第 三 章　　类和对象

类是面向对象程序设计的核心，它实际上是由用户定义的一种新的复杂数据类型。

类是通过抽象数据类型ADT方法来实现的一种(数据)类型，它是对某一类对象的抽象，而对象是某种类的一个实例，因此类和对象是密切相关的。

类是将不同类型的数据和与这些数据相关的操作封装在一起的集合体。因此它具有更高的抽象，性实现了信息的隐藏和封装（Encapsulation）。

我们生活在科学技术飞速发展的现代化社会，需要了解和处理的事情太多、太复杂。为了减少必须了解处理的事情，我们是在某一程度的细节中生活的，也就是说，为了便于达到某种目的，在某种程度上必须要隐藏一些与此目的无关紧要的具体细节。例如在现实生活中人们通常把电视看成一种娱乐和获取信息的工具来使用，不会考虑它的内部结构和工作原理，即将“制造电视机”和“使用电视机”区别开来。“制造电视”需要具备电视制造技术的专门人才来完成。

在面向对象的计算机世界中，这种细节程度的数据隐藏就叫抽象。在OOP中也有分工明确的两种编程。一种是标准类库的设计，这是由搞面向对象语言系统的软件专门人员来完成。另一种是面向对象应用程序设计，它是由从事计算机应用的各类专业人员完成。

§3.1 类的定义

1.类的定义格式：类的定义具有如下标准化格式：

```
class    类名        ← 类头
{
      private:
        数据成员或成员函数的说明
      protected:
        数据成员或成员函数的说明
      public:
        数据成员或成员函数的说明
};
  <各成员函数的实现(定义部分)>
```

类体

说明：(1)类的定义格式分为说明部分和实现部分。说明部分是用来描述该类中的成员，包括数据成员的说明和成员函数的说明，成员函数是用来对数据成员进行操作的，又称为“方法”。实现部分是用来定义各种成员函数的，以描述这些成员函数是如何实现对数据成员的操作。总之说明部分将告诉编程者这个类是“干什么”的，而实现部分是告知“怎么干”的。

(2)类的说明部分由类头和类体两部分组成，类头有关键字class和类名组成，类名是由编程者启用的一种标识符，有些软件开发商如Microsoft公司，用‘C’(Class)开头的字符串作为类名，也有的用大写字母‘T’（Type）开头的字符串作为类名，以便与对象名、函数名区别。开始的字符串作为类名，以便与对象名、函数名区别。类头是用来向编译系统声明定义了一个新的类型，而类体是对类的组织形式进行具体的描述。由访问限制符（private、protected、public）和数据成员、成员函数组成，整个类体用一对大括号包围起来，完整地表达对类的描述。

```
class Date {    //tdate.h
public:         //三个公有成员函数。
    void SetDate(int y = 2000,
                 int m = 1, int d = 1);
         int IsLeapYear( );
         void PrintDate( );
private:
         int year, month, day;
    //三个私有数据成员。
 };
```

日期类Date的实现部分为：

```
void Date::SetDate(int y, int m,
                   int d)
{  year = y;  month = m;    day = d;    }
int Date::IsLeapYear( )//判断该年是否为闰年。
{ return (year%4 == 0 && year%100 != 0)
          || (year % 400 == 0);        }
void Date::PrintDate( )  //显示年、月、日的具体值。
{   cout << year << "," << month << ","
         << day << endl;
}
```


例如，用类定义一个关于日期的抽象数据类型ADT（Abstract Data Type）。日期类取类名Date，能设置年、月、日的具体值，并能判断该年是否为闰年，还能显示年、月、日的具体值。日期类Date的说明部分为：

(3)类由数据和函数组成，称为类的成员，因此，类有两种成员：数据成员和成员函数。成员函数又称为“方法”（mathod）。本例中Date类共有6个成员，其中，三个私有数据成员，三个公有成员函数。

(4)通常习惯上将类定义的说明部分或者整个定义部分（包含实现部分）放到一个头文件中。例如，可将前面定义的类Date放到一个名为tdate.h的头文件中。在需要引用Date类的源程序的开头处写上：

```
#include "tdate.h"
```

