面向对象的程序设计

——从第2讲开始

第2讲 类和对象

2.1 面向对象程序设计方法概述

对于规模比较小的程序，编程者可以直接编写出一个面向过程的程序，详细地描述每一瞬时的数据结构及对其的操作过程。但是当程序规模较大时，就显得力不从心了。C++就是为了解决编写大程序过程中的困难而产生的。

2.1.1 什么是面向对象的程序设计

面向对象的程序设计的思路和人们日常生活中处理问题的思路是相似的。在自然世界和社会生活中，一个复杂的事物总是由许多部分组成的。

当人们生产汽车时，分别设计和制造发动机、底盘、车身和轮子，最后把它们组装在一起。在组装时，各部分之间有一定的联系，以便协调工作。

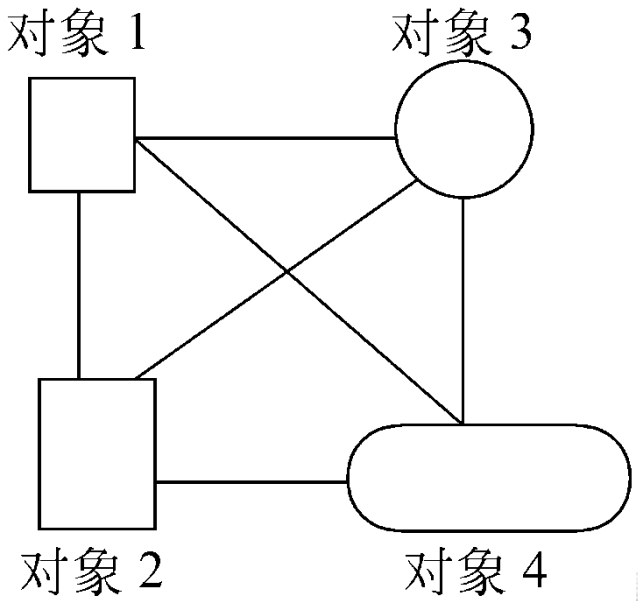
这就是面向对象的程序设计的基本思路。

为了进一步说明问题，下面先讨论几个有关的概念。

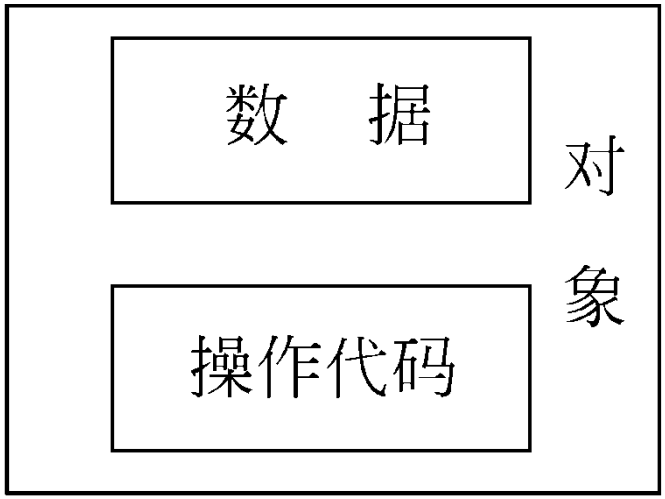
1. 对象

客观世界中任何一个事物都可以看成一个对象(object)。任何一个对象都应当具有这两个要素，即属性(attribute)和行为(behavior)，它能根据外界给出的信息进行相应的操作。在一个系统中的多个对象之间通过一定的渠道相互联系，如图。

要使某一个对象实现某一种行为(即操作)，应当向它传送相应的消息。



使用面向对象的程序设计方法设计一个复杂的软件系统时，首要的问题是确定该系统是由哪些对象组成的，并且设计这些对象。在C++中，每个对象都是由数据和函数(即操作代码)这两部分组成的，见下图。数据体现了前面提到的“属性”，函数是用来对数据进行操作的，就是前面提到的行为，在程序设计中也称为方法(method)。调用对象中的函数就是向该对象传送一个消息(message)，并要求该对象实现某一行为(功能)。



2. 封装与信息隐蔽

面向对象程序设计方法的一个重要特点就是“封装性”(encapsulation)。

所谓“封装”，指两方面的含义：

一是将有关的数据和操作代码封装在一个对象中，形成一个基本单位，各个对象之间相对独立，互不干扰。

二是将对象中某些部分对外隐蔽，即隐蔽其内部细节，只留下少量接口，以便与外界联系，接收外界的消息。这种对外界隐蔽的做法称为信息隐蔽(imformation hiding)。信息隐蔽还有利于数据安全，防止无关的人了解和修改数据。

