第 5 章

类 和 对 象

5.1 本章简介

5.1.1 软件开发方法

1．面向过程

面向过程的程序设计主要是将程序设计的工作围绕设计解题过程来进行，使用传统的过程设计语言。程序采用模块化结构，基于功能分解，程序的功能通过程序模块之间的相互调用完成。采用自顶向下逐步求精（逐步抽象）方法。不足之处在于：数据与操作的描述分离；数据缺乏保护；不能适应需求的改变。由于功能分解模型较难与现实世界的实际系统相吻合，开发出的软件系统难以适应需求的变化。可维护性差。

2．面向对象

面向对象的程序设计主要是把求解问题中的事物看作不同的对象，每个对象由一些数据和对这些数据所实施的操作构成，在 C++中，对数据的操作是通过函数来实现的。面向对象程序设计强调的是数据抽象，一方面加强了数据保护，另一方面实现了对现实世界活动的直接模拟，能较好地适应需求的变化，实现了数据及其操作的封装，稳定性好，当系统的功能需求发生变化时不会引起软件结构的整体变化。

面向对象继承机制可以大大提高软件的可重用性，便于实现功能的扩充、修改、增加或删除。降低软件的调试、维护难度，而且特别适合于需要多人合作的大型软件的开发。

5.1.2 面向对象方法的由来和发展

随着软件开发规模的不断扩大和开发方式的变化，程序设计开始被人们作为一门科学来对待，经过多年研究，在计算机科学中发展了许多程序设计的方法。下面通过回顾计算机语言的发展过程，来了解一下面向对象的方法是如何产生的。

早期的程序设计都是用机器语言或汇编语言编写的。这种程序的设计相当麻烦，严重影响了计算机的普及应用。随着计算机的应用日益广泛，发展了一系列不同风格的、为不同对象服务的程序设计语言。

最早的高级语言是在 20 世纪 50 年代中期研制的 FORTRAN 语言，它在计算机语言发

展史上具有划时代的意义。该语言引进了许多现在仍然使用的程序设计概念，如变量、数组、分支、循环等。但在使用中也发现了一些不足，不同部分的相同变量名容易发生错误。20 世纪 50 年代后期，高级语言 Algol 在程序段内部对变量实施隔离。Algol60 提出了块结构的思想，实际上也是一种初级的封装。在 20 世纪 60 年代开发的 Simula67，是面向对象的鼻祖，它将 Algol60 块结构的概念向前推进了一步，提出了对象的概念。20 世纪 70 年代出现的 Ada 语言是一种基于对象的语言，是支持数据抽象类型的重要语言之一，它丰富了面向对象的概念。到 20 世纪 80 年代中期以后，面向对象的程序设计语言广泛地应用于程序设计。

由于自 20 世纪 60 年代末到 70 年代初，出现了大型的软件系统，如操作系统、数据库等，由此给程序设计带来了新的问题。大型软件系统的研制需要花费大量的人力和物力，但当时编写出来的软件可靠性差，错误多，难以维护，已经到了程序员无法控制的地步，这就是“软件危机”。1969 年，E.W.Dijkstra 首先提出了结构化程序设计的概念，他强调了从程序结构和风格上研究程序设计，为“软件危机”起了很大的缓解作用。到 20 世纪 70 年代末结构化设计方法将软件划分成若干个可单独命名和编址的部分，它们被称为模块，模块化使软件能够有效地被管理和维护，能够有效地分解和处理复杂问题，但结构化的程序设计由于模块内数据共享及函数的调用关系错综复杂，导致软件维护变得非常困难。所以，20 世纪 80 年代，在软件开发各种概念和方法积累的基础上，就如何超越程序的复杂性障碍，如何在计算机系统中自然地表示客观世界等问题，提出了面向对象的设计方法，它不是以过程为中心，而是以对象代表的问题为中心环节，提出了“对象+对象+……=程序设计”理论，使人们对复杂系统的认识过程与系统的程序设计实现过程尽可能一致。

5.1.3　面向对象语言

20 世纪 80 年代中期以后，面向对象的程序设计语言广泛地应用于程序设计，并且有许多新的发展。归纳起来大致可分为两类：一类是纯面向对象的语言，如 Smalltalkhe、Eiffel 和 Java；另一类是混合型的面向对象语言，如 C++和 ObjectiveC。

C++语言的创作灵感来源于计算机语言多方面成果的凝聚，特别是 BCPL、C 语言和 Simula67，同时也借鉴 Algol68 语言。C++是一门高效实用的混合型程序设计语言，它最初的设计目标是：支持面向对象编程技术，支持抽象形态的类。C++语言包括两部分：一是 C++基础部分，它是以 C 语言为核心的；另一部分是 C++面向对象特性部分，是 C++对 C 语言的扩充部分。这样它既支持面向对象程序设计方法，又支持结构化程序设计方法。

C++的基础部分与 C 语言相比除了输入输出等一些细微的差别外，C++可以说是 C 语言的加强版，它保留了 C 语言功能强、效率高、风格简洁、适合大多数系统程序设计任务等优点，使得 C++与 C 之间取得了兼容性，因此，在过去的软件开发中积累的大量 C 的库函数和实用程序都可在 C++中应用。另外，C++语言通过对 C 的扩充，克服了原有 C 语言的缺点，完全支持面向对象程序设计方法。它支持类的概念。类是一种封装数据和对这些数据进行操作的用户自定义类型，类提供了数据隐蔽，确保了程序的稳定性、可靠性和可维护性。它还支持继承和多态性等特性，使得其代码具有高度的可重用性。面向对象程序设计有以下基本特点：

1．抽象

抽象是人类认识问题的最基本手段之一。面向对象方法中的抽象是对具体问题（对象）进行概括，抽出一类对象的公共性质并加以描述的过程。事实上，对问题进行抽象的过程，就是一个分析问题、认识问题的过程。

