第 5 章　继承与派生

本章主要内容：

(1) 类的继承有关概念：继承与派生、基类与派生类、单继承与多继承等。

(2) 不同继承方式下基类成员的访问控制问题。

(3) 派生类的构造函数和析构函数的定义，派生类对象的构造顺序。

(4) 多继承及虚基类的概念。

(5) 图形界面程序设计：几何形状的面积和体积计算。

继承(Inheritance)是面向对象程序设计的一个重要特征。C++ 中的继承可以以已有的类为基础定义新的类，新类会自动拥有已有类(父类)的成员，新类只需要定义父类中没有的成员。继承机制实现了软件代码的重用，从而缩短了软件开发的周期。

5.1　继承概述

继承使类之间建立一种层次关系，类似于现实世界中的分类层次关系。人们把具有共同特征的事物归为一级大类，该大类提供一般化的描述，该类可以进一步细化分为几个子类别。比如图 5-1，中国图书馆图书分类法将所有学科的图书按其学科内容分成几大类，每一大类下分许多小类，每一小类下再分子类。其中，子类自动拥有大类的特征。

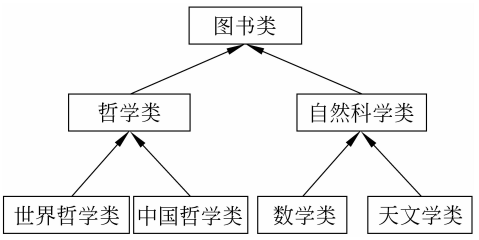

图 5-1　图书分类的层次关系

类的这种层次关系具有传递性，就是哲学类、自然科学类图书都是图书，都具有图书的特征，也就是说哲学类、自然科学类图书继承了图书类，除了继承的特征之外，它们还有各自不同的特征。而数学类和天文学类都属于自然科学类图书，则数学类和天文学类图书继承了自然科学类，并且通过自然科学类间接继承了图书类，如此，形成了多层次的继承传递关系。C++ 支持这种继承机制，通过已有类定义一个新类，新类自动继承基类的成员，实现了软件代码的复用。

问题引入：假设现有一个雇员类 Employee，用它来记录某公司雇员的姓名、年龄、薪水等。公司内除了一般雇员之外，还有相应的行政管理人员。从广义上来讲，他们也是雇员，有姓名、年龄和薪水等。然而，他们与普通雇员有不同之处，例如他们具有行政级别，可以管理其他雇员等。为此需要定义一个名为 Manager 的新类。当然我们可以单独定义一个 Manager 的新类，但这样会重复 Employee 中的代码。为了避免重复编码，我们应用继承机制以 Employee 类为基础定义新的 Manager 类。

这种在已有类的基础上定义(派生)新类的方式就是继承机制。其中，被继承的已有

类称为基类(父类),派生出的新类称为派生类(子类)。继承使得子类继承父类的属性和操作,并可以增加新的属性和操作,还可以改造不适用的父类成员,以适用于求解新的问题。因此,上述问题中,以 Employee 为基类定义派生类 Manager,Manager 类就继承了基类的姓名、年龄、薪水等成员。这种由一个基类派生定义出一个新的类称为单继承方式,也可以由多个基类共同派生出一个新的类,则称为多继承方式。我们主要讲解单继承方式。

5.2 基类与派生类

在公司雇员问题中,以 Employee 为基类定义派生类 Manager,需要先定义基类 Employee。Employee 类包含的数据成员有姓名、年龄和薪水,包含成员函数有构造函数、析构函数和输出函数。Employee 类的定义如下。

```
//定义基类 Employee
class Employee
{
public:
    Employee(char * n ="Noname",short a =0,float s =0)     //构造函数
    {  name =new char[strlen(n)+1];
       strcpy(name,n);
       age =a;
       salary =s;
    }
    void Print()                                           //输出函数
    {   cout<<"name:"<<name<<endl;
        cout<<"age:"<<age<<endl;
        cout<<"salary:"<<salary<<endl;
    }
    ~Employee()                                            //析构函数
    { delete []name; }
private:
    char * name;                                           //姓名
    short age;                                             //年龄
    float salary;                                          //薪水
};
```

5.2.1 派生类的定义

如何以 Employee 为基类定义派生类 Manager 呢?