C++的对象中的函数名就是对象的对外接口，外界可以通过函数名来调用这些函数来实现某些行为。

3. 抽象

在程序设计方法中，常用到抽象(abstraction）这一名词。抽象是将有关事物的共性归纳、集中的过程。C和C++中的数据类型就是对一批具体的数的抽象。

对象是具体存在的，如一个三角形可以作为一个对象，10个不同尺寸的三角形是10个对象。如果这10个三角形对象有相同的属性和行为，可以将它们抽象为一种类型，称为三角形类型。在C++中，这种类型就称为“类(class)”。这10个三角形就是属于同一“类”的对象。类是对象的抽象，而对象则是类的特例，或者说是类的具体表现形式。

4. 继承与重用

如果在软件开发中已经建立了一个名为A的“类”，又想另外建立一个名为B的“类”，而后者与前者内容基本相同，只是在前者的基础上增加一些属性和行为，只需在类A的基础上增加一些新内容即可。这就是面向对象程序设计中的继承机制。利用继承可以简化程序设计的步骤。

“白马”继承了“马”的基本特征，又增加了新的特征(颜色)，“马”是父类，或称为基类，“白马”是从“马”派生出来的，称为子类或派生类。

C++提供了继承机制，采用继承的方法可以很方便地利用一个已有的类建立一个新的类。这就是常说的“软件重用”(software reusability) 的思想。

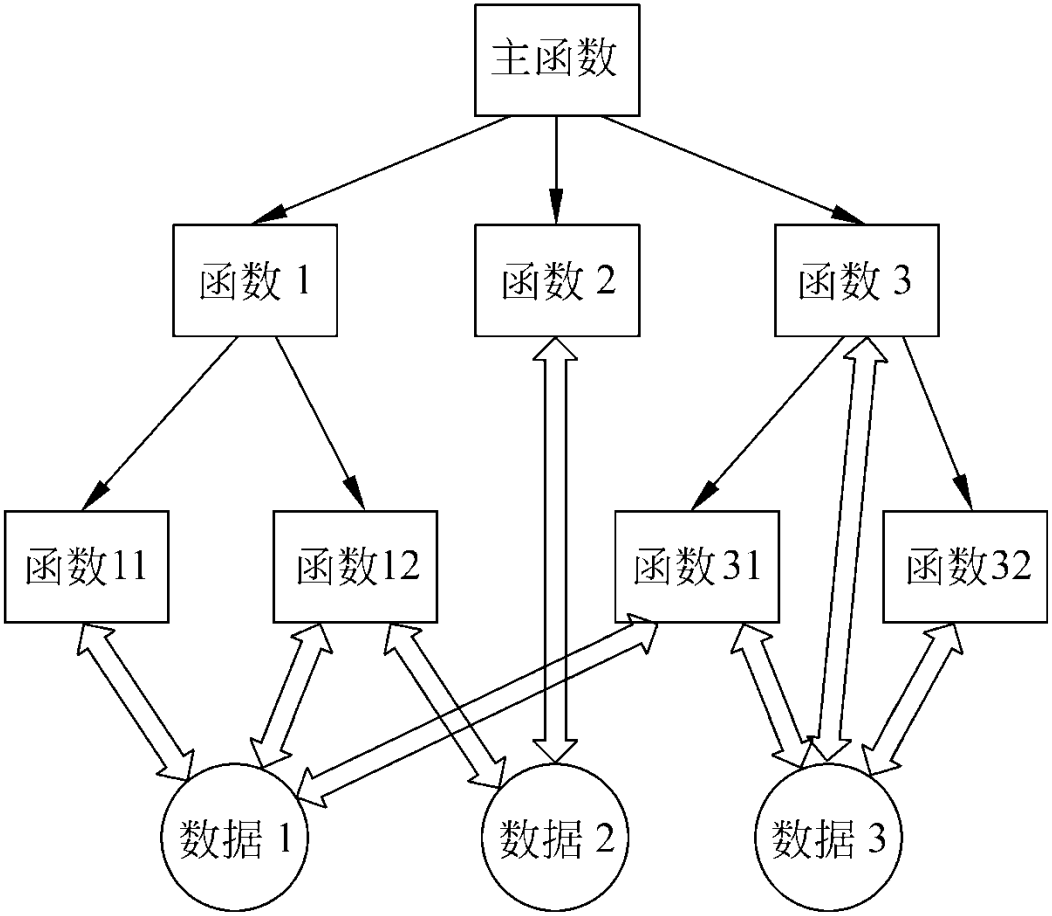
5. 多态性

如果有几个相似而不完全相同的对象，有时人们要求在向它们发出同一个消息时，它们的反应各不相同，分别执行不同的操作。这种情况就是多态现象。如，在Windows环境下，用鼠标双击一个文件对象(这就是向对象传送一个消息)，如果对象是一个可执行文件，则会执行此程序，如果对象是一个文本文件，则启动文本编辑器并打开该文件。

在C++中，所谓多态性(polymorphism)是指：由继承而产生的相关的不同的类，其对象对同一消息会作出不同的响应。多态性是面向对象程序设计的一个重要特征，能增加程序的灵活性。

2.1.2 面向对象程序设计的特点

传统的面向过程程序设计是围绕功能进行的，用一个函数实现一个功能。所有的数据都是公用的，一个函数可以使用任何一组数据，而一组数据又能被多个函数所使用（见图）。



面向对象程序设计采取的是另外一种思路。它面对的是一个个对象,一组操作调用一组数据。