则tdate.h的全部内容将嵌入到这条语句的位置上。


```
#include <iostream>
using namespace std;
#include "tdate.h"
void main( )
{ Date date1, date2;
//定义日期类Date的两个对象date1和date2。
  date1.SetDate(2000, 5, 4);
  //给对象date1设置年、月、日的具体值。
  date2.SetDate(2000, 4, 9);
  //给对象date2设置年、月、日的具体值。
  int leap = date1.IsLeapYear( );
  //判断对象date1的年份是否为闰年。
     cout << "LEAP = " << leap << endl;
  //输出判断结果。
     date1.PrintDate( );
  //显示对象date1年、月、日的具体值。
  date2.PrintDate( ); //显示对象date2年月日
}
```


(5) class允许隐藏内部成员。它是依靠类定义中的三个访问限制符public、private、protected来确定隐藏的程度，它们将类体划分成三大部分。图11.1形象地描绘了这三部分的区别。

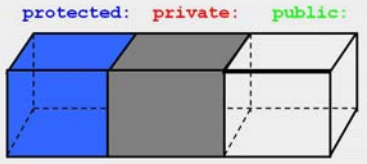

图3.1 三种访问限制符的区别

以public:开头的程序部分称为公有部分。
以private:开头的程序部分称为私有部分。
以protected:开头的程序部分称为保护部分。

(6)类的定义只是定义了某种类的组织形式，即定义了一个新的class类型，相当于ANSI C中的结构体定义。编译系统并不给class类型的每个数据成员分配内存空间。

2.访问限制符public、private、protected

它们将类体分为三个部分，每一部分都可以有数据成员和成员函数，也可以只有数据成员或成员函数，但不同的访问限制符规定了该部分所具有的访问权限。

(1)Public指定的为公有部分是透明的，它的数据成员和成员函数是开放的，既可以由本类的成员函数访问，也可有程序的其他部分直接访问。例如允许该类的对象去直接访问它的公有部分

即编写成“ 对象名.公有成员名 ”的形式。

(2)Private指定的为私有部分象一个黑盒子，完全是隐藏的，它只能由本类的成员函数访问，不允许程序其他部分访问，即在成员函数体内直呼其名地写出私有成员名，如在日期类Date的SetDate( )成员函数体内，可以直接写：

```
year = y;
month = m;
day = d;”
```

等语句。但是，不允许程序其他部分直接访问，例如不允许该类的对象去直接访问它的私有部分，即不允许编写成“对象名.私有成员”的形式

例如在main()函数内，用该类的对象去直接访问私有数据成员，即如下都是非法的：

```
date1.day = 6;
date1.month = 12;
date1.year = 1999;
cout << date1.day;
```

(3)protected指定的保护部分是半透明的，它可由本类成员函数或它的派生类成员函数直接访问，但也不允许程序其他部分直接访问它。保护部分主要用于类的继承，将在后面章节讨论。

(4)通常总是将数据成员指定为私有的以实现数据隐藏，这些数据成员是用来描述该类对象的属性，编程者无法直接访问它们而隐藏起来。

而将成员函数指定为公有的，作为该类对象访问私有数据成员的一个接口界面，即对象访问私有数据成员的一条消息通路提供给外界使用。因此私有数据只能通过公有成员函数访问，从而隐藏了处理这些数据的实现细节，使得类对数据的描述和类提供给外a部世界来处理数据的界面这两件事情互相独立，这就给出了面向对象的重要特性，使得一个类的用户唯一需要做的事情就是访问类的界面。正如图3.2 所示：

