西北工業大學

NORTHWESTERN POLYTECHNICAL UNIVERSITY

程序设计基础
Programming in C++

U10G13027/U10G13015

主讲：魏英，计算机学院

- 继承（inheritance）是面向对象程序设计的一个重要特性，是软件复用（software reusability）的一种形式。

- 它允许在原有类的基础上创建新的类，新类可以从一个或多个原有类中继承数据和函数，这样可以节省程序开发的时间，克服了程序无法重复使用的缺点。

- 在C++中，继承就是在一个已存在的类的基础上建立一个新的类。已存在的类称为基类（base class），又称为父类；新建立的类称为派生类（derived class），又称为子类。

表10-1 基类和派生类的现实示例

基类	派生类
大学生	本科生、研究生
形状	圆、三角形、矩形、球体、立方体
职员	教职员工、后勤人员
交通工具	汽车、轮船、飞机、自行车
控件	按钮、组合框、编辑框、列表框、状态条、工具条、卷滚条、选项卡、树形控件

很多情况下，一个类的对象也是另一个类的对象。例如，矩形是四边形。因而在C++中，矩形类Rect可以由四边形类Quad继承而来。在此，四边形类Quad是基类，矩形类Rect是派生类。矩形是特殊的四边形，但是如果断言四边形一定都是矩形，则是不对的，因为四边形中还有平行四边形或其他形状。

- 定义派生类的一般形式为：

```
class 派生类名 : 类派生列表 { //类体
      成员列表
};
```

- 类派生列表（class derivation list）指定了一个或多个基类（base class），具有如下形式：

```
访问标号 基类名
```

- 访问标号表示继承方式，可以是public（公有继承）、protected（保护继承）或private（私有继承），决定了对继承成员的访问权限。访问标号是可选的，如果未给出则默认为private（私有继承）。

- 派生类的成员列表定义了派生类自己新增加的数据成员和成员函数。

- 例如：

```
class Base { //Base类定义，基类
public: //基类公有成员
      void set(int a,int b,int c) { i=a,j=b,k=c; }
      int getk() { return k; }
protected: //基类保护成员
      int i, j;
private:  //基类私有成员
      int k;
};
class Derived : public Base { //Derived类定义，派生类
public: //派生类公有成员
      void setm(int a) { m=a; }
      int mul() { return m*i*j*getk(); }
private: //派生类私有成员
      int m;
};
```

- 在一个派生类中，其成员由两部分构成：一部分是从基类继承得到的，另一部分是自己定义的新成员，所有这些成员仍然分为public（公有）、private（私有）和protected（保护）三种访问属性。

10.1.3 派生类的构成

- 实际编程中，设计一个派生类包括3个方面的工作：
- （1）从基类接收成员。
- 除了构造函数和析构函数，派生类会把基类全部的成员继承过来。这种继承是没有选择的，不能选择接收其中一部分成员，而舍弃另一部分成员。
- （2）调整基类成员的访问。
- 派生类接收基类成员是程序员不能选择的，但是程序员可以对这些成员作出访问策略。此外，可以在派生类中声明一个与基类成员同名的成员，则派生类中的新成员会覆盖基类的同名成员。
- （3）在定义派生类时增加新的成员。
- 一般还应当自己定义派生类的构造函数和析构函数。

- 如果没有继承，一个类只有两种类型的访问者：类成员和类用户。

- 有了继承，就有了类的第三种访问者：派生类成员。

- 派生类通常需要访问基类成员，为了允许这种访问而仍然禁止对基类的一般访问，可以使用protected访问标号。类的protected部分仍然不能被类用户访问，但可以被派生类访问。只有基类类成员及其友元可以访问基类的private部分，派生类不能访问基类的私有成员。如图10.3所示。

- 类的保护成员用protected访问标号声明，可以认为protected访问标号是private和public的混合：

- ①像私有成员一样，保护成员不能被类用户访问。
- ②像公有成员一样，保护成员可以被该类的派生类访问。

- 如果在一个类中声明了保护成员，就意味着该类可能要用作基类，在它的派生类中会访问这些成员。

10.2.2 派生类成员的访问权限

- 不同的继承方式决定了基类成员在派生类中的访问属性。

（1）公有继承（public inheritance）

- 基类的公有成员和保护成员在派生类中保持原有访问属性，私有成员仍为基类私有。

（2）私有继承（private inheritance）

- 基类的所有成员在派生类中为私有成员。

（3）保护继承（protected inheritance）

- 基类的公有成员和保护成员在派生类中成了保护成员，私有成员仍为基类私有。

表10-2 不同继承方式下的基类特性和访问属性

继承方式	基类访问属性	在派生类的访问属性	派生类成员	派生类用户
public	**public**	**public**	可以访问	可以访问
	protected	**protected**	可以访问	不可以
	private	**inaccessible**	不可以	不可以
private	**public**	**private**	可以访问	不可以
	protected	**private**	可以访问	不可以
	private	**inaccessible**	不可以	不可以