程序设计者的任务包括两个方面：一是设计所需的各种类和对象，即决定把哪些数据和操作封装在一起；二是考虑怎样向有关对象发送消息，以完成所需的任务。这时他如同一个总调度，不断地向各个对象发出命令，让这些对象活动起来(或者说激活这些对象)，完成自己职责范围内的工作。各个对象的操作完成了，整体任务也就完成了。

2.1.3 类和对象的作用

类是C++中十分重要的概念，它是实现面向对象程序设计的基础。类是所有面向对象的语言的共同特征，所有面向对象的语言都提供了这种类型。一个有一定规模的C++程序是由许多类所构成的。

C++支持面向过程的程序设计，也支持面向对象的程序设计。在面向过程的结构化程序设计中，人们常使用这样的公式来表述程序：

程序=算法＋数据结构

即：算法和数据结构两者是互相独立、分开设计的，面向过程的程序设计是以算法为主体的。

在实践中人们逐渐认识到算法和数据结构是互相紧密联系不可分的，应当以一个算法对应一组数据结构，而不宜提倡一个算法对应多组数据结构，或一组数据结构对应多个算法。

面向对象程序设计就是把一个算法和一组数据结构封装在一个对象中。因此，就形成了新的观念：

对象 = 算法 ＋ 数据结构

程序 = (对象+对象+对象+…) + 消息

消息的作用就是对对象的控制。程序设计的关键是设计好每一个对象，及确定向这些对象发出的命令，使各对象完成相应操作。

2.1.4 面向对象的软件开发

随着软件规模的迅速增大，软件人员面临的问题十分复杂。需要规范整个软件开发过程，明确软件开发过程中每个阶段的任务。

面向对象的软件工程包括以下几个阶段：

1. 面向对象分析(object oriented analysis,OOA)

面向对象的分析，要按照面向对象的概念和方法，在对任务的分析中，从客观存在的事物和事物之间的关系，归纳出有关的对象(包括对象的属性和行为)以及对象之间的联系，并将具有相同属性和行为的对象用一个类(class)来表示。

2. 面向对象设计(object oriented design,OOD)

首先是进行类的设计，类的设计可能包含多个层次(利用继承与派生)。然后以这些类为基础提出程序设计的思路和方法，包括对算法的设计。

3. 面向对象编程(object oriented programming, OOP)

根据面向对象设计的结果，用一种计算机语言把它写成程序，显然应当选用面向对象的计算机语言(例如C++)。

4. 面向对象测试(object oriented test,OOT)

测试的目的是发现程序中的错误并改正它。面向对象测试以类作为测试的基本单元。

5. 面向对象维护(object oriented soft maintenance, OOSM)