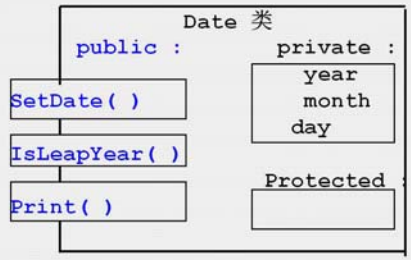

图3.2 Date类的描述

例程3.2用类定义一个计数器的抽象数据类型ADT(Abstract Data Type)。计数器允许取值范围为0~4294967295的正整数，可进行的操作是计数器加1、减1和读计数器的值。其类的定义如下：


```
#include <iostream>
using namespace std;
class Counter{
public :
  Counter()     //Counter类的构造函数。
   { value = 0;//初始化数据成员值。
cout << "Constructor called !\n";
//输出显示调用了本构造函数的信息。
  }
  void Increment( )
  //在允许取值范围内计数器加1。
   { if(value < 4294967295) value++; }
   void Decrement( )
  //当计数器的数据成员值不为0时减1。
   { if(value > 0) value--; }
  unsigned ReadValue( )//读计数器的值
  {  return value; }
  ~Counter() //Counter类的析构函数。
  { cout << "Destructor called !\n"; }
  //只输出调用了本析构函数的信息，其它什么也不做。
  private :
   unsigned value;
  //Counter类的私有数据成员。
 };
```



```
void main()  //Counter类的测试程序
{
   Counter c1 ,c2;      //定义了Counter类的两个对象c1和c2。

   for(int i = 1; i <= 6; i++)
   {
      c1.Increment( );//在第1个for语句的循环体内6次向对象c1发送增量运算的消息
      cout << "value of c1 = "<< c1.ReadValue( )<< endl;
                        //每循环一次，显示对象c1的计数值。
      c2.Increment( );   //同样6次向对象c2发送增量运算的消息。
   }
   cout << "After loop ,value of c2 = "<< c2.ReadValue()<< endl;
                        //循环完成后，显示对象c2的计数值。
   for(i = 1; i <= 5; i++)
      c2.Decrement( ); //在第2个for语句的循环体内5次向对象c2发送减量运算的消息

   cout << "Final value of c1 = "
        << c1.ReadValue( )
        << ",value of c2 = "
        << c2.ReadValue( ) << endl; //显示对象c1和c2最终的计数值。
}
```


该程序的输出结果：

```
Constructor called !
Constructor called !
value of c1 = 1
value of c1 = 2
value of c1 = 3
value of c1 = 4
value of c1 = 5
value of c1 = 6
After loop ,value of c2 = 6
Final value of c1 = 6,value of c2 = 1
Destructor called !
Destructor called !
```

C++ 程序设计

3. 数据成员：如前所述，类的定义与结构体的定义一样，仅仅定义了类的组织形式，给出了一个新的class类型，编译系统并不给类的每个数据成员分配具体的内存空间。数据成员既可以放在公有部分称为公有数据成员，又可以放在私有或保护部分称为私有或保护数据成员。它们的定义格式都是：

```
<类型>    数据成员名;
```

(1) 数据成员只有<类型>说明，而无<存储类>说明，所以将类的数据成员说明为auto、register和extern型都是无意义的。

C++ 程序设计

(2) 在类体中不允许对所定义的数据成员进行初始化。例如，下面的定义都是错误的：

```
class Counter{
public :   ...
private :
     unsigned value = 0;   //error
};
class Date {
public :
        ...
private :
   int year(1999),month(9),day(16);
                              //error
};
```


C++ 程序设计

(3) 数据成员的类型可以是基本数据类型(char、int、...、double等)和复杂数据类型(数组、指针、引用、...、结构变量等)，或已经定义了的类对象。当另一个类的对象作为这个类的成员时，称为“对象成员”，这另一个类必须预先定义好。例如：(教材p352，参见p52exg.cpp)

```
class N  {
public :
        ...
};//不能采用先声明，后定义的形式。”class N;”
class M {
public :
         ...
private :
   N  n;        // n是N类的对象
};
```


C++ 程序设计

4.成员函数(Member Function)：类体(即紧跟类头后一对大括号所包围的程序部分)中不仅定义了该类的对象所具有的数据结构，还定义了对该类对象进行操作的成员函数(方法)，其定义格式如下：

```
<返回类型> [<类名> :: ]成员函数名
函数头    (类型  参数1, ...,类型  参数n)
{      ...      ...      ...      }
          函数体
```

(1) 所有成员函数都必须在类体内用函数原型加以声明。而成员函数的定义(方法的实现部分)则较灵活，既可在类体外定义，也可在类体内定义在类体外定义成员函数时，必须用类名加作用域运算符’::’来指明所定义的成员函数属于哪个类；在类体内定义的成员函数，其函数头可不写

[<类名> ::]。

(2)在类体内定义成员函数时，编译系统自动地将成员函数当作内联函数处理，其函数头前不必写关键字”inline”。例如Counter类中的Increment()、Decrement()和 ReadValue()都是内联函数。若是类体外定义的成员函数，必须用关键字”inline”指明它为内联函数。

(3)在定义了类的对象后，才可以用如下格式调用
(公有)成员函数：对象名.成员函数名(实参表)；
例如：
```
Counter c1, c2;
c1.Increment( );
c2. Decrement( );
```

