

## C++ 程序设计

③用虚函数实现程序运行时多态性的关键之处是，必须用指向**基类的指针访问虚函数**。只有在同一个指向基类的指针访问虚函数时，多态性才能实现。

如例程中，一个单界面的函数Show()，其形参采用指向基类A的指针(A \*p)。而在Show()函数体内指明函数调用“p-> Display();”，即用指向基类A的指针p去访问虚函数。但究竟是调 A::Display() 还是 B::Display()，取决于程序运行时指向基类A的指针p所具有的地址值。

在执行调用语句“Show(pa);”，实参传递给形参时，将pa赋给了指向基类A的指针p，则调用A::Display()。而执行“Show(pb);”时，则将pb赋给了指向基类A的指针p，而调用B::Display()。

26 华中科技大学人工智能与自动化学院 面向对象程序设计 黎云 王卓

## C++ 程序设计

因此是根据同一个指向基类的指针p传递消息“动态选中”，激活对象所属类的虚函数完成该实现版本所规定的功能。这种“动态选中”的性质称为**虚特性**，也称**虚函数机制**。

所谓“**动态联编**”（Dynamic Binding 或Late Binding滞后联编）是在程序运行时刻，将函数的界面（形式参数）与函数的不同实现版本（函数体）动态地进行匹配的联编过程，称为动态联编。

顺便指出，访问虚函数的Show(A \* p)函数，在实现了动态联编机制后能呈现多态性，故称为“多态函数”。

27 华中科技大学人工智能与自动化学院 面向对象程序设计 黎云 王卓

## C++ 程序设计

④如前所述，引用也是通过地址方式传递消息，对虚函数的访问也可通过对基类A的对象引用。此时Show()可定义为：

```
void Show(A & a){ a.Display(); }

void main()
{
    A a ; B b ;    //自动型对象
    Show(a);       //Call A::Display()
    Show(b);       //Call B::Display()
}
```

28 华中科技大学人工智能与自动化学院 面向对象程序设计 黎云 王卓

## C++ 程序设计

```

class Point {
public:
    Point(double i, double j)
    { x = i; y = j; }
    virtual double Area() const
    { return 0.0; }
private:
    double x, y;
};
class Rectangle : public Point {
public:
    Rectangle(double i, double j,
               double k, double l);
    double Area(void) const
    { return w * h; }
private:
    double w, h;
};

```

29

华中科技大学人工智能与自动化学院

面向对象程序设计

黎云 王卓

## C++ 程序设计

```

void fun(Point & s) //单接口界面
{ cout << "The Area = "
  << s.Area() << endl; }

```

```

void main()
{ Rectangle rec(3.0, 5.2,
                15.0, 25.0);

  fun(rec);
}

```

该程序的输出结果为: The Area = 375

30

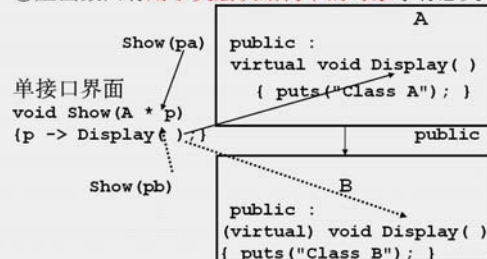
华中科技大学人工智能与自动化学院

面向对象程序设计

黎云 王卓

## C++ 程序设计

⑤虚函数只有用于类层次结构中的对象才有意义。



多实现版本

虚函数用于类层次结构中的对象

31

华中科技大学人工智能与自动化学院

面向对象程序设计

黎云 王卓

## C++ 程序设计

对于不是类层次结构中的基类成员函数，虽然在语法上可以定义为虚函数，但只会引起不必要的运行时间开销。一个虚函数是属于基类A公有部分或保护部分的成员函数，那么这个虚函数可以在A类的公有派生类B中重新定义，即一个函数原型完全相同，而函数体内的执行代码不同的同名虚函数，得到了在派生类B中的新实现版本，编程者可以用一个单界面Show (A \*p)调用多个实现版本A::Display()、B::Display()。