2．封装

利用封装的特性，在编写程序时，对于已有的成果，使用者不必了解具体的实现细节，而只需通过外部接口，依据特定的访问规则，就可以使用现有的资源。

3．继承

编程过程中任何问题都从头开始描述是不现实的，一个特定的问题，很可能前人已经进行过较为深入的探讨，这些结果怎么利用？还有在程序设计的后期，对问题又有了深入的体会，这些新的认识成果怎么加到已有的成果中？继承就是解决这些问题的良策，C++语言提供了类的继承机制，允许程序员在保持原有特性的基础上，进行更具体、更详细的类的定义与扩展。通过类的这种层次结构，就可以反映出对事物认识的发展过程。新的类由原有的类产生，继承或扩展了原有类的特征和功能，新的类称为原有类的派生类。关于继承和派生，本书会在以后的章节中进行详细的探讨。

4．多态

如果说继承讨论的是类与类的层次关系，那么多态则是考虑这种层次关系下类自身成员函数之间的关系问题，是解决功能和行为再抽象的问题。多态是指具有相似功能的不同函数使用同一个函数名称来实现的现象。这也是人类思维方式的一种直接模拟。

5.2 本章知识目标

本章需要读者在了解面向对象程序设计基本特点的基础上，理解类和对象的概念及它们之间的关系，掌握类和对象的定义和使用方法，掌握构造函数、拷贝功能的构造函数、析构函数的概念和使用方法。具体目标包括：

（1）掌握类的声明、类成员的定义方法，了解类成员的访问控制权限。

（2）掌握对象的定义和使用。

（3）重点掌握类的构造函数和析构函数的作用、定义和使用。

（4）了解对象数组和对象指针的定义和使用，掌握对象内存的动态分配。

（5）掌握对象作为函数参数的使用方法。

（6）了解友元的作用及使用友元的有关问题。

5.3 对象和类

5.3.1 对象和类的概念

（1）现实世界中的对象和类。

① 现实世界中的对象是认识世界的基本单元，世界就是由这些基本单元——对象组成的，如一个人、一辆车等。对象既可以很简单，也可以很复杂，复杂的对象可由若干个简单对象组成。对象是现实世界中的实体。

② 现实世界中的类是对一组具有共同属性和行为的对象的抽象。如笼统地说人是指的一个类，具体的某个人是人这个类的一个对象。类和对象是抽象和具体的关系。

（2）面向对象中的对象和类。

① 面向对象中的对象是由描述其属性的数据和定义在数据上的操作组成的实体，是数据单元和过程单元的集合体。如学生张三是一个对象，他由描述他的特征的数据和他能提供的操作来组成：

对象名：张三。

属性——年龄：20，性别：男，身高：173 厘米，体重：71 公斤，特长：篮球运动，专业：计算机科学和技术。

操作：回答有关自己的提问，吃饭、减肥、打篮球、计算机软件开发、维护、组网。

这里的属性说明了张三这个对象的特征，操作说明了张三能做什么。

② 面向对象中的类是一组对象的抽象，是将这组对象中相同的数据属性和操作行为抽象出来并加以描述和说明。类是创建对象的样板。

③ 对象和类的关系：对象是类的一个具体实例，有了类才能创建对象。当给类中的属性和行为赋予实际的值以后，就得到了类的一个对象。例如人这个类：

类名：人。

属性：年龄，性别，身高，体重，特长，专业。

操作：回答提问，对外服务、减肥、美白、增高、学习打篮球等。

对象：张三、李四、王五等。

特别要指出的是，在面向对象程序设计中，类为对象的创建提供样板。对象作为类的实例出现在内存运行的过程中，占有内存空间，是运行时存在的实体。类实际上是一个新的数据类型，使用它时要在源程序中说明，而说明部分在内存中是不分配存储空间的。对象通过类来定义，对象在内存中分配空间并完成相应操作。在 C++中，把描述类的属性的数据称为成员数据，把描述行为的操作称为成员函数。

5.3.2　类的确定和划分

（1）确定和划分类的重要性。

面向对象技术是将系统分解成若干对象，对象之间的相互作用构成了整个系统。类是创建对象的样板，当解决实际问题时，需要正确地进行分类。这是软件开发的第一步工作，划分的结果直接影响软件的质量。

（2）确定和划分类的一般原则。

类的确定和划分没有统一的方法，基本依赖设计人员的经验、技巧和对实际问题的理解与把握。一个基本的原则是：寻求系统中各事物的共性，将具有共性的那些事物划分成一个类；设计类应有明确的标准，设计的类应该是容易理解和使用的。同一系统，达到的目标不同，确定和划分的类也不相同。例如，一个学生管理系统，分类时既可以按专业分类，也可以按年级分类，甚至可以按年龄和性别分类。

由于问题的复杂性，不能指望一次就能正确地确定和划分类，需要不断地对实际问题进行分析和整理，反复修改才能得出正确的结果。另外，不能简单地将面向过程中的一个模块直接变成类，类不是模块函数的集合。

5.4 类的声明

类是面向对象程序设计的基础和核心，也是实现数据抽象的工具。类实质上是用户自定义的一种特殊的数据类型，特殊之处就在于和一般的数据类型相比，它不仅包含相关的数据，还包含能对这些数据进行处理的函数，同时，这些数据具有隐蔽性和封装性。类中包含的数据和函数统称为成员，数据称为成员数据，函数称为成员函数，它们都有自己的访问权限。

1．类声明的形式

类的声明即类的定义，其语法与结构体的声明类似，一般形式如下：

```
class 类名
{
  private:
    私有成员数据和成员函数
  protected:
    保护成员数据和成员函数
  public:
    公有成员数据和成员函数
};
```