(4)成员函数与普通函数一样，可设置参数的默认值(缺省值Default)。

§3.2 对象的定义：

定义一个类就相当于创建了一个新的class类型。要使用类，还必须用已经定义的类去说明它的实例变量（对象）。在C++中，class类型一旦被定义，它的实例变量（对象）就能被创建、初始化、并定义指针变量指向它。

在程序中，该类名就可以象基本数据类型‘int’；、...、‘double’等一样使用，用来定义具体的实例变量（对象）。例如：

```
Date date1 ,date2 , date[31] ,*pdate;
Counter c1 ,c2 ,*pc;
```


1.对象的定义格式：在定义好称之为“类名”的一个类以后，就可以用如下格式定义它的对象：

```
<存储类>  <类名>  <对象名表>;
```

<对象名表>中可以有一个或多个对象名，若为多个对象名时用逗号分隔开。对象名可以是一般对象名（如上例中Date、Counter为类名，date1、date2为Date类的两个一般对象名，
c1、c2为Counter类的一般对象），也可以是对象数组名（如date[31]是由31个Date类的对象有序排列成date[0]、date[1]、...、date[30]等组成的集合，取对象数组名为date），还可以是指向该类对象的指针名或对该类对象的引用名。
例如：

```
Date date1 ,& Rdate = date1;
Counter c1 ,&Rc = c1;
```


2.指向对象的指针（简称“对象指针”）：
(1)对象指针的定义格式：

```
<存储类>  <类名>  指针名;
```

例如：
```
Counter c1 ,c2 ,*pc;
```

这里所说的<存储类>是指针变量本身的存储类型，与ANSI C标准一样，有extern型、static型、auto型等。而auto的说明经常缺省，extern在定义时不写，声明时必须写。顺便指出，例中只定义了一个对象指针pc，但pc并没有定向，还必须通过初始化或赋值操作将已存在的对象地址赋给它。即pc = &c1; 另外，在C++中对象指针可作为另一个类的成员，此时的对象指针是没有<存储类>说明的。

(2) 只要一个类已经声明（可以没有定义），则指向该类的对象指针或该类对象的引用都可作为另一个类的成员。(而本身却是不可以的)


```
#include <iostream>
using namespace std;
class M;
class N {
private :
   int id;
public :
   M * pm;
   N(int i) { id = i; }
   int GetName( ) { return id; }
};
class M {
public :
   int value;
   M(int v) { value = v; }
   M( ) { value = 0; }
   int GetValue( )  { return value; }
};
```



```
void main( )
{  M m1(6) , m2(8);   N n(4);
   n.pm = &m2;
   cout << "value of m1 = "
        << m1.GetValue( ) << endl;
   cout<<"value pointed by member pm of
       n = "<< n.pm -> value << endl;
   cout << "id of n = "
        << n.GetName( ) << endl;
}
```

程序的输出结果：

```
value of m1 = 6
value pointed by member pm of n = 8
id of n = 4
```


3.访问类对象成员的方法：如前所述，类有两种成员，即数据成员和成员函数。每个对象都是类的一个实例，都具有自己的数据成员值，而该类所有对象的数据结构及其访问权限却是由它的类统一定义的。因此同一个类所创建的对象，其数据结构都是相同的，类中的成员函数是共享的，而不同对象的数据成员值可以是不相同的。例如，对已经定义的Counter类可用如下语句定义对象和对象指针：

```
Counter c1 ,c2 ,*pc;
```

对象c1、c2都具有各自的value值，但value的数据类型(unsigned)和访问权限(private)则由Counter类统一定义。由于它是私有数据成员，程序的其他部分不能直接访问它，必须通过公有部分的成员函数。

```
c1.value;  //error 不能直接访问私有数据成员
c1.ReadValue( );//用公有成员函数访问私有数据成员
```