⑥虚函数必须是类的成员函数，非成员函数不能说明为虚函数，构造函数和静态成员函数也不能说明为虚函数。析构函数呢？

32

华中科技大学人工智能与自动化学院

面向对象程序设计

黎云 王卓



## C++ 程序设计

## (2) 虚函数和重载成员函数的区别

① 重载成员函数使用静态联编机制，虚函数采用动态联编机制

② 基类A中指明的虚函数f()，编程者若在派生类B中重新定义f()时，必须确保派生类B中的新函数与基类A中的虚函数f()具有**完全相同的函数原型**（其中包括函数名、参数的个数和类型以及排列顺序都必须相同），才能**覆盖**原虚函数f()而产生虚特性执行动态联编机制，否则只要有一个参数不同，编译系统就认为它是一个全新的函数，而不实现动态联编机制。

33 华中科技大学人工智能与自动化学院 面向对象程序设计 黎云 王卓

## C++ 程序设计

```
#include <iostream>
using namespace std;
class A{
public:
    virtual void Print(int a, int b)
    { cout << "a = " << a << " , b = "
      << b << endl; }
};
class B : public A {
public:
    virtual void Print(int a, double d)
    { cout << "a = " << a << " , d = "
      << d << endl; }
};
void Show(A * p)
{    p -> Print(3, 5.8); }
```

34 华中科技大学人工智能与自动化学院 面向对象程序设计 黎云 王卓

## C++ 程序设计

```
void main( )
{
    A * pa = new A;
    B * pb = new B;
    Show(pa);      //Call A::Print( )
    Show(pb);      //Call A::Print( )
    delete pa;
    delete pb;
}
该程序的输出结果:
a = 3 , b = 5
a = 3 , b = 5
```

35 华中科技大学人工智能与自动化学院 面向对象程序设计 黎云 王卓

## C++ 程序设计

本例中的多态函数Show(A \* p)总是调用基类的虚函数A::Print(int a, int b)，因为派生类中的虚函数A::Print(int a, double d)与基类虚函数A::Print()原型不是完全相同，这样前者不能覆盖后者，即**不执行动态联编机制**。

如前所述，重载成员函数必须在**同一个作用域内**，确保函数**参数个数**不同，或者至少有一个**参数的类型**不同。

36 华中科技大学人工智能与自动化学院 面向对象程序设计 黎云 王卓

## C++ 程序设计

## 3. 例程说明

```
#include <iostream>
using namespace std;
class A {
    int a;
public:
    virtual void f( )
    { puts("Function A::f( )"); }
    void g( )
    { puts("Function A::g( )"); }
    virtual void h( )
    { puts("Function A::h( )"); }
};
```

37

华中科技大学人工智能与自动化学院

面向对象程序设计

黎云 王卓

## C++ 程序设计

```
class B : public A {
    int b;
public:
    void g( )
    { puts("Function B::g( )"); }
    virtual void h( )
    { puts("Function B::h( )"); }
};
void Do(A & a)
{ a.f( ); a.g( ); a.h( ); }
void main( )
{
    A a;
    B b;
    Do(a);    Do(b);
}
```

38

华中科技大学人工智能与自动化学院

面向对象程序设计

黎云 王卓

## C++ 程序设计

该程序的输出结果:

```
Function A::f( )
Function A::g( )
Function A::h( )
Function A::f( )
Function A::g( )
Function B::h( )
```

39

华中科技大学人工智能与自动化学院

面向对象程序设计

黎云 王卓

## C++ 程序设计

(1) A为基类，公有继承得B类。因此在派生类B的公有部分仍有一个从基类A继承而来的虚函数f()，由于它在派生类中没有重新定义，其函数体代码仍然是：

```
virtual void B::f()
{ puts("Function A::f( )"); }
```