首先,派生类定义的一般语法形式如下:

class 派生类名:访问控制 基类名 1,访问控制 基类名 2,…,访问控制 基类名 n

```
{
    数据成员和成员函数声明
};
```

其中,访问控制表示继承的方式,具体分为三种方式。

(1) public:表示公有继承方式。

(2) protected:表示保护继承方式。

(3) private:表示私有继承方式。

如果省略继承方式,则默认为私有继承。如果是对 struct 结构的继承,则默认为公有。一个派生类可以继承多个基类称为多继承,如果只继承一个基类就称为单继承。

根据继承的一般语法形式,派生类 Manager 可以简单定义如下:

```
class Manager: public Employee
{ };
```

其中,public Employee 表示 Manager 类是从 Employee 类中公有派生定义的,它继承了所有属于 Employee 的数据成员和成员函数(除了构造函数和析构函数)。所以,尽管 Manager 类中没有定义新的成员,但它实际上具有 Employee 类的数据成员和成员函数。所以,下面的语句是合法的:

```
Manager m;
m.Print();                  //打印输出 Manager 的 name、age 和 salary
```

在此处,Manager 类为派生类,派生类的对象 m 可直接调用从基类继承的成员函数 Print()。前提是,Manager 类是 Employee 类的公有派生类,而且 Print()函数是基类的公有成员。所以,Print()也成为派生类的公有成员函数,也就是 public 继承方式使得基类的公有成员在派生类中仍然为公有。

一般情况下,派生类不会仅仅简单继承基类的成员,继承时也应有“变异”,例如,Manager 中的数据成员应能描述行政级别。所以,应在派生类中增加新的数据成员或成员函数。我们将 Manager 类进一步完善定义如下:

```
class Manager: public Employee
{
public:
    void Print_level()
    { cout<<"level:"<<level<<endl; }
private:
    int level;
};
```

这样,派生类 Manager 除了继承 Employee 类中的特征,还加入了私有成员 level 和公有成员函数 Print_level(),分别表示行政级别和打印级别。派生类的成员结构如图 5-2 所示。派生类无条件继承了基类成员,它的新增公有成员和继承自基类的公有成员一起提供了操纵派生类的公有接口。

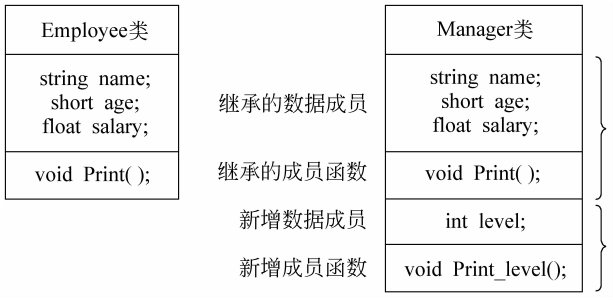

图 5-2 基类与派生类成员构成示意图

📖**注意**：原则上，派生类不会继承基类的以下成员。

(1) 基类的构造函数和析构函数。

(2) 基类的友元。

虽然基类的构造函数和析构函数没有被继承，但一个派生类的新对象被创建或撤销时总是会调用基类的构造函数和析构函数，以完成对基类继承来的成员的处理工作。

以上 Employee 类与 Manager 类的完整程序测试如例 5-1 所示。

【例 5-1】 继承与派生类的定义示例。

```
//D:\QT_example\5\Example5_1.cpp
#include <iostream>
#include <cstring>
using namespace std;
//定义基类 Employee
class Employee
{
public:
    Employee(char * n = " Noname",short a = 0,float s = 0)   //构造函数
    {   name = new char[strlen(n)+1];
        strcpy(name,n);
        age = a;
        salary = s;
    }
    void Print()                                             //输出函数
    {   cout<<"name:"<<name<<endl;
        cout<<"age:"<<age<<endl;
        cout<<"salary:"<<salary<<endl;
    }
    ~Employee()                                              //析构函数
    { delete[]name; }
private:
    char * name;
    short age;
```

```
    float salary;
};
//定义派生类 Manager
class Manager: public Employee
{
public:
    void Set_level(int l =0)
    { level =l; }
    void Print_level()
    { cout<<"level:"<<level<<endl; }
private:
    int level;
};
int main()
{
    Employee e("lihua",30,3500);                    //定义一个雇员对象
    e.Print();
    Manager m;                                      //定义一个管理员对象
    m.Set_level();
    m.Print();
    m.Print_level();
    return 0;
}
```

程序运行结果为：

```
name:lihua
age:30
salary:3500
name: Noname
age:0
salary:0
level:0
```

从运行结果可看出，派生类的对象 m 输出的是成员的初始化值，通过调用基类构造函数实现了对基类继承的成员初始化。如果需要为这部分成员传递实际的参数值，则需要定义派生类的构造函数为其传递数据，详见 5.3 节的内容。