因为对象的封装性，修改一个对象对其他对象影响很小。利用面向对象的方法维护程序，大大提高了软件维护的效率。

现在设计一个大的软件，是严格按照面向对象软件工程的5个阶段进行的，这5个阶段的工作不是由一个人从头到尾完成的，而是由不同的人分别完成的。

如果所处理的是一个较简单的问题，可以不必严格按照以上5个阶段进行，往往由程序设计者按照面向对象的方法进行程序设计，包括类的设计(或选用已有的类)和程序的设计。

2.2 类的声明和对象的定义

2.2.1 类和对象的关系

在C++中对象的类型称为类(class)。类代表了某一批对象的共性和特征。

前面已说明：类是对象的抽象，而对象是类的具体实例(instance)。正如同结构体类型和结构体变量的关系一样，人们先声明一个结构体类型，然后用它去定义结构体变量。同一个结构体类型可以定义出多个不同的结构体变量。

类是抽象的，不占用内存，而对象是具体的，占用存储空间。

2.2.2 声明类

声明一个类：

```
class Student      //以class开头
{ int num;
char name[20];
char sex; //以上3行是数据成员
void display( ) //这是成员函数
{cout<<"num:"<<num<<endl;
cout<<"name:"<<name<<endl;
cout<<"sex:"<<sex<<endl;
//以上4行是函数中的操作语句
}
};
Student  stud1，stud2;
//定义了两个Student 类的对象stud1和stud2
```

可以看到声明类的方法是由声明结构体类型的方法发展而来的。

可以将上面类的声明改为

```
class Student    //声明类类型
{ private:  //声明以下部分为私有的
int num;
char name[20];
char sex;
public:  //声明以下部分为公用的
void display( )
{cout<<"num:"<<num<<endl;
cout<<"name:"<<name<<endl;
cout<<"sex:"<<sex<<endl; }
};
Student  stud1，stud2;
//定义了两个Student 类的对象
```

类是一种广义的数据类型。类这种数据类型中的数据既包含数据，也包含操作数据的函数。

一般是把数据隐蔽起来，而把成员函数作为对外界的接口。

如果在类的定义中既不指定private，也不指定public，则系统就默认为是私有的。

归纳以上对类类型的声明，可得到其一般形式如下：

class 类名

{ private:

私有的数据和成员函数;

public:

公用的数据和成员函数;

};

private和public称为成员访问限定符(member access specifier)。

还有一种成员访问限定符**protected**(受保护的)，用**protected**声明的成员称为受保护的成员，它不能被类外访问(这点与私有成员类似)，但可以被派生类的成员函数访问。

在一个类体中，关键字**private**和**public**可以分别出现多次。每个部分的有效范围到出现另一个访问限定符或类体结束时(最后一个右花括号)为止。但是为了使程序清晰，应该养成这样的习惯：使每一种成员访问限定符在类定义体中只出现一次。

现在的C++程序多数先写public部分，把private部分放在类体的后部。这样可以使用户将注意力集中在能被外界调用的成员上，使阅读者的思路更清晰一些。

在C++程序中，经常可以看到类。为了用户方便，常用的C++编译系统往往向用户提供类库(但不属于C++语言的组成部分)，内装常用的基本的类，供用户使用。不少用户也把自己或本单位经常用到的类放在一个专门的类库中，需要用时直接调用，这样就减少了程序设计的工作量。

2.2.3 定义对象的方法

2.2.2节的程序段中，最后一行用已声明的**Student**类来定义对象，这种方法是很容易理解的。经过定义后，**stud1**和**stud2**就成为具有**Student**类特征的对象。**stud1**和**stud2**这两个对象都分别包括**Student**类中定义的数据和函数。

定义对象可以有几种方法，见下页。

1. 先声明类类型，然后再定义对象

（1） **class** 类名 对象名

如 **class Student stud1,stud2;**

把**class**和**Student**合起来作为一个类名，用来定义对象。

（2） 类名 对象名

如 **Student stud1，stud2;**

直接用类名定义对象。这两种方法是等效的。

第**1**种方法是从**C**语言继承下来的，第**2**种方法是**C++**的特色，显然第**2**种方法更为简捷方便。

2. 在声明类类型的同时定义对象

```
class Student    //声明类类型
{ public:   //先声明公用部分
   void display( )
    {cout<<"num:"<<num<<endl;
cout<<"name:"<<name<<endl;
cout<<"sex:"<<sex<<endl; }
private:   //后声明私有部分
int num;
char name[20];
char sex;
}stud1，stud2; //定义了两个Student类的对象
```

在定义Student类的同时，定义了两个Student 类的对象。

3. 不出现类名，直接定义对象

```
class                          //无类名
{private:          //声明以下部分为私有的
 ┆
 public:        //声明以下部分为公用的
 ┆
}stud1，stud2; //定义了两个无类名的类对象
```