其中，class 是声明类的关键字，类名是给声明的类起的名字；花括号给出了类的主体；最后的分号说明类的声明到此结束。

2．类声明的内容及类成员的访问控制

（1）类声明的内容。

类声明的内容包括成员数据和成员函数两部分，需要对类的成员数据和成员函数以及它们的访问权限做相应的说明。

① 成员数据———声明成员数据的数据类型、名字，以及访问权限。

② 成员函数———定义成员函数及对它们的访问权限。可以在类内定义成员函数，也可以在类外定义成员函数。在类外定义成员函数时先在类内说明该成员函数的原型，再在类外进行定义。

（2）访问控制———类的成员的访问控制是通过类的访问控制权限来实现的。访问控制权限分为三种：

① private：声明该成员为私有成员。私有成员只能被本类的成员函数访问，类外的任何成员对它的访问都是不允许的。私有成员是类中被隐蔽的部分，通常是描述该类对象属性的数据成员，这些成员数据用户无法访问，只有通过成员函数或某些特殊说明的函数才可访问，它体现了对象的封装性。当声明中默认控制访问权限时，系统默认成员为私有

成员。

② protected：声明该成员为保护成员，一般情况下与私有成员的含义相同，它们的区别表现在类的继承中对新类的影响不同。保护成员的具体内容将在本书后面的有关章节中介绍。

③ public：声明该成员为公有成员。公有成员可以被程序中的任何函数访问，它提供了外部程序与类的接口功能。成员函数通常是公有成员。

例 5-1　类与对象的简单举例。

```
class Student                                          //以 class 开头
{
    private:
     int num;
     char name[20];
     char sex;                                         //以上 3 行是类的数据成员
    public:
     void setstu(int n,char nam[20],char s)            //这是成员函数
    {
       num = n;
       strcpy(name,nam);
       sex=s;
    }
    void display( )                                    //这是成员函数
     {
       cout<<"num:"<<num<<endl;
       cout<<"name:"<<name<<endl;
       cout<<"sex:"<<sex<<endl;
     }
};                                                     //类定义完最后有一个“;”
void main()
{
  Student stud1,stud2;            //定义了两个 Student 类的对象 stud1 和 stud2
  stud1.setstu(1001, "Jerry" ,'m');
  stud1. display( );
  stud2.setstu(1002, "Flora" , 'f');
  stud2. display( );
}
```

例子中的“class Student”是类头，由关键字 class 与类名 Student 组成，class 是声明类时必须使用的关键字，后面紧跟的一对花括号是类体。类中的声明和操作需要写到类体中，类体中是类的成员列表，列出类中的全部成员。还需注意，类的声明以分号结束，可以看到类是一种导出的数据类型。这种数据类型中既包含数据，也包含对数据进行操作的函数，本例中 setstu 和 display 都是函数，其作用分别是给某个学生的数据赋值和输出本对象中所有学生的学号、姓名和性别。另外，值得一提的是类的定义只是定义了一种导出的数据类型，定义过程中并不为类中的数据成员分配任何的存储空间，所以不能直接在类定义的时

候对其中的数据成员初始化。

类中数据成员的存储空间是在建立对象的时候分配的，对象是类的具体实例。一个对象可以看作是一个具有某种类类型的变量。与普通变量一样，对象也必须先经声明才可以使用。声明一个对象的一般形式为：

```
<类名>  <对象 1>, <对象 2>, …
```

例如语句：Student stud1,stud2;即声明了两个名为 stud1 和 stud2 的学生，它们都是 Student 类的对象。通过对象访问其数据成员或函数成员时需要使用成员运算符“.”来访问对象的成员，在上面的例子中可以使用 stud1.setstu(1001, "Jerry" ,'m')，但不能使用 stud1.num=1001，原因是数据成员 num 是私有的，在类外无法直接访问到类中的私有数据成员，所以对类中私有的数据成员的操作只能通过公有的成员函数，例如通过 stud1.setstu(1001, "Jerry" , 'm');程序的执行就可以将名为 stud1 的学生学号赋值为 1001，姓名赋值为 Jerry，性别赋值为男。

同类型的对象之间可以相互赋值，例如 stud2=stud1；就是使得学生 stud2 和学生 stud1 具有相同的学号、姓名和性别。

成员函数函数体的定义除了可以定义在类体中，还可以在类体外进行定义。

例 5-2 在类体外定义成员函数功能。

```
class Student
{
  private:
    int num;
    char name[20];
    char sex;
  public:
     void setstu(int n,char nam[20],char s);    //这是成员函数的声明
     void display( );                           //这是成员函数的声明
};
void Student::setstu(int n,char nam[20],char s)//这是成员函数在类体外的定义
{
     num = n;
     strcpy(name,nam);
     sex=s;
}
void Student::display( )                        //这是成员函数在类体外的定义
   {
      cout<<"num:"<<num<<endl;
      cout<<"name:"<<name<<endl;
      cout<<"sex:"<<sex<<endl;
   }
void main()
{
  Student stud1,stud2;
```

```
  stud1.setstu(1001,"Jerry" , 'm');
  stud1. display( );
  stud2.setstu(1002, "Flora" , 'f');
  stud2. display( );
}
```

若在类体外定义成员函数，必须在成员函数名前加上类名和作用域运算符(::)。作用域运算符用来标识该函数成员属于哪个类。作用域运算符的使用格式为：

<类名>::<成员函数名>(<参数表>)

这样做的好处是可以在类体定义时清楚地看到类中所有的成员函数，特别对于成员函数非常多的类，这样定义使程序更清晰，有利于程序的可读性。

5.5　构造函数和析构函数

当声明了类并定义了类的对象以后，编译程序需要为对象分配内存空间，进行必要的初始化，C++中有一种专门的函数完成这个工作，就是构造函数。构造函数可以由系统自动生成，也可以由用户自己定义。对象被撤销时，就要回收内存空间，并做一些善后工作，这个任务是由析构函数完成的。析构函数也是属于类的，也可以由系统自动生成或用户自定义。