这也就是说，基类A的虚函数f()和派生类的虚函数f()具有完全相同的函数原型，则采用动态联编机制。

40

华中科技大学人工智能与自动化学院

面向对象程序设计

黎云 王卓





## C++ 程序设计

## 4. 虚函数表VFT (Virtual Function Table)

使用虚函数引入动态联编技术，使得class类型注入了很大的灵活性。然而这一灵活性却要增加额外开销。

(1) C++为每个至少拥有一个虚函数的类建立一个与之有关的虚函数表VFT，该表是由一系列函数指针组成。现以vt.cpp为例，对于基类A和公有派生类B，编译系统分别为它们建立了一个VFT如图5.4所示。

## C++ 程序设计

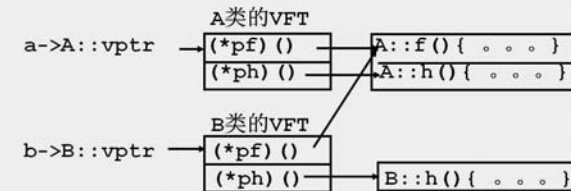


图5.4 对象 a和b的虚函数表与寻找路径

## C++ 程序设计

(2) 一个类只有一个虚函数表VFT，不管该类建立了多少个对象，每个类的VFT由它的所有对象共享，如图5.4所示，每个对象都具有一个指针vp\_ptr，该指针指向本对象所属类的虚函数表VFT。

(3) 虚函数表VFT是自动生成的，其内的函数指针也是隐含的，编程者不必编写操作它们的语句。

(4) 基类A的VFT包含两个函数指针A::(\*pf)()和A::(\*ph)()，分别指向两个虚函数A::f()和A::h()。基类对象a含有一个指针A::vp\_ptr指向A类的虚函数表VFT。

## C++ 程序设计

派生类B中除了包含有从A类继承的虚函数A::f()外，并对基类A中的虚函数h()重新加以定义为B::h()。因此B类的VFT也包含两个函数指针B::(\*pf)()和B::(\*ph)()，前者指向虚函数A::f()，另一个指向B::h()。公有派生类B的对象b含有一个指针B::vp\_ptr指向B类的虚函数表VFT。

## C++ 程序设计

(5) 从图5.4可知, 非虚函数 $A::g()$ 和 $B::g()$ 在虚函数表VFT中没有入口地址, 即无函数指针指向它们。

(6) 调用虚函数时, C++是通过对象查找到对应类的VFT。例中在调用多态函数 $Do(A \& a)$ 时, 用基类对象 $a$ 作为实参(调用语句为:  $Do(a)$ ), 通过对象 $a$ 的指针 $A::vptr$ 寻找到 $A$ 类的VFT。首先执行“ $a.f()$ ”语句, 从 $A$ 类的VFT中查到对应的函数指针 $A::(*pf)()$ , 从而引导到调用虚函数 $A::f()\{...\}$ , 如此类推。

用 $B$ 类的对象 $b$ 调用多态函数 $Do(A \& a)$ 时, 实际上是按顺序调用了 $A::f()$ 、 $A::g()$ 和 $B::h()$ 等函数, 这是因为:

49 华中科技大学人工智能与自动化学院 面向对象程序设计 黎云 王卓

## C++ 程序设计

➤ 在执行实参 $b$ 传递给形参( $A \& a$ )的过程中, 由传递过来的对象 $b$ 的指针 $B::vptr$ 寻找到 $B$ 类的VFT。对于 $a.f()$ 函数调用, 从 $B$ 类的VFT查到函数指针 $(*pf)()$ 。由于 $B$ 类的 $f()$ 只是继承 $A$ 类的虚函数 $f()$ 而没有重新定义, 因此 $B$ 类的函数指针 $B::(*pf)()$ 是指向 $A::f()$ , 从而引导到执行 $A::f()\{...\}$ 。

➤ 由于 $g()$ 不是虚函数仍采用静态联编机制, 编译时将对象 $b$ 的引用隐式转换成 $A$ 类的对象引用, 直接调用 $A::g()\{...\}$ 。

➤ 对 $a.h()$ 函数调用, 经对象 $b$ 的指针 $B::vptr$ 寻找到 $B$ 类的VFT, 其内函数指针 $B::(*ph)()$ 指向了 $B::h()$ , 因此执行 $B::h()\{...\}$ 。