直接定义对象，很少用。

在实际的程序开发中，一般都采用上面3种方法中的第1种方法。在定义一个对象时，编译系统会为这个对象分配存储空间，以存放对象中的成员。

2.2.4 类和结构体类型的异同

C++允许用struct来定义一个类。将前面用关键字class声明的类类型改为用关键字struct：

```
struct Student  //用关键字struct来声明一个类类型
{private:       //声明以下部分为私有的
int num; //以下3行为数据成员
char name[20];
char sex;
public:     //声明以下部分为公用的
 void display( ) //成员函数
{cout<<"num:"<<num<<endl;
cout<<"name:"<<name<<endl;
cout<<"sex:"<<sex<<endl; }
};
Student  stud1，stud2;   //定义了两个Student类的对象
```

用struct声明的结构体类型实际上也就是类

用struct声明的类，如果对其成员不作private或public的声明，系统将其默认为public。如果想分别指定私有成员和公用成员，则应用private或public作显式声明。

而用class定义的类，如果不作private或public声明，系统将其成员默认为private，在需要时也可以自己用显式声明改变。

建议尽量使用class来建立类，写出完全体现C++风格的程序。

2.3 类的成员函数

2.3.1 成员函数的性质

类的成员函数(简称类函数)，它与一般函数的区别只是：它是属于一个类的成员，出现在类体中。它可以被指定为private(私有的)、public(公用的)或protected(受保护的)。

在使用类函数时，要注意调用它的权限(它能否被调用)以及它的作用域。

成员函数可以访问本类中任何成员(包括私有的和公用的)，可以引用在本作用域中有效的数据。

一般的做法是将需要被外界调用的成员函数指定为public，它们是类的对外接口。

有的函数是为本类中的成员函数所调用的，就应该将它们指定为private。这种函数的作用是支持其他函数的操作，是类中其他成员的工具函数(utility function)，类外用户不能调用这些私有的工具函数。

2.3.2 在类外定义成员函数

类体中只写成员函数的声明，而在类的外面进行函数定义

```
class Student
{ public:
void display( ); //公用成员函数原型声明
private:
int num;string name;
char sex; //以上3行是私有数据成员
};
void Student::display( ) //在类外定义display类函数
{cout<<"num:"<<num<<endl;          //函数体
cout<<"name:"<<name<<endl;
cout<<"sex:"<<sex<<endl;
}
Student stud1,stud2; //定义两个类对象
```

成员函数在类外定义时，必须在函数名前面加上类名，予以限定(qualifed)，“::”是作用域限定符(field qualifier)或称作用域运算符，用它声明函数是属于哪个类的。

类函数必须先在类体中作原型声明，然后在类外定义，也就是说类体的位置应在函数定义之前，否则编译时会出错。

如果一个函数，其函数体只有2~3行，一般可在声明类时在类体中定义。多于3行的函数，一般在类体内声明，在类外定义。

2.3.3 inline 成员函数

类的成员函数可以指定为内置函数（内联函数）。

为了减少时间开销，如果在类体中定义的成员函数中不包括循环等控制结构，C++系统会自动将它们作为内置(inline)函数来处理。即在程序调用这些成员函数时，并不是真正地执行函数的调用过程(如保留返回地址等处理)，而是把函数代码嵌入程序的调用点。这样可以大大减少调用成员函数的时间开销。

```
class Student
{public:
void display( )
{cout<<"num:"<<num<<endl;
cout<<"name:"<<name<<endl;
cout<<"sex:"<<sex<<endl;
}
private:
int num;
string name;
char sex;
};
```

对类内定义的成员函数，可以省略inline

```
class Student
{ public:
inline void display( ); //声明此成员函数为内置函数
private:
int num;
string name;
char sex;
};
inline void Student::display( )
// 在类外定义display函数为内置函数
{cout<<"num:"<<num<<endl;
cout<<"name:"<<name<<endl;
cout<<"sex:"<<sex<<endl;
}
```