5.5.1　构造函数

1．构造函数的作用

构造函数是一种特殊的成员函数，被声明为公有成员，其作用是对对象进行初始化，分配内存空间。

2．构造函数的定义与使用

（1）构造函数的名字必须与类的名字相同。

（2）构造函数的参数可以是任何数据类型，但它没有返回值，不能为它定义返回值类型，包括 void 型在内。

（3）对象定义时，编译系统会自动地调用构造函数完成对象内存空间的分配和初始化工作。

（4）构造函数是类的成员函数，具有一般成员函数的所有性质，可访问类的所有成员，可以是内联函数，可以带有参数表，可以带有默认的形参值，还以可重载。

前面的例子是通过 setstu 函数来对对象初始化的，可以将该函数更换为构造函数，例如：

例 5-3　用构造函数初始化对象。

```
class Student
{
  private:
```

```
    int num;
    char name[20];
    char sex;
  public:
   Student (int n,char nam[20],char s)   //这是构造函数
   {
     num = n;
     strcpy(name,nam);
     sex=s;
   }
 void display( )
   {
      cout<<"num:"<<num<<endl;
      cout<<"name:"<<name<<endl;
      cout<<"sex:"<<sex<<endl;
      }
  }
};
void main()
{
  Student stud1(1001, "Jerry" , 'm'),stud2(1002, "Flora" , 'f');
  stud1. display( );
  stud2. display( );
}
```

当执行到语句"Student stud1(1001, "Jerry",'m') "时，构造函数会被系统自动调用，将语句中的三个实参 1001、Jerry 和 m 分别传给构造函数中的三个形参 n、nam 和 s，从而完成对对象 stud1 的赋值。可见，从函数调用的角度来看，构造函数是建对象时由系统自动调用的。

C++编译系统规定：每新建一个对象就要调用一次构造函数，当程序中没有定义任何构造函数时，系统会产生一个不带参数的什么都不做的默认构造函数，也称为缺省的构造函数，其形式为：

```
<类名>( )
{ }
```

在类中定义的构造函数，若没有参数或者所有的参数都具有缺省值时，也称其为缺省的构造函数。需要注意的是，对一个类来说，缺省构造函数只能有一个。

5.5.2 拷贝构造函数

拷贝构造函数是一种特殊的构造函数，具有一般构造函数的所有特性，其形参是本类对象的引用。其作用是使用一个已经存在的对象（由拷贝构造函数的参数指定的对象），去初始化一个新的同类的对象。

用户可以根据自己实际问题的需要定义特定的拷贝构造函数，以实现同类对象之间数

据成员的传递。如果用户没有声明类的拷贝构造函数，系统就会自动生成一个缺省拷贝构造函数，这个缺省拷贝构造函数的功能是把初始值对象的每个数据成员的值都复制到新建立的对象中。因此，也可以说是完成了同类对象的克隆（Clone），这样得到的对象和原对象具有完全相同的数据成员，即完全相同的属性。

定义一个拷贝构造函数的一般形式为：

```
class 类名
{
public:
   类名(形参表);                          //构造函数
   类名(类名& 对象名);                    //拷贝构造函数
   …
};
类名::类名(类名&对象名);                  //拷贝构造函数的实现
{
   函数体
}
```

下面给出一个拷贝构造函数的例子。对于屏幕上的一个点，可以通过给出水平和垂直两个方向的坐标值 X 和 Y 来确定。定义一个确定点的位置的类 Point。

```
class Point
{
public:
    Point (int xx=0, int yy=0 ) {X=xx; Y=yy; }      //构造函数
    Point (Point & p);                               //拷贝构造函数
    int GetX ( )  {return X; }
    int GetY ( )  {return Y; }
private:
    int X, Y;
};
```

类中定义了内联构造函数和拷贝构造函数。拷贝构造函数的实现如下：

```
Point::Point (Point & p)
{
   X=p. X;
   Y=p. Y;
   cout<< "拷贝构造函数被调用" << endl ;
}
```

普通构造函数是在对象创建时被调用，而拷贝构造函数在以下三种情况下都会被调用：

（1）当用类的一个对象去初始化该类的另一个对象时。例如：

```
int main (   )
{    Point A (1, 2);
```

```
    Point B (A);          //用对象 A 初始化对象 B，拷贝构造函数被调用
    cout<<B .GetX ( ) << endl ;
    return 0;
}
```

（2）如果函数的形参是类的对象，在调用函数时，进行形参和实参结合时。例如：

```
void f (Point p )
{   cout<< p . GetX ( )<<endl;
}
int main ( )
{   Point A (1, 2);
    f(A);     //函数的形参为类的对象，当调用函数时，拷贝构造函数被调用
    return 0;
}
```

（3）如果函数的返回值是类的对象，函数调用完成返回时。例如：

```
Point g ( )
{   Point A (1, 2);
    return A;   //函数的返回值是类对象，函数返回时，调用拷贝构造函数
}
int main ( )
{   Point B;
    B=g ( );
    return 0;
}
```

例 5-4 拷贝构造函数的使用方法。

```
#include<iostream>
usingnamespacestd;
classStudent
  {
    char *name;
    int no;
    static int counter;
    public:
      Student()
      {
        name="Paul";
        no=counter;
      }
Student(char  *na)
{
  name=na;
  no=counter;
  ++counter;
```

```
}
Student(Student  &stu)
{
  name=stu.name;
  no=counter;
  ++counter;
}
void display( )
   {
      cout<<"name:"<<name<<endl;
      cout<<"counter:"<<counter<<endl;
      }
};
int Student::counter=1;
void main()
{
  Student stu1("Jerry");
  Student stu2(stu1);
  stu1.display( );
  stu2.display( );
}
```

5.5.3　构造函数的重载

为了适应不同的情况，增加程序设计的灵活性，C++允许对构造函数重载，也就是可以定义多个参数及参数类型不同的构造函数，用多种方法对对象初始化。这些构造函数之间通过参数的个数或类型来区分。下面是构造函数重载的例子。