续表10-2 不同继承方式下的基类特性和访问属性

继承方式	基类访问属性	在派生类的访问属性	派生类成员	派生类用户
protected	**public**	**protected**	可以访问	不可以
	protected	**protected**	可以访问	不可以
	private	**inaccessible**	不可以	不可以

例10.1

```
1   class A {
2       int k;
3     public:
4       int i;
5     protected:
6       void f2();
7       int j;
8   };
9   class B:public A {
10      int m;
11    public:
12      void f3();
13    protected:
14      void f4();
15  };
```

```
16  class C: protected B {
17      int n;
18    public:
19      void f5();
20  };
21  int main()
22  {
23      return 0;
24  }
```

类A是类B的公有基类，类B是类C的保护基类。各成员在不同类的访问属性如表10-3所示。

表10-3 例10.1各成员在不同类的访问属性

	i	f2	j	k	f3	f4	m	f5	n
基类A	公有	保护	保护	私有					
公有派生类B	公有	保护	保护	不可访问	公有	保护	私有		
保护派生类C	保护	保护	保护	不可访问	保护	保护	不可访问	公有	私有

- 一个公有派生类的对象在使用上可以被当做基类的对象，反之则禁止。具体表现在：

- ①派生类的对象可以赋值给基类对象；
- ②派生类的对象可以初始化基类的引用；
- ③派生类对象的地址可以赋给指向基类的指针。

- 例如：

```
class Base { }; //基类
class Derive : public Base { }; //公有派生类
Derive d; //定义派生类对象
```

- 这时，支持下面三种操作：

```
Base b=d; //派生类对象赋值给基类，复制基类继承部分
Base &rb=d; //基类引用到派生类对象
Base *pb=&d; //基类指针指向派生类对象
```

- 赋值兼容规则是C++多态性的重要基础之一。

- 在定义派生类时，派生类并没有把基类的构造函数和析构函数继承下来。因此，对继承的基类成员初始化的工作要由派生类的构造函数承担，同时基类的析构函数也需要由派生类的析构函数来调用。

- 1. 派生类构造函数的定义
- 在执行派生类的构造函数时，使派生类的数据成员和基类的数据成员同时都被初始化。其定义形式必须如下：

```
派生类名(形式参数列表) : 基类名(基类构造函数实参列表),派生类初始化列表
{
        派生类初始化函数体
}
```

例如：

```
class Point { int x,y;
public: Point(int a,int b):x(a),y(b) { } //构造函数
};
class Rect : public Point
{ int h,w;
  public:
   Rect(int a,int b,int c,int d):Point(a,b),h(c),w(d) { }
  //派生类构造函数
};
```

除去一个类从一个基类派生，C++还支持一个派生类同时继承多个基类。

- 1. 多重继承派生类的定义
- 如果已经定义了多个基类，那么定义多重继承的派生类的形式为：

```
class 派生类名:访问标号1 基类名1,访问标号2 基类名2,...
{ //类体
      成员列表
};
```

- 例如：

```
class A { };
class B : public A { }; //A→B
class C : public A { }; //A→C
class D : public B,public C { }; //A→B,C→D
```

- 2. 多重继承派生类的构造函数
- 多重继承派生类的构造函数形式与单一继承时的构造函数形式基本相同，只是在派生类的构造函数初始化列表中调用多个基类构造函数。一般形式为：

```
派生类名(形式参数列表) : 基类名1(基类1构造函数实参列表),
                          基类名2(基类2构造函数实参列表),
                          …,
                          派生类初始化列表
{
        派生类初始化函数体
}
```