(1) 数据成员的访问方法：如上所述，对于私有数据成员只能通过公有成员函数才能访问它。而公有数据成员可直接访问它，其格式为：

对象名.公有数据成员名
或　对象指针名 -> 公有数据成员名

其中<对象指针名>应与对象属于相同的class类型，且必须通过初始化或赋值操作将已存在的对象地址赋给它。当然也可以通过成员函数访问它。


```
class Point {
public :       //四个公有成员函数。
void SetPoint(int a , int b);
//设定点的X、Y坐标值。
int ReadX( ) { return x; }
//读取点的X坐标值。
int ReadY( ) { return y; }
//读取点的Y坐标值。
void Move(int xOffset ,
          int yOffset);
//计算点移动后的X、Y坐标值。
    int x , y; //两个公有数据成员x和y。
};
void Point::SetPoint(int a , int b)
{     x = a;          y = b;      }
void Point::Move(int xOffset ,
                 int yOffset)
{  x += xOffset;    y += yOffset;  }
```



```
例3.5  编写Point类的测试程序。
#include <iostream>
#include "tpoint.h"
using namespace std;
void main( )
{ Point p1 , p2;   //定义Point类的两个对象p1和p2。
   Point &ref = p1 , * ptr = &p2;      //还定义了一个对对象p1的引用
ref和指向对象p2的对象指针ptr。
        p1.x = 3; p1.y = 5;
   //可用对象名直接访问公有数据成员，给p1
      点设定X、Y坐标值。
   p2.SetPoint(8 , 10);   //也可用公有成员函数SetPoint( )给p1点设定
X、Y坐标值。
   p1.Move(2 , 1);     //调用公有成员函数Move( )，移动p1点。
   p2.Move(1 , -2);    //调用公有成员函数Move( )，移动p2点。
   cout << "x1 = " << ref.x << ","
        << "y1 = " << ref.y << endl;    //使用对象引用ref代替p1直
接访问它的公有数据成员x和y。
   cout << "x2 = " << ptr -> x << ","
        << "y2 = " << ptr -> y << endl;    //使用对象指针ptr代替
p2直接访问它的公有数据成员x和y。
}
```


该程序的输出结果：

```
x1 = 5,y1 = 6
x2 = 9,y2 = 8
```

(2)成员函数的访问方法：访问成员函数的格式：

对象名．公有成员函数名（参数表）；

或 对象指针名 -> 公有成员函数名（参数表）；

其中<对象指针名>应与对象属于相同的class类型，且必须通过定向操作指向了已存在的对象。

(3)通过引用对象访问成员的方法。引用对象可用来代替被引用的对象名访问成员，其格式：

引用对象名．公有数据成员名

或 引用对象名．公有成员函数名（参数表）

§3.3对象的初始化：

构造函数和析构函数是在类体中说明的两种特殊的成员函数。

1.构造函数：C++中“类”只定义了一组对象的类型。要使用这个类还必须用“类”说明它的对象，每个对象中的数据成员还必须赋初值，这些功能都是由构造函数完成的。

(1) 构造函数是一种特殊的成员函数，其定义格式和其他成员函数相同，只是它以类名作为函数名，例如Counter类的构造函数为Counter( )，且不能指定返回类型和显式的返回值，连void都不能写，即无返回类型。若某类定义了构造函数，则每当该类的对象创建时，编译系统为它的对象分配内存空间，并自动调用构造函数初始化所创建的对象，从而确保每个对象自动地初始化。例如：

```
Counter c1;
```