50 华中科技大学人工智能与自动化学院 面向对象程序设计 黎云 王卓

## C++ 程序设计

## 虚析构函数

➤ 用多态性处理动态分配的类层次结构中的对象时存在一个问题。如果 $delete$ 运算符用于指向派生类对象的基类指针, 而程序中又显式地用该运算符删除每一个对象, 那么不管基类指针所指向的对象是何种类型, 也不管每个类的析构函数名是不同的这样一种情况, 系统都会为这些对象调用基类的析构函数。

➤ 这种问题有一种简单的解决方法, 即将基类析构函数声明为虚析构函数。这样就会使所有派生类的析构函数自动成为虚析构函数(即使它们与基类析构函数名不同)。

➤ 这时, 如果使用 $delete$ 运算符时, 系统会调用相应类的析构函数, 基类析构函数在派生类析构函数之后自动执行。

51 华中科技大学人工智能与自动化学院 面向对象程序设计 黎云 王卓

## C++ 程序设计

## § 5.6 纯虚函数和抽象类

## 5.6.1 纯虚函数

纯虚函数是一种特殊的虚函数, 它在基类中声明为虚函数, 但却没有定义实现部分, 而是在它的各派生类中定义各自的实现版本, 其一般格式为:

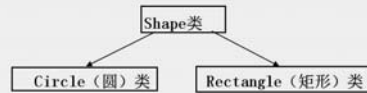
```
class 类名 {
    virtual <类型> 纯虚函数名(参数表) = 0;
    ...
};
```

其中,  $<类型>$ 是该纯虚函数返回值的数据类型。由此可见, 它与普通虚函数的基本一样, 仅在函数头后加上“ $= 0$ ”, 即将该虚函数指明为纯虚函数。

52 华中科技大学人工智能与自动化学院 面向对象程序设计 黎云 王卓

## C++ 程序设计

**注意：**在它的所有派生类中都必须定义该纯虚函数的实现版本，否则将产生编译错误。



Shape类和其派生类

## C++ 程序设计

## 5.6.2 抽象类

定义：含有纯虚函数的类，称为“抽象类”。

**至少**含有一个纯虚函数。

**只要**有一个纯虚函数就是抽象类。

## C++ 程序设计

**主要作用：**抽象类机制用来描述一般**抽象概念性**的东西。

如描述平面几何形状的**Shape**类，以它作为基类派生出表示具体形状的圆、矩形和椭圆等的变种Circle类、Rectangle类和ellipse类等才是描述具体实体的类。但是**抽象类却为它的所有派生类提供了一个统一的公共接口界面**，编程者可利用虚函数机制实现“**单接口界面、多实现版本**”这样灵活多变的功能。

Visual C++的标准类库MFC (Microsoft Foundation Class) 中的CObject类就是抽象类，它是MFC所有标准类的根，即最上层的基类，也可作为用户所开发的面向对象程序系统的根。

## C++ 程序设计

- 只能作为**基类**，而不能为抽象类**创建对象**。
- 不能用来说明函数参数的类型和返回类型，不能用于显式转换类型。
- 可以为抽象类定义**指针变量和引用**。



## C++ 程序设计

```
const float PI = 3.1415926;
//定义圆周率常量PI。
//定义一个描述抽象几何形状类Shape，即抽象类。
class Shape {
protected:
    float x, y;

    //平面几何形状的基点在屏幕上的x和y坐标的值。
    int fillpattern;

    //记录Draw()函数的填充方式。
    int color;

    //记录Draw()函数的填充颜色。
```

57 华中科技大学人工智能与自动化学院 面向对象程序设计 黎云 王卓

## C++ 程序设计

```
public:
    Shape(float h = 0, float v = 0,
          int fill = 0); //构造函数。
    float GetX(void) const
    { return x; } //读取x坐标值。
    float GetY(void) const
    { return y; } //读取y坐标值。
    void SetPoint(float h, float v);
    //设定平面几何形状基点的x和y坐标值。
    int GetFill(void) const;
    //返回填充方式。
    void SetFill(int fill) const;
    //设置填充方式。
    //纯虚函数，其派生类必须定义各自的求面积和周长的成员函数。
```