例 5-5　构造函数的重载。

```
class Student
  {
  private:
    int num;
    char name[20];
    char sex;
  public:
   Student( )
   {
     num = 0001;
     strcpy(name,"admin");
     sex='m';
   }
   Student (int n,char nam[20],char s)
   {
     num = n;
     strcpy(name,nam);
```

```
    sex=s;
}
void display( )
   {
   cout<<"num:"<<num<<endl;
   cout<<"name:"<<name<<endl;
   cout<<"sex:"<<sex<<endl;
   }
  }
};
void main()
{
  Student stud1(1001,"Jerry" , 'm'),stud2;
  stud1. display( );
  stud2. display( );
}
```

运行结果：

```
num:1001
name:Jerry
sex:m
num:0001
name:admin
sex:m
```

5.5.4 析构函数

析构函数与构造函数的作用正好相反，它用来完成对象被删除前的扫尾工作。析构函数是在撤销对象前由系统自动调用的，析构函数执行后，系统回收对象的存储空间，对象也就消失了。

析构函数也是类中的一种特殊的成员函数，它具有以下一些特性：

（1）析构函数名是在类名前加求反符号“~”构成；

（2）析构函数不指定返回类型；

（3）析构函数没有参数，也不能重载析构函数，即一个类只能定义一个析构函数；

（4）在撤销对象时，系统自动调用析构函数。

以下三种情况发生时系统会自动调用析构函数撤销对象：

（1）对象的作用域结束；

（2）用 delete 运算符释放 new 运算符创建的对象；

（3）生成的临时对象使用完毕后。

例 5-6 重新定义学生类 Student，增加析构函数。

```
class Student
{
  private:
```

```
   int num;
   char name[20];
   char sex;
  public:
   Student(int n,char nam[20],char s)       //这是构造函数
   {
    num = n;
    strcpy(name,nam);
    sex=s;
   }
~Student ()                                 //这是析构函数
   {
     cout<<"Destructor of Student " <<endl;
   }
void display( )
   {
      cout<<"num:"<<num<<endl;
      cout<<"name:"<<name<<endl;
      cout<<"sex:"<<sex<<endl;
   }
  }
};
void main()
{
 Student stud1(1001, "Jerry",'m');
 stud1.display ( ) ;
}       // stud1 的作用域结束，系统自动调用析构函数
```

一般来讲，如果希望程序在对象被删除时需要进行一些操作（例如信息保存等），就可以写到析构函数中。事实上，在很多情况下，用析构函数来进行扫尾工作是必不可少的。例如，在 Windows 操作系统中，每一个窗口就是一个对象，在窗口关闭之前，需要保存显示于窗口中的内容，就可以在析构函数中完成。

每个类都必须有且只能有一个析构函数。跟构造函数一样，如果在类中没有显式的定义析构函数，编译器将生成一个默认的析构函数，也称为缺省析构函数，它的格式为：

```
~<类名>() { }                               //空析构函数
```

例 5-7　下面的例子综合使用构造函数、拷贝构造函数、构造函数重载和析构函数。试分析程序的输出结果。

```
#include <iostream.h>
class Test{
public:
        Test()                              //缺省构造函数
        {
            val=0;
```

```
            cout<<"Default constructor."<<endl;
        }
        Test(int n)                                    //重载构造函数
        {
            val=n;
            cout<<"Constructor of "<<val<<endl;
        }
        Test(const Test& t)                            //拷贝构造函数
        {
            val=t.val;
            cout<<"Copy constructor."<<endl;
        }
        ~Test()                                        //析构函数
        {  cout<<"Destructor of "<<val<<endl;  }
        private:
        int val;
    };
    void fun1(Test  t)                                 //A
    {  }                                               //B
    Test fun2()
    {
        Test tt;                                       //C
        return tt;                                     //D
    }
    void main( )
    {
        Test t1(1);                                    //E
        Test t2=t1;                                    //F
        Test t3;                                       //G
        t3=t1;
        fun1(t2);                                      //H
        t3=fun2();                                     //I
    }                                                  //J
```

执行程序后，输出结果为：

```
Constructor of 1
Copy constructor.
Default constructor.
Copy constructor.
Destructor of 1
Default constructor.
Copy constructor.
Destructor of 0
Destructor of 0
Destructor of 0
```

```
Destructor of 1
Destructor of 1
```

在执行主函数时，E 行建立类 Test 的 t1 对象，调用了带参数的构造函数，输出第一行 Constructor of 1。F 行建立类 Test 的 t2 对象时，用 t1 的数据成员初始化 t2，将调用拷贝构造函数，输出第二行 Copy constructor。G 行建立对象 t3 时调用缺省构造函数，输出第三行 Default constructor。执行 H 行调用函数 fun1(t2)时，由于 fun1 采用的是传值调用方式，因此要建立一个局部对象 t，将实参对象 t2 复制给形参对象 t，调用拷贝构造函数，输出第四行 Copy constructor。执行到函数 fun1 结束时（H 行），形参 t 的作用域结束，撤销对象 t 时调用析构函数，输出第五行 Destructor of 1。执行 I 行时，先调用函数 fun2，进入函数 fun2 执行 C 行建立局部对象 tt 时，调用缺省构造函数，输出第六行 Default constructor。由于 tt 对象是一个局部对象，执行到 fun2 函数结束时要撤销该对象。为了返回对象 tt 的值，在执行 D 行时，系统新建立一个匿名（临时）对象，用对象 tt 初始化该匿名对象，调用拷贝构造函数，输出第七行 Copy constructor。在函数 fun2 返回时，撤销对象 tt 调用析构函数，输出第八行 Destructor of 0。在执行 I 行的赋值运算时，将 fun2 返回的匿名对象赋给对象 t3 后，系统撤销匿名对象时调用析构函数，输出第九行 Destructor of 0。执行到 J 行结束整个程序时，在主函数中建立的三个局部对象 t3、t2 和 t1 的作用域结束，撤销这三个对象时三次调用析构函数，输出第十至十二行 Destructor of 0、Destructor of 1 和 Destructor of 1。