编译系统为对象c1的每个数据成员(value)分配内存空间，并调用构造函数Counter( )自动地初始化对象c1的value值设置为0

(2)构造函数只完成新对象的初始化工作，不执行对象的主要任务。即在类创建新对象时，为对象分配内存空间，给数据成员赋初值。

(3)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，即构造函数可重载。


```
#include <iostream>
using namespace std;
class Complex {
public:
  Complex(double real,
          double imag);
  //Constructor(1)，一般构造函数。
    Complex(void);
  //Constructor(2)，无参数构造函数。
  Complex(const Complex & c);
  //Constructor(3)，复制构造函数
  Complex(double real);
  //Constructor(4)，类型转换构造函数。
    void print(int i);
private :
  double  real, imag;
};
```

```
//一般构造函数，用两个实数分别给新创建对象的
  两个数据成员赋初值。
Complex::Complex(double r , double i)
{  real = r;      imag = i;
 cout<<"Constructor(1)called:real="
 << real << " ,imag = " << imag << endl;
 //输出被调用的提示信息。
}
//无参数构造函数，创建新对象并将它的数据成员
  都设定成0。
Complex::Complex(void)
{ real = 0.0;      imag = 0.0;
cout<<"Constructor(2)called:real = "
<< real << " ,imag = " << imag << endl;
//输出被调用的提示信息。
}
```



```
/* 复制构造函数，把已存在对象的每个数据成员
   都对应赋值给新创建对象的数据成员，即对新
   创建对象进行“位模式拷贝”。*/
Complex::Complex(const Complex & c)
{ real = c.real;      imag = c.imag;
  cout<<"Constructor(3)called:real = "
 << real << " ,imag = " << imag << endl;
 //输出被调用的提示信息。
}
//类型转换构造函数，把一个实数转换成虚数部分
  为零的复数。
Complex::Complex(double r)
{ real = r;      imag = 0.0;
  cout<<"Constructor(4)called:real = "
  << real << " ,imag = " << imag << endl;
  //输出被调用的提示信息。
}
```



```
//输出显示复数类对象的值，形参i用来表示对
  象的序号。
void Complex::print(int i)
{ if(imag < 0)
     cout << "c" << i << " = " << real
          << imag << "i" << endl;
  else
       cout << "c" << i << " = " << real
          << "+" << imag << "i" << endl;
}
```



```
void main( )
{ Complex c1 ,c2(6 ,8), c3(5.6 ,7.9),\
            c4 = c2;

  c1.print(1);   //输出显示复数类对象c1。
  c2.print(2);   //输出显示复数类对象c2。
  c3.print(3);   //输出显示复数类对象c3。
  c4.print(4);   //输出显示复数类对象c4。
  c1 = Complex(1.2 ,3.4);
  //直接调用构造函数。
  c3 = c2;
  //把复数类对象c2的每个数据成员值赋给c3。
  c2 = 5;
  //调用类型转换构造函数，把实数转换成复数。
  c1.print(1);   //输出显示复数类对象c1。
  c2.print(2);   //输出显示复数类对象c2。
  c3.print(3);   //输出显示复数类对象c3。
  c4.print(4);   //输出显示复数类对象c4。
}
```

C++ 程序设计

该程序输出结果：

```
Constructor(2)called:real = 0,imag= 0 //创建c1，调用无参数构造函数。
Constructor(1)called:real = 6,imag= 8 //创建c2，调用一般构造函数。
Constructor(1)called : real = 5.6,
imag=7.9 //创建c3，调用一般构造函数。
Constructor(3) called : real = 6 ,
imag = 8 //创建c4，调用复制构造函数。
c1 = 0+0i
//执行“c1.print(1);”语句。
c2 = 6+8i
//执行“c2.print(2);”语句。
c3 = 5.6+7.9i
//执行“c3.print(3);”语句。
c4 = 6+8i
//执行“c4.print(4);”语句。
```


C++ 程序设计

```
Constructor(1) : real = 1.2 ,imag = 3.4        //执行“c1 = Complex(1.2 ,3.4);”。
Constructor(4) called : real = 5 ,imag = 0 //执行“c2 = 5;”。
c1 = 1.2+3.4i
c2 = 5+0i
c3 = 6+8i
c4 = 6+8i
```


C++ 程序设计

(4)通常在程序中不直接调用构造函数，可以在创建对象时显式地调用构造函数？

例如，

```
Date today = Date(1998,4,9);
Counter c1 = Counter( );
```

将today定义为类Date的对象（例子程序理解）

类名 对象名 = 构造函数名（实参表）；

由于“构造函数名”就是“类名”，所以可用更简单的形式：

类名 对象名（实参表）；

此时是在创建对象时，隐式地（自动地）调用构造函数。

C++ 程序设计

2．析构函数：它是构造函数的配对物，与构造函数一样是与类同名的成员函数，并在函数名前加上一个’~’以便与构造函数相区别。例如：count.cpp中的析构函数~Counter( )。

(1)一个类只能有一个析构函数，且析构函数没有参数和返回值，它实现与构造函数相反的功能。在删除对象时执行一些清理任务，例如释放对象所占用的内存空间，有些对象可能还需做其他的事情，如当撤消一个显示窗口对象时，还需将它们从屏幕上清除掉。

(2)对于用new创建的(动态)对象，只有用delete撤消时编译系统才调用析构函数。

(3)如果编程者没有给类定义析构函数，那么编译系统将生成一个什么也不做的缺省析构函数。例如，Complex类的缺省析构函数为：

```
Complex::~Complex( ){   }    //空函数
```

有人会问为什么要把缺省析构函数设计成什么都不做(对编程者来说)呢？因为只有这样对于各种存储类的对象，才能够按照它本来的创建和撤消管理机制运行。例如，自动对象在程序运行期间，每当遇到它的声明时就创建它，当离开它的作用域时就自动撤消它。这将在§3.6对象的存储类中详细讲述。