- 其调用顺序是：
  ①调用基类构造函数，各个基类按定义时的次序先后调用；
  ②执行派生类初始化列表；
  ③执行派生类初始化函数体；

```
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<<"age"<<age<<"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(string nam,char s,float sco)
   {
      name1=nam;
      sex=s;
      score=sco;
   }  //构造函数
   void display1( )  //输出学生有关数据
   {
      cout<<"name:"<<name1<<"sex:"<<sex<<"score:"<<score<<endl;
   }
   protected :    //保护部分
            string name1;
            char sex;
            float score;
 };
```

```
class Graduate:public Teacher,public Student  //派生类Graduate
{
   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( )  //输出研究生的有关数据
   {
      cout<<"name:"<<name<<endl;
      cout<<"age:"<<age<<endl;
      cout<<"sex:"<<sex<<endl;
      cout<<"score:"<<score<<endl;
      cout<<"title:"<<title<<endl;
      cout<<"wages:"<<wage<<endl;
   }
   private:
        float wage;   //工资
};
```

```
int main( )
{
   Graduate grad1("Wang-li",24,'f',"assistant",89.5,1234.5);
   grad1.show( );
   return 0;
}
```

程序运行结果如下:

```
name: Wang-li
age: 24
sex:f
score: 89.5
title: assistance
wages: 1234.5
```

- 1. 二义性问题
- 多重继承时，多个基类可能出现同名的成员。在派生类中如果使用一个表达式的含义能解释为可以访问多个基类的成员，则这种对基类成员的访问就是不确定的，称这种访问具有二义性（ambiguous）。C++要求派生类对基类成员的访问必须是无二义性的。

- 例如：

```
class A {public:
      void fun() { cout<<"a.fun"<<endl; }
};
class B {public:
      void fun() { cout<<"b.fun"<<endl; }
      void gun() { cout<<"b.gun"<<endl; }
};
class C:public A,public B {public:
      void gun() { cout<<"c.gun"<<endl; }
      void hun() { fun(); } //出现二义性
};
```

```
C c,*p=&c;
c.A::fun(); //成员名限定消除二义性
c.B::fun(); //成员名限定消除二义性
p->A::fun(); //成员名限定消除二义性
p->B::fun(); //成员名限定消除二义性
```

- C++提供虚基类（virtual base class）的机制，使得在继承共同基类时只保留一份成员。
- 1. 虚基类的定义
- 虚基类是在派生类定义时，指定继承方式时声明的。声明虚基类的一般形式为：

```
class 派生类名: virtual 访问标号 虚基类名,...
{ //类体
      成员列表
};
```

- 需要注意，为了保证虚基类成员在派生类中只继承一次，应当在该基类的所有直接派生类中声明为虚基类。否则仍然会出现对基类成员的多次继承。

2.虚基类的初始化

- 如果在虚基类中定义了带参数的构造函数，而且没有定义默认构造函数.则在其**所有派生类**中，通过构造函数的初始化表对虚基类进行初始化。例如：

```
class A { public: A(int) {} }; //定义基类
class B : virtual public A { public: B(int
a):A(a) {} }; //对基类A初始化
class C : virtual public A { public: C(int
a):A(a) {} }; //对基类A初始化
class D : public B,public C { public: D(int
a):A(a),B(a),C(a) {} };
```

- **在最后的派生类中不仅要负责对其直接基类进行初始化，还要负责对虚基类初始化。**

- 面向对象程序设计的真正力量不仅仅是继承，而是允许派生类对象像基类对象一样处理，其核心机制就是**多态**和**动态联编**。

- **多态是指同样的消息被不同类型的对象接收时导致不同的行为**。所谓消息是指对类成员函数的调用，不同的行为是指不同的实现，也就是调用了不同的函数。

- 2.静态联编
- **联编**（binding）又称绑定，就是将模块或者函数合并在一起生成可执行代码的处理过程。

- **在编译阶段就将函数实现和函数调用绑定起来称为静态联编**（static binding）。静态联编在编译阶段就必须了解所有的函数或模块执行所需要的信息，它对函数的选择是基于指向对象的指针（或者引用）的类型。C语言中，所有的联编都是静态联编，C++中一般情况下联编也是静态联编。

例10.2

```
 1  #include <iostream>
 2  using namespace std;
 3  class Point { //Point类表示平面上的点
 4      double x,y; //坐标值
 5  public:
 6      Point(double x1=0,double y1=0) : x(x1),y(y1) { }
//构造函数
 7      double area() { return 0; } //计算面积
 8  };
 9  class Circle:public Point { //Circle类表示圆
10      double r; //半径
11  public:
12      Circle(double x,double y,double r1) :
                          Point(x,y),r(r1) { } //构造函数
13      double area() { return 3.14*r*r; } //计算面积
14  };
```

```
15  int main()
16  { Point a(2.5,2.5); Circle c(2.5,2.5,1);
17    cout<<"Point area="<<a.area()<<endl; //基类对象
18    cout<<"Circle area="<<c.area()<<endl; //派生类对象
19    Point *pc=&c , &rc=c; //基类指针、引用指向或引用派生类
对象
20    cout<<pc->area()<<endl; //静态联编基类调用
21    cout<<rc.area()<<endl; //静态联编基类调用
22    return 0;
23  }
```