注意在主函数中，建立对象的顺序是 t1、t2 和 t3，而撤销对象的顺序则是 t3、t2 和 t1。析构函数的执行顺序与构造函数的执行顺序相反。

5.6　对象应用

5.6.1　成员对象

类的数据成员可以是基本类型的数据，也可以是类类型的对象。因此，可以利用已定义的类的对象作为另外一个类的数据成员，这些对象就称为成员对象。

当类中出现了成员对象时，该类的构造函数要包含对成员对象的初始化，通常采用成员初始化列表的方法来初始化成员对象。定义带有成员初始化列表的构造函数的一般格式为：

```
<类名>(<形参表>) : <成员对象 1>(<实参表 1>),<成员对象 2>(<实参表 2>),…
{
        …                                    //类成员的初始化
}
```

其中，成员对象实参表中可以使用类的形参表中的参数名，也可以使用常量表达式。

在使用成员对象时要注意以下几点：

（1）成员对象的初始化必须在该类的构造函数的成员初始化列表中进行。

（2）当建立一个类的对象时，如果这个类中有成员对象，执行构造函数时，先执行所有成员对象的构造函数，再执行本类对象的构造函数体。当类中有多个成员对象时，调用

其构造函数的顺序仅与成员对象在类中说明的顺序有关，与成员初始化列表中给出的成员对象的顺序无关。

（3）如果在构造函数的成员初始化列表中没有给出对成员对象的初始化，则表示使用成员对象的缺省构造函数。如果成员对象所在的类中没有缺省构造函数时，将产生错误。

（4）析构函数的执行顺序与构造函数的执行顺序相反。

例 5-8 下面用例子说明类中的成员对象的产生和撤销关系，分析以下程序的输出结果。

```
#include <iostream.h>
class Counter{
    int val;
public:
    Counter() { val=0; cout<<"Default Constructor of Counter"<<endl; }
                                    //缺省构造函数
    Counter(int x) { val=x; cout<<"Constructor of Counter:"<<val<<endl; }
                                    //重载构造函数
    ~Counter() { cout<<"Destructor of Counter:"<<val<<endl; } //析构函数
};
class Example{
    Counter c1,c2;                                          //A
    int val;
public:
    Example() { val=0; cout<<"Default Constructor of Example"<<endl; }
                                    //缺省构造函数
    Example(int x):c2(x),c1()       //重载构造函数和成员初始化列表
    { val=x; cout<<"Constructor of Example:"<<val<<endl; }
    ~Example() { cout<<"Destructor of Example:"<<val<<endl; }
    void Print() { cout<<"value="<<val<<endl; }
};
void main( )
{
    Example e1,e2(4);                                       //B
    e2.Print();                                             //C
}
```

执行程序后，输出结果为：

```
Default Constructor of Counter
Default Constructor of Counter
Default Constructor of Example
Default Constructor of Counter
Constructor of Counter:4
Constructor of Example:4
value=4
Destructor of Example:4
Destructor of Counter:4
Destructor of Counter:0
```

```
Destructor of Example:0
Destructor of Counter:0
Destructor of Counter:0
```

以上程序中，类 Example 中有 Counter 类的成员对象 c1 和 c2。因此建立 Example 类的对象时，要同时建立成员对象 c1 和 c2。成员对象的初始化在构造函数的成员初始化列表中进行，建立对象时要首先执行成员对象 c1 和 c2 的构造函数，再执行 Example 的构造函数体。

执行 B 行建立 Example 类的对象 e1 时，调用 Example 类的缺省构造函数。Example 类的缺省构造函数的成员初始化列表为空，则先调用成员对象 c1 和 c2 的缺省构造函数。因此先输出第一行和第二行，然后执行 Example 的缺省构造函数的函数体时输出第三行。在 B 行中 接着建立对象 e2 时，根据 e2 的参数，调用 Example 类中带有一个参数的构造函数。在该构造函数的成员初始化列表中，已显式列出了成员对象 c2 的初始化。由于成员对象的初始化顺序与成员对象在类中说明的顺序有关，而与成员初始化列表中给出的顺序无关，因此应先对 c1 初始化，调用 Counter 类的缺省构造函数，输出第四行；再对 c2 初始化，调用带有一个参数的构造函数，输出第五行；然后，执行 Example 类中带有一个参数的构造函数，输出第六行。执行 C 行时，输出第七行，即 value=4。

主程序结束要撤销对象 e1 和 e2 时，调用析构函数。析构函数的执行顺序与构造函数的执行顺序相反，先执行 e2 的析构函数，再执行 e1 的析构函数。析构函数的执行顺序正好与对立对象时的顺序相反，由于对象 e2 中包含有成员对象，在执行完 e2 的析构函数后，立即执行其成员函数的析构函数，接着执行成员对象 c2 的析构函数，最后执行 c1 的析构函数。注意，对于具有同一类生存期的多个类的对象，不论这些对象所属的类是一个独立的类，还是含有成员对象的类，建立对象时构造函数的执行顺序与撤销对象时析构函数的执行顺序都是严格相反的。

5.6.2　对象数组

对象数组是指数组元素为对象的数组，该数组中所有元素必须是同一个类的对象。对象数组的定义、赋值和使用与普通数组一样，只是数组的元素与普通数组不同，它是类的对象。

对象数组的定义格式为：

```
<类名> <数组名>[<大小>]…;
```

例如：

```
Student stud[5];
```

使用对象数组成员的一般格式是：

```
<数组名>[<下标表达式>].<成员名>
```

例如：

```
stud[0].name;
```

使用对象数组时，对象数组的初始化和对象数组元素的赋值也是使用 for 循环进行的。

例如：

```
for(i=0;i<5;i++)
{
  stud[i].num=n;
  strcpy(stud[i].name,nam);
  stud[i].sex=s;
}
```