58 华中科技大学人工智能与自动化学院 面向对象程序设计 黎云 王卓

## C++ 程序设计

```
virtual float Area(void) const = 0;
virtual float Perimeter(void) const = 0;
virtual void Draw(void) const;
};
Shape::Shape(float h, float v, int fill) : x(h),
y(v), fillpattern(fill)
{ } //函数体为空。

/*在Shape的派生类Draw()成员函数进行初始化填充方式时调用的虚函数。*/
void Shape::Draw(void) const
{ setfillstyle(fillpattern,
               color);
  //调用BC31标准图形库中的setfillstyle()
}
//从Shape类公有继承的派生类Circle。
```

59 华中科技大学人工智能与自动化学院 面向对象程序设计 黎云 王卓

## C++ 程序设计

```
class Circle : public Shape {
protected:
    float radius;

public:
    Circle(float h = 0, float v = 0,
           float r = 0, int fill = 0);
    //初始化圆心、半径和填充方式的构造函数。

    float GetRadius(void) const; //读圆半径值。
    void SetRadius(float r); //设置圆半径值。
    virtual float Area(void) const; //求圆面积。
    virtual float Perimeter(void) const; //求圆周长。

    virtual void Draw(void) const;
    //调用Shape类的Draw()初始化填充方式，再调用库函数circle()画圆
};
```

60 华中科技大学人工智能与自动化学院 面向对象程序设计 黎云 王卓

## C++ 程序设计

```
//Circle类的实现部分。
Circle::Circle(float h, float v, float r, int fill) :
Shape(h, v, fill) { radius = r; }

float Circle::GetRadius(void) const {return radius;}

void Circle::SetRadius(float r) { radius = r; }

float Circle::Perimeter(void) const
{ return 2 * PI * radius; }

float Circle::Area(void) const
{ return PI * radius * radius; }

void Circle::Draw(void) const
{ Shape::Draw( );
  circle(x, y, radius);
} //调用Shape类的Draw( )初始化填充方式，再调用标准库函数circle( )
画圆。
```

61 华中科技大学人工智能与自动化学院 面向对象程序设计 黎云 王卓

## C++ 程序设计

```
//从Shape类公有继承的派生类Rectangle。
class Rectangle : public Shape {
protected :
    float length, width;
public :
    //初始化基点、长度和填充方式的构造函数。
    Rectangle(float h = 0, float v = 0, float l = 0,
float w = 0, int fill = 0);
    float GetLength(void) const;
    void SetLength(float l);
    float GetWidth(void) const;
    void SetWidth(float w);
    virtual float Area(void) const;
    Virtual float Perimeter(void) const;

    virtual void Draw(void) const;
    /* 调用 Shape 类的 Draw( ) 初始化填充方式，再调用库函数
rectangle( )画矩形。*/
};
```

62 华中科技大学人工智能与自动化学院 面向对象程序设计 黎云 王卓

## C++ 程序设计

```
//Rectangle类的实现部分。
Rectangle::Rectangle(float h, float v,
float l, float w, int fill)
: Shape(h, v, fill)
{ length = l; width = w; }
float Rectangle::GetLength(void) const { return length; }
void Rectangle::SetLength(float l) { length = l; }
float Rectangle::GetWidth(void) const { return width; }
void Rectangle::SetWidth(float w) { width = w; }
float Rectangle::Area(void) const
{ return length * width; }
float Rectangle::Perimeter(void) const
{ return 2 * (length + width); }
void Rectangle::Draw(void) const
{
Shape::Draw( );
//调用Shape类的Draw( )初始化填充方式。
rectangle(x, y, x + length,
y + width);
//再调用标准库函数rectangle( )画矩形。
}
```

63 华中科技大学人工智能与自动化学院 面向对象程序设计 黎云 王卓