运行结果：

```
Point area=0
Circle area=3.14
0
0
```

- 3.动态联编

- **在程序运行的时候才进行函数实现和函数调用的绑定称为动态联编**（dynamic binding）。

- 如果希望“pc->area()”调用Circle::area()，则需要将Point类的area函数设置成虚函数。其定义形式为：

```
virtual double area() { return 0; } //计算面积
```

- 1. 虚函数的定义
- 虚函数只能是类中的一个成员函数。在成员函数定义或声明前面加上关键字virtual，即定义了虚函数：

```
class 类名 { //类体
        …
        virtual 返回类型 函数名(形式参数列表); //虚函数
        …
};
```

- 例如：

```
class Point { //Point类表示平面上的点
        …
        virtual double area(); //虚函数声明
        virtual double volumn() { return 0; } //虚函数
定义
};
```

- 需要注意，virtual只在类体中使用。

10.6.2 虚函数

```
class Student
{  public:
       Student(int,string,float);
       virtual void display();
   protected:
       int num;string name;float score;
} ;

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

void Student::display()
{  cout<<"num:"<<num<<"\nname:"
         <<name<<"\nscore:"<<score
         <<"\n\n";
}
```

```
class Graduate: public Student
{  public:
       Gtaduate(int,string,float,float);
       virtual void display();
   private:
       float pay;
} ;

Graduate::Graduate
(int n, string nam,float s,float p)
::Student(n,nam,s),pay(p){}

void Graduate::display()
{cout<<"num:"<<num<<"\nname:"
        <<name<<"\nscore:"<<score
        <<"\npay="<<pay<<endl;

}
```

```
int main()
{   Student stud1(1001,"Li",87.5);
    Graduate grad1(2001,"Wang",98.5,563.5);
    Student *pt=&stud1;
    pt->display();
    pt=&grad1;
    pt->display();
    return 0;
}
```

如果没有虚函数的运行结果：

```
num:1001
name:Li
score:87.5

num:2001
name:Wang
score:98.5
```

- 在许多情况下，不能在基类中为虚函数给出一个有意义的定义，这时可以将它说明为纯虚函数（pure virtual function），将具体定义留给派生类去做。纯虚函数的定义形式为：

```
virtual 返回类型 函数名(形式参数列表)=0;
```

- 即在虚函数的原型声明后加上“=0”，表示纯虚函数根本就没有函数体。
- 纯虚函数的作用是在基类中为其派生类保留一个函数的名字，以便派生类根据需要对它进行定义。

- 包含有纯虚函数的类称为**抽象类**（abstract class）。一个抽象类只能作为基类来派生新类，所以又称为抽象基类（abstract base class）。抽象类不能定义对象。

例10.4

```
#include <iostream>
using namespace std;
class Sharp { //Sharp类，抽象类
public:
    virtual double area() =0; //纯虚函数
    virtual double volumn() =0; //纯虚函数
};
class Circle : public Sharp { //Circle类表示圆
public:
    Circle(double a):r(a) { }
    virtual double area() { return 3.1415926*r*r; }
//虚函数
    virtual double volumn() { return 0; }; //虚函数
private:
    double r;
};
```

10.6.5 抽象类

```
16  class Cylinder : public Circle { //Cylinder表示圆柱体
17  public:
18      Cylinder(double a,double b):Circle(a),h(b) { } //调用
Circle类构造函数
19      virtual double volumn() { return area()*h; }; //虚函数
20  private:
21      double h;
22  };
23  int main()
24  {
25      Circle a(10.0); //定义Circle对象
26      Cylinder b(5.6,10.5); //定义Cylinder对象
27      cout<<a.area()<<","<<b.volumn()<<endl; //静态联编
28      Sharp *pb; //定义基类指针
29      pb=&b; //指向Circle对象
30      cout<<pb->area()<<","<<pb->volumn()<<endl; //动态联编
31      return 0;
32  }
```


程序设计