5.6.3 对象指针

对象指针指的是一个对象在内存中的首地址。取得一个对象在内存中首地址的方法与取得一个变量在内存中首地址的方法一样，都是通过取地址运算符“&”。例如，若有：

```
Student  *ptr, s1;
```

则 ptr=&s1;表示对象 s1 在内存中的首地址并赋给指针变量 ptr，即指针变量 ptr 指向对象 s1 在内存中的首地址。

若要使用对象指针引用对象成员，首先要定义对象指针，再把它指向一个已创建的对象或对象数组，然后引用该对象的成员或对象数组元素的成员。用对象的指针引用对象成员或对象数组元素的成员使用操作符“→”，而不是“•”，这一点与指针和结构体变量的结合是一样的。

例 5-9 用对象指针引用对象数组。

```
#include<iostream.h>
class Student
{
  int num;
  public:
    void set_num(int a)
    {  num=a;  }
    void show_ num ()
    {  cout<<num<<endl;  }
};
void main()
{
  Student *ptr,s[2];
  s[0].set_ num (1011);
  s[1].set_ num (1012);
  ptr=s;
  ptr->show_ num ();
  ptr++;
  ptr-> show_ num ();
}
```

运行结果：

```
1011
1012
```

C++提供了一个特殊的对象指针——this 指针。它是指向调用该成员函数（包括构造

函数）的当前对象的指针，成员函数通过这个指针可以知道当前是哪一个对象在调用它。

this 指针是一个隐含的指针，它隐含于每个类的成员函数中，它明确地表示成员函数当前操作数据所属的对象。当一个对象调用成员函数时，编译程序先将对象的地址赋给 this 指针，然后调用该成员函数，每次成员函数存取数据成员时，都隐含地使用了 this 指针。

例 5-10　使用 this 指针操作类和对象。

```
#include<iostream>
using namespace std;
class Student                              //以 class 开头
{
private:
   int num;
   char name[20];
   char sex;                               //以上 3 行是数据成员
public:
   void setstu(int num,char name[20],char sex)
   {
    this->num = num;
    strcpy(name, this->name);
    this->sex=sex;
}
void display( )                            //这是成员函数
   {
      cout<<"num:"<<num<<endl;
      cout<<"name:"<<name<<endl;
      cout<<"sex:"<<sex<<endl;
     }
  }
};
void main()
{
  Student stud;
  stud.setstu(1001, "Jerry" ,'m');
  stud. display( );
}
```

可见，“this->成员变量”的形式即为成员函数访问类中数据成员的形式。注意，this 指针只能在类的成员函数中使用，它指向该成员函数被调用的对象，另外，只有非静态成员函数才有 this 指针，静态成员函数没有 this 指针。

5.7　静态成员

静态成员是指类中用关键字 static 说明的成员，包括静态成员数据和静态成员函数。

静态成员用于解决同一个类的不同对象之间数据和函数共享的问题，也就是说，不管这个类创建了多少个对象，这些对象的静态成员使用同一个内存空间，所以，静态成员是属于所有对象的，或者说是属于整个类的，有的书上又称之为类成员。

5.7.1 静态成员数据

静态成员数据是指类中用关键字 static 说明的数据成员。它是类的数据成员的特例，类的所有对象共用静态数据成员的内存空间，从而实现同类对象之间的数据共享。使用静态数据成员时应注意以下问题：

（1）静态数据成员声明时，加关键字 static 说明。

（2）静态数据成员必须初始化，但只能在类外进行。一般放在声明与 main()之间的位置。静态数据成员的值缺省时默认为 0。初始化的形式为：<数据类型><类名>::<静态数据成员名>=<值>，例如，int Student::num=0。

（3）静态数据成员不属于任何一个对象，可以在类外通过类名对它进行引用。引用的一般形式为：<类名>::<静态数据成员名>。

（4）静态成员与普通数据成员一样，要服从访问控制，当它被声明为私有成员时，只能在类内直接引用，类外无法引用。当它被声明为公有成员时，可以在类外通过类名来引用。

例 5-11 使用静态数据成员的例子。

```
#include<iostream.h>
class Csample
{
    int n;
    static int k;
public:
    CSample(int i){n=i;k++;};
    void disp();
};
void CSample::disp( )
{
    cout<<"n="<<n<<",k="<<k<<endl;
}
int CSample::k=0;
void main( )
{
    CSample a(10),b(20),C(30),d(40);
    a.disp();
    b.disp();
    C.disp();
    d.disp();
}
```

运行结果：

```
n=10   k=4
n=20   k=4
n=30   k=4
n=40   k=4
```

5.7.2　静态成员函数

静态成员函数是指类中用关键字 static 说明的成员函数。它属于类，由同一个类的所有对象共同使用和维护，为这些对象所共享。静态成员函数可以直接引用该类的静态数据成员和成员函数，不能直接引用非静态数据成员。使用静态成员函数时应注意以下问题：

（1）静态成员函数可以在类内定义，也可以在类外定义，在类外定义时不用前缀 static。

（2）系统限定静态成员函数为内部连接，所以不会与文件连接的其他同名函数相冲突，保证了静态成员函数的安全性。

（3）静态成员函数中没有隐含 this 指针，调用时可用下面两种方法之一：

```
<类名>::<静态函数名()>;  或  <对象名>::<静态函数名()>;
```

（4）一般而言，静态成员函数不能访问类中的非静态成员。

例 5-12　使用静态成员函数的例子。

```
#include<iostream>
using namespace std;
class Student
{
    private:
        float score;
        static int count;
        static float total_score;
    public:
    void account(float score1)
    {
      score=score1;
      ++count;
      total_score=total_score+score;
    }
    static float sum()
    {
      return total_score;
    }
      static float average()
    {
      return total_score/count;
    }
};
  int Student:: count=0;
```

```
float Student:: total_score=0.0;
int main()
{
  Student s1,s2;
  s1.account(99);
  cout<<Student::sum()<<endl;
  cout<<Student::average()<<endl;
  s2.account(80);
  cout<<Student::sum()<<endl;
  cout<<Student::average()<<endl;
}
```