5.2.2 访问控制

一个派生类中包含了从基类继承来的成员和自己定义的成员，当创建一个派生类对象时，系统会建立所有这些成员，为它们分配相应的存储空间。派生类对象对自己定义的成员的访问决定于成员的访问属性，但是派生类对象对继承来的成员的访问，除了与基类成员本身的访问属性有关外，还与继承方式有关。下面分别按继承的三种方式进行讨论。

1. 公有继承(public)

通过公有继承方式,基类的公有成员成为派生类的公有成员,基类的保护成员成为派生类的保护成员,但基类的私有成员在派生类中不可见,即不能被这个派生类访问。

【例 5-2】 公有继承的示例,基类定义私有数据成员。

```
//D:\QT_example\5\Example5_2.cpp
#include <iostream>
using namespace std;
//基类 Box
class Box
{
public:
    void SetLength(int l)
    { length=l; }
    void SetWidth(int w)
    { width=w; }
    void SetHeight(int h)
    { height=h; }
    void ShowBox()
    {   cout<<"length="<<length<<endl;
        cout<<"width="<<width<<endl;
        cout<<"height="<<height<<endl;
    }
private:
    int length,width,height;
};
//派生类 ColorBox
class ColorBox : public Box            //public 公有继承
{
public:
    void SetColor(char * c)
    { color =c; }
    void ShowColorBox()
    {   ShowBox();                     //调用继承来的 public 成员函数
        cout<<"color="<<color<<endl; //访问自己定义的 private 成员
    }
private:
    char * color;                      //新增加的成员,体现派生类的变化、进化
};
//主函数
int main()
```

```
{
    ColorBox ob1;                          //定义派生类的对象,分配内存,初始化数据成员
    ob1.SetLength(3);
    ob1.SetWidth(1);
    ob1.SetHeight(2);
    ob1.SetColor("red");
    cout<<"ob1.ShowBox():\n";
    ob1.ShowBox();
    cout<<"ob1.ShowColorBox():\n";
    ob1.ShowColorBox();
    return 0;
}
```

程序运行结果为:

```
ob1.ShowBox():
length=3
width=1
height=2
ob1.ShowColorBox():
length=3
width=1
height=2
color=red
```

程序中,ob1 作为派生类 ColorBox 的对象,包含的成员及各成员的访问属性如下。

私有数据成员:color。

不可访问数据成员:length、width、height。

公有成员函数:SetLength()、SetWidth()、SetHeight()、ShowBox()、SetColor()和 ShowColorBox()。

其中,length、width、height 是基类的私有数据成员,派生类会自动继承,但无论以何种方式继承都不能访问该成员,保持了基类私有成员的隐蔽性。而基类的公有成员函数通过公有继承后就成为派生类的公有成员。所以,ob1 调用继承的公有成员 SetLength()、SetWidth()、SetHeight()实现对继承的数据 length、width 和 height 的访问。

为了验证在派生类中不能访问继承的基类私有数据成员,可以修改 ShowColorBox()函数为如下。

```
void ShowColorBox()
{   cout<<"length="<<length<<endl;       //Error,不可访问
    cout<<"width="<<width<<endl;         //Error,不可访问
    cout<<"height="<<height<<endl;       //Error,不可访问
    cout<<"color="<<color<<endl;         //Ok,访问自己的 private 成员
}
```

编译出现错误提示,因为 Box::length、Box::width 和 Box::height 是私有的。如果

需要在派生类内访问基类继承的数据成员，但又不想在类外被访问，就需要将基类 Box 的数据成员改变为 protected 访问属性。这样，在上面的 ShowColorBox()函数中就可以直接访问基类的保护成员。修改的基类 Box 定义如下。

```
class Box
{
public:
    void SetLength(int l)
    { length=l; }
    void SetWidth(int w)
    { width=w; }
    void SetHeight(int h)
    { height=h; }
    void ShowBox()
    {   cout<<"length="<<length<<endl;
        cout<<"width="<<width<<endl;
        cout<<"height="<<height<<endl;
    }
protected:                    //保护访问属性
    int length,width,height;
};
```

此时，派生类继承的数据成员 length、width 和 height 就成了保护成员，可以在类内访问，但不可以在类外通过对象访问，则上面修改的派生类成员函数 ShowColorBox()也就没有问题了。所以，保护成员是专门在继承中定义的，保护属性限制了基类的成员只在其派生类中可见，而在类外是不可见的。