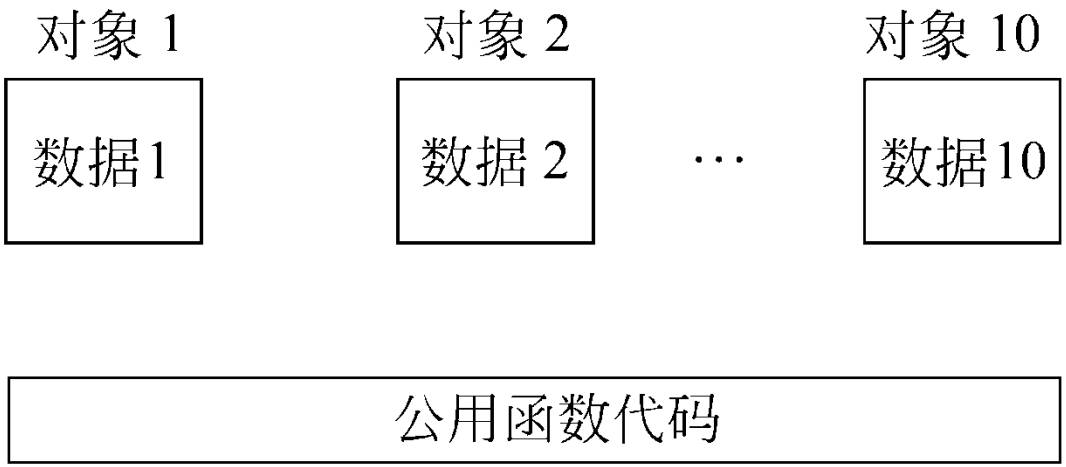
如果成员函数在类体外定义，想将这些成员函数指定为内置函数，应当用inline作显式声明。

如果在类体外定义inline函数，则必须将类定义和成员函数的定义都放在同一个头文件中(或者写在同一个源文件中)，否则编译时无法进行置换(将函数代码的拷贝嵌入到函数调用点)。

只有在类外定义的成员函数规模很小而调用频率较高时，才将此成员函数指定为内置函数。

2.3.4 成员函数的存储方式

C++编译系统中，每个对象所占用的存储空间只是该对象的数据部分所占用的存储空间，而不包括函数代码所占用的存储空间，从而节约了存储空间。



如果对同一个类定义了10个对象，这些对象的成员函数对应的是同一个函数代码段，而不是10个不同的函数代码段。

例：

```
class Time
{public:
int hour;
int minute;
int sec;
void set( )
{cin>>a>>b>>c;}
};

cout<<sizeof(Time)<<endl;
```

输出的值是12。

不包括函数代码所占用的存储空间

2.4 对象成员的引用

在程序中经常需要访问对象中的成员。访问对象中的成员可以有3种方法：

- 通过对象名和成员运算符访问对象中的成员；
- 通过指向对象的指针访问对象中的成员；
- 通过对象的引用变量访问对象中的成员。

2.4.1 通过对象名和成员运算符访问对象中的成员

例如在程序中可以写出以下语句：

stud1.num=1001; //假设num为公用的整型数据成员

表示将整数1001赋给对象stud1中的数据成员num。

不仅可以在类外引用对象的公用数据成员，而且还可以调用对象的公用成员函数，但同样必须指出对象名，如

stud1.display( ); //正确，调用对象stud1的公用成员函数

display( ); //错误，没有指明是哪一个对象的display函数

应该注意所访问的成员是公用的(public)还是私有的(private)。只能访问public成员，而不能访问private成员，如果已定义num为私有数据成员，下面的语句是错误的：

```
stud1.num=10101;
//num是私有数据成员，不能被外界引用
```

在类外只能调用公用的成员函数。在一个类中应当至少有一个公用的成员函数，作为对外的接口，否则就无法对对象进行任何操作。

2.4.2 通过指向对象的指针访问对象中的成员

```
class Time
{public:                    //数据成员是公用的
int hour;
int minute;
};
Time t,*p;                  //定义对象t和指针变量p
p=&t;                       //使p指向对象t
cout<<p->hour; //输出p指向的对象中的成员hour
```

在p指向t的前提下，**p->hour，(*p).hour和t.hour三者等价**。

2.4.3 通过对象的引用变量来访问对象中的成员

如果为一个对象定义了一个引用变量，它们是共占同一段存储单元的，实际上它们是同一个对象。

如果已声明了Time类，并有以下定义语句：

```
Time t1;                //定义对象t1
Time &t2=t1;
 //定义Time类引用变量t2，并使之初始化为t1
cout<<t2.hour;          //输出对象t1中的成员hour
```