(4)编译系统自动调用构造函数的次序和调用析构函数的次序是相反的。为了验证这一点，给Counter类新增加一个私有数据成员int who;用来记录所生成对象的序号，变为：


```
#include <iostream>
using namespace std;
class Counter{
public :
  Counter(int id)
     { value = 0;   who = id;
       cout << "Object " << who
          <<" initialized !\n"; }
            ...
  ~Counter( )
     { cout << "Object " << who
          << " destroyed !\n"; }
private :
     unsigned value;
  int who;
};
```



```
void main( )
{     Counter c1(1) ,c2(2);
     cout << "OK !\n";
}
```

其输出结果为：

```
Object 1 initialized !
Object 2 initialized !
OK !
Object 2 destroyed !
Object 1 destroyed !
```


3.构造函数的种类：

构造函数可带有各种类型的参数，根据参数所具有的类型可将它分为如下四种：

(1)无参数的构造函数

(2)拷贝初始化构造函数

(3)类型转换构造函数

(4)一般构造函数

C++ 程序设计

(1) 无参数的构造函数：

当程序中写有这样的语句定义新对象时，如：

```
void main( )
{  Complex c1;        ...        }
```

则在Complex类中必须定义这样一个构造函数，

```
Complex::Complex(void)
{ real = 0.0;  imag = 0.0  }
```

显然它是不带任何参数的构造函数称为无参数的构造函数，当它的函数体为空时称为默认构造函数或缺省构造函数(Default Constructor Function)，即什么都不做的无参构造函数称为默认构造函数或缺省构造函数。

C++ 程序设计

①若某个类一个构造函数都未提供，则C++提供一个什么也不做的默认构造函数来创建对象，而不做任何初始化工作。例3.7

```
#include <iostream>
using namespace std;
 class Point {
 public : //无构造函数
    //默认构造函数Point( ) {  空  }
    void SetPoint(int a , int b);
int ReadX( ) { return x; }
int ReadY( ) { return y; }
void Move(int xOffset ,
              int yOffset);
private :
    int x , y;    //两个私有数据成员x和y 。
};
```


C++ 程序设计

```
//设定点的X、Y坐标值。
void Point::SetPoint(int a , int b)
{     x = a;        y = b;        }
//计算点移动后的X、Y坐标值。

void Point::Move(int xOffset ,
                      int yOffset)
{     x += xOffset;  y += yOffset;  }
```


C++ 程序设计

```
void main( )
{     Point p1 ,p2;
  //调用默认构造函数，只创建新对象但对它们不赋初值。
   cout << "x1 = " <<p1.ReadX( ) << " ,"
        << "y1 = " << p1.ReadY( ) << endl;
  cout << "x2 = " <<p2.ReadX( ) << " ,"
        << "y2 = " << p2.ReadY( ) << endl;
  p1.SetPoint(3 ,5);     //给对象p1设定点的X、Y坐标值。
  p2.SetPoint(8 ,10);    //给对象p2设定点的X、Y坐标值。
     p1.Move(2 ,1);      //移动p1点。
  p2.Move(1 ,-2);    //移动p2点。
  cout << "x1 = " << p1.ReadX( ) << " ,"
        << "y1 = " << p1.ReadY( ) << endl;
  cout << "x2 = " << p2.ReadX( ) << " ,"
        << "y2 = " << p2.ReadY( ) << endl;
}
```

该程序的输出结果： 随机数

```
x1 = -858993460 ,y1 = -858993460
x2 = -858993460 ,y2 = -858993460
x1 = 5 ,y1 = 6
x2 = 9 ,y2 = 8
```


②只要某个类定义了一个构造函数(不一定是无参数的构造函数)，C++不再提供默认构造函数。编程者需要何种类型的构造函数，都必须自行定义。

③与变量定义类似，在用默认构造函数创建对象时，若创建的是全局对象或静态对象，则对象的所有数据成员都初始化为零(数)或空(字符串)。例如若将p52x1.cpp作如下修改：

```
Point p1,p2; → static Point p1,p2;
```

程序的输出结果变为：

```
x1 = 0 ,y1 = 0
x2 = 0 ,y2 = 0
x1 = 5 ,y1 = 6
x2 = 9 ,y2 = 8
```