总之，公有继承的特点是基类的 public 和 protected 成员作为派生类的成员时，都保持了原有的访问属性。基类的公有成员也是派生类的公有接口，既可以在类内也可以在类外被访问；基类的保护成员作为派生类的保护成员可在类内访问，但不可在类外访问。基类的私有成员虽然被继承，但在派生类内或类外都不可访问。

2. 私有继承(private)

通过私有继承方式，基类的公有成员和保护成员都成为派生类的私有成员，只能在派生类内被访问，不能在类外通过对象访问，也不能被这个派生类的子类所访问；基类的私有成员在派生类中不可见，不能被派生类访问。

【例 5-3】 私有继承示例，基类定义保护数据成员。

```
//D:\QT_example\5\Example5_3.cpp
#include <iostream>
using namespace std;
//基类 Box
class Box
{
```

```
public:
    void SetLength(int l)
    { length=l; }
    void SetWidth(int w)
    { width=w; }
    void SetHeight(int h)
    { height=h; }
    void ShowBox()
    {   cout<<"length="<<length<<endl;
        cout<<"width="<<width<<endl;
        cout<<"height="<<height<<endl;
    }
protected:                                  //保护成员
    int length,width,height;
};
//派生类 ColorBox
class ColorBox : private Box                //私有继承
{
public:
    void SetColor(char * c)
    { color=c; }
    void ShowColorBox()
    {   cout<<"length="<<length<<endl; //访问继承的成员
        cout<<"width="<<width<<endl;   //访问继承的成员
        cout<<"height="<<height<<endl; //访问继承的成员
        cout<<"color="<<color<<endl;   //访问自己定义的 private 成员
    }
private:
    char * color;                           //新增的数据成员
};
int main()
{
    ColorBox ob1;                           //定义派生类的对象,所有成员分配内存
    ob1.SetWidth(1);                        //错误 1: 调用继承来的 private 成员函数
    ob1.SetColor("red");
    cout<<"ob1.ShowBox():\n";
    ob1.ShowBox();                          //错误 2: 调用继承来的 private 成员函数
    cout<<"ob1.ShowColorBox():\n";
    ob1.ShowColorBox();
    return 0;
}
```

私有继承时,ColorBox 类的对象 ob1 包含的成员及访问属性如下。

私有数据成员:length、width、height、color。

私有成员函数：SetLength()、SetWidth()、SetHeight()、ShowBox()。

公有成员函数：SetColor()、ShowColorBox()。

所以，程序中出现的两处编译错误，是因为对象 ob1 调用了从基类继承的私有成员函数。可见，私有继承限制了基类成员在派生类的访问权限，使得基类公有和保护成员成为派生类的私有成员，只能在类内被访问，若继续向下派生子孙类，则该私有成员就成了不可见成员，类内类外都无法访问。

3. 保护继承(protected)

通过保护继承方式，基类的公有成员和保护成员都成为派生类的保护成员，在派生类中或其子类内都可以被访问，但不能在类外被访问；而基类的私有成员在派生类中不可见。

修改例 5-3 中派生类 ColorBox 定义的继承方式为 protected：

```
class ColorBox :protected Box            //保护继承
{
    //成员定义与例 5-3 相同
};
```

则 ColorBox 类包含的成员及访问属性如下。

私有数据成员：color。

保护数据成员：length、width、height。

保护成员函数：SetLength()、SetWidth()、SetHeight()、ShowBox()。

公有成员函数：SetColor()、ShowColorBox()。

基类的公有和保护成员都成为派生类的保护成员，只能在 ColorBox 类内访问，而不能在类外通过 ob1 对象访问，测试过程如同例 5-3。

综上所述，在三种不同继承方式下，派生类从基类继承的成员的访问属性如表 5-1 所示。

表 5-1 不同继承方式的派生类继承成员的访问属性

基类成员属性	public	protected	private
公有继承	public	protected	不可见
私有继承	private	private	不可见
保护继承	protected	protected	不可见

可以得出如下结论。

(1) 无论哪种继承方式，基类的私有成员对派生类都是不可见的，但基类的公有和保护成员对派生类都是可见的。这样就保证了基类私有成员的隐蔽性。否则，如果派生类可以访问它的基类的私有成员，则该派生类的子类也可以访问这些数据，这将扩散对私有数据的访问，通过类层次结构将会丢失封装的优点。

(2) 派生类的对象对基类继承来的成员访问，取决于基类的成员在派生类中变成了什么类型的成员。例如，私有继承时，基类的公有成员和保护成员都变成了派生类中的私