由于t2与t1共占同一段存储单元(即t2是t1的别名)，因此t2.hour就是t1.hour。

类的实现：成员函数对数据成员的操作

2.5 类的封装性和信息隐蔽

2.5.1 公用接口与私有实现的分离

类的作用是把数据和算法封装在用户声明的抽象数据类型中。

在声明了一个类以后，用户主要是通过调用公用的成员函数来实现类提供的功能(例如对数据成员设置值，显示数据成员的值，对数据进行加工等)。因此，公用成员函数是类的对外接口。

用户可以调用公用成员函数来实现某些功能，而这些功能是在声明类时已指定的，用户可以使用它们而不应改变它们。实际上用户往往并不关心这些功能是如何实现的细节，而只需知道调用哪个函数会得到什么结果，能实现什么功能即可。

通过成员函数对数据成员进行操作称为类的实现。

为了防止用户任意修改公用成员函数，改变对数据进行的操作，往往不让用户看到公用成员函数的源代码，显然更不能修改它，用户只能接触到公用成员函数的目标代码。

类中被操作的数据是私有的，实现的细节对用户是隐蔽的，即所谓“私有实现”。形成了信息隐蔽。信息隐蔽是软件工程中一个非常重要的概念。它的好处在于：

(1) 如果想修改或扩充类的功能，只需修改本类中有关的数据成员和与它有关的成员函数，程序中类外的部分可以不必修改。

(2) 如果在编译时发现类中的数据读写有错，不必检查整个程序，只需检查本类中访问这些数据的少数成员函数。

2.5.2 类声明和成员函数定义的分离

在面向对象的程序开发中，一般做法是将类的声明(其中包含成员函数的声明)放在指定的头文件中，用户如果想用该类，只要把有关的头文件包含进来即可，不必在程序中重复书写类的声明。

由于在头文件中包含了类的声明，因此在程序中就可以用该类来定义对象,并可以调用这些对象的公用成员函数。

为了实现信息隐蔽，对类成员函数的定义一般不放在头文件中，而另外放在一个.cpp文件中。

例如，可以分别写两个文件：

```
//student.h (这是头文件，在此文件中进行类的声明)
class Student      //类声明
{ public:
void display( ); //公用成员函数原型声明
private:
int num;
char name[20];
char sex;
};
```

```
//student.cpp   //在此文件中进行函数的定义
#include <iostream>
#include "student.h"//不要漏写此行，否则编译通不过
void Student::display( ) //在类外定义display类函数
{cout<<"num:"<<num<<endl;
cout<<"name:"<<name<<endl;
cout<<"sex:"<<sex<<endl;
}
```

完整的源程序还应当有包括主函数的源文件：

```
//main.cpp    主函数模块
#include <iostream>
#include "student.h"  //将类声明头文件包含进来
int main( )
{Student stud; //定义对象
stud.display( ); //执行stud对象的display函数
return 0;
}
```

这是一个包括3个文件的程序，组成两个文件模块：一个是主模块main.cpp，一个是student.cpp。在主模块中又包含头文件student.h。在预编译时会将头文件student.h中的内容取代#include "student.h"行。

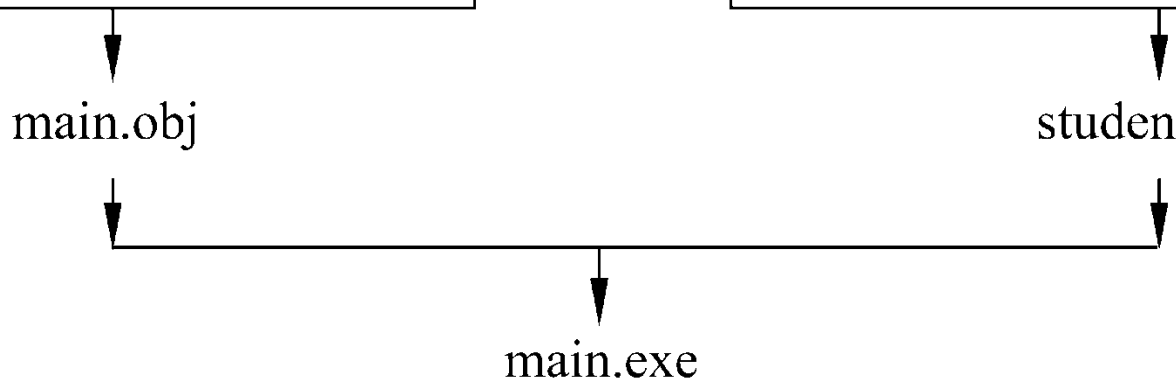
请注意： 由于将头文件student.h放在用户当前目录中，因此 "student.h"不用尖括号<student.h>，否则编译时会找不到此文件。

主模块 main.cpp

```
#include <iostream>
#include "student.h"
void main( )
{
   …
}
```

成员函数定义文件 student.cpp

```
#include <iostream>
#include "student.h"
void Student: :display( )
{
   …
}
```



2.5.3 面向对象程序设计中的几个名词

类的成员函数——称为“方法”(method)，“方法”是指对数据的操作，只有被声明为公用的方法（成员函数）才能被对象在外界所激活。

外界是通过发“消息”来激活有关方法的。所谓“消息”，其实就是一个命令，由程序语句来实现。

stud是对象

display( )是方法

语句“stud.display( );”是消息

2.6 类和对象的简单应用举例

例2.1 最简单的例子。

```
#include <iostream>
using namespace std;
class Time //定义Time类
{public: //数据成员为公用的
 int hour;
 int minute;
 int sec;
};
```

```
int main( )
{ Time t1; //定义t1为Time类对象
cin>>t1.hour; //输入设定的时间
cin>>t1.minute;
cin>>t1.sec;
cout<<t1.hour<<":"<<t1.minute<<":"<
<t1.sec<<endl;//输出时间
return 0;
}
```

例2.2 引用多个对象的成员。

（1） 程序(a)

```
#include <iostream>
using namespace std;
class Time
{public:
int hour;
int minute;
int sec;
};//同前面一样
```

可以使用函数来进行输入和输出。见程序(b)。

```
int main( )
{Time t1; //定义对象t1
cin>>t1.hour; //向t1的数据成员输入数据
cin>>t1.minute;
cin>>t1.sec;
cout<<t1.hour<<":"<<t1.minute<<":"<<t1.sec<<endl;  //输出t1中数据成员的值
Time t2; //定义对象t2
cin>>t2.hour; //向t2的数据成员输入数据
cin>>t2.minute;
cin>>t2.sec;
cout<<t2.hour<<":"<<t2.minute<<":"<<t2.sec<<endl;  //输出t2中数据成员的值
return 0;
}
```

（2） 程序(b)

```
int main( )
{
void set_time(Time&);     //函数声明
void show_time(Time&);     //函数声明
Time t1; //定义t1为Time类对象
set_time(t1); //调用set_time函数，向t1对象中的数据成员输入数据
show_time(t1); //调用show_time函数，输出t1对象中的数据
Time t2; //定义t2为Time类对象
set_time(t2); //调用set_time函数，向t2对象中的数据成员输入数据
show_time(t2); //调用show_time函数，输出t2对象中的数据
return 0;
}
```

```
void set_time(Time& t) //定义函数set_time，形参t是引用变量
{ cin>>t.hour; //输入设定的时间
 cin>>t.minute;
 cin>>t.sec;
}
void show_time(Time& t) //定义函数show_time，形参t是引用变量
{cout<<t.hour<<":"<<t.minute<<":"<<t.sec<<endl; //输出对象中的数据
}
```

例2.3 将例2.2的程序改用含成员函数的类来处理。

```
#in
usi
cla
{pu
voi
voi
 pr
int
int
int
};
```

```
int main( )
{Time t1;                //定义对象t1
t1.set_time( );          //调用对象t1的成员函数set_time，向t1的数据成员输入数据
t1.show_time( );         //调用对象t1的成员函数show_time，输出t1的数据成员的值
Time t2;                 //定义对象t2
t2.set_time( );          //调用对象t2的成员函数set_time，向t2的数据成员输入数据
t2.show_time( );         //调用对象t2的成员函数show_time，输出t2的数据成员的值
return 0;
}
```

```
void Time::set_time( )   //在类外定义set_time函数
{
 cin>>hour;
 cin>>minute;
 cin>>sec;
}

void Time::show_time( ) //在类外定义show_time函数
{
 cout<<hour<<":"<<minute<<":"<<sec<<endl;
}
```

注意：

(1) 在主函数中调用两个成员函数时，应指明对象名t1,t2。

(2) 在类外定义函数时，应指明函数的作用域(如void Time∶∶set_time( ))。

(3) 应注意区分什么场合用域运算符“∶∶”，什么场合用成员运算符“.”，不要搞混。

作业：

每两周交一次，提交至ftp://172.21.85.11 ,用户名:student,密码:11111111，提交要求请登录FTP查看。

P67：

第2，3，5，6题。