运行结果：

```
99
99
179
89.5
```

5.8 友元函数和友元类

C++引入了友元函数和友元类来解决特殊情况下在类的外部直接访问类中私有成员的问题，即友元提供了类的成员函数与一般函数之间、不同类或对象的成员函数之间进行数据共享的一种手段。通过友元这种方式，一个普通函数可以访问封装在类内部的私有成员，即在类的外部通过友元可以看见类内部的一些属性。但这样做，在一定程度上会破坏类的封装性，使程序的可维护性变差，使用时一定要慎重。

一个类中，声明为友元的可以是不属于任何类的一般函数，也可以是另一个类的成员函数，还可以是一个完整的类。

5.8.1 友元函数

友元函数是在类声明中用关键字 friend 说明的非成员函数。它不是当前类的成员函数，而是独立于当前类的外部函数，可以访问当前类的所有私有或公有成员。其位置可以定义在类内部，也可以定义在类外部，但通常都定义在类外部。普通函数声明为友元函数的一般形式为：

```
friend<数据类型><友元函数名>(参数表);
```

下面是友元函数的例子。

例 5-13 普通函数作为友元函数示例。

```
#include <iostream.h>
#include <string.h>
class CStudent{
private:
```

```
  char m_name[20];
  int m_id;
  char m_gender;
  int  m_age;
public:
   CStudent(char *,int,char,int);
   friend void output(CStudent &stu);
};
CStudent::CStudent(char *name,int id,char gender,int age)
{
    strcpy(m_name,name);
    m_id=id;
    m_gender=gender;
    m_age=age;
}
void output(CStudent &stu)
{
    cout<<"姓名"<<stu.m_name<<endl;
    cout<<"学号"<<stu.m_id<<endl;
    cout<<"性别"<<stu.m_gender<<endl;
    cout<<"年龄"<<stu.m_age<<endl;
}
void main()
{
   CStudent  stu("Jerry",1001,'m',6);
   output(stu);
}
```

使用友元函数应注意由于友元函数不是成员函数，因此，在类外定义友元函数体时，不必像成员函数那样，在函数名前加“类名::”。另外，当一个函数需要访问多个类时，应该把这个函数同时定义为这些类的友元函数，这样，这个函数才能访问这些类的数据，如例 5-14 所示。

例 5-14　友元函数访问多个类的数据。

```
#include<iostream.h>
#include<string.h>
    class boy;           //提前声明，以便使用该类的对象
    class girl
    {
      char name[20];
      int age;
      public:
        void init(char n[],int a);
        friend void prt(girl ob1,boy ob2);  //声明函数prt()为类的girl友元函数
    };
```

```
void girl::init (char n[],int a)
{
  strcpy(name,n);
  age=a;
}
class boy
{
  char name[20];
  int age;
  public:
    void init(char n[],int a);
    friend void prt(girl ob1, boy ob2);   //声明函数prt()为类的boy友元函数
};
void boy::init (char n[],int a)
{
  strcpy(name,n);
  age=a;
}
void prt(girl ob1, boy ob2)
{
  cout<<"name:"<<ob1.name<<"     age:"<<ob1.age<<"\n";
  cout<<"name:"<<ob2.name<<"     age:"<<ob2.age<<"\n";
}
void main()
{
  girl g1,g2,g3;
  boy b1,b2,b3;
  g1.init("Leah",6);
  g2.init("Stacy",7);
  g3.init("Judith",5);
  b1.init("Jerry",6);
  b2.init("Michael",5);
  b3.init("Jim",7);
  prt(g1,b1);                          //调用友元函数prt()
  prt(g2,b2);                          //调用友元函数prt()
  prt(g3,b3);                          //调用友元函数prt()
}
```

运行结果：

```
name: Leah          age:6
name: Jerry         age:6
name: Stacy         age:7
name: Michael       age:5
name: Judith        age:5
name: Jim           age:7
```

5.8.2　友元成员

如果一个类的成员函数是另一个类的友元函数，则称这个成员函数为友元成员函数。通过友元成员函数，不仅可以访问自己所在类对象中的私有和公有成员，还可访问由关键字 friend 声明语句所在的类对象中的私有和公有成员，从而可使两个类数据共享，相互合作，协调工作，完成某个任务。

例 5-15　使用友元成员的例子。

```
#include<iostream.h>
#include<string.h>
class boy;
class girl
{
  char *name;
  int age;
  public:
    girl(char *n,int a)
    {
      name=new char[strlen(n)+1];
      strcpy(name,n);
      age=a;
    }
    void prt(boy &b);
    ~girl()
    {  delete name;  }
};
class boy
{
  char *name;
  int age;
  public:
    boy(char *n,int a)
    {
      name=new char[strlen(n)+1];
      strcpy(name,n);
      age=a;
    }
    friend void girl::prt(boy &b);
    ~boy()
    {  delete name;  }
};
void girl::prt (boy &b)
{
  cout<<"girl\'s name:"<<name<<"     age:"<<age<<"\n";
```

```
  cout<<"boy\'s name:"<<b.name<<"     age:"<<b.age<<"\n";
}
void main()
{
  girl g("Stacy",15);
  boy b1("Jim",16);
  g.prt(b1);
}
```

运行结果：

```
girl's name: Stacy  age:15
boy's name: Jim     age: 16
```

应注意的是，当一个类的成员函数作为另一个类的友元函数时，必须先定义成员函数所在的类，如例 5-15 中，类 girl 的成员函数 prt()为类 boy 的友元函数，就必须先定义类 girl。并且在声明友元函数时，要加上成员函数所在类的类名和运算符“::”，如例 5-15 中的语句：

```
friend void girl::prt(boy &b);
```

另外，在主函数中一定要创建一个类 girl 的对象。只有这样，才能通过对象名调用友元函数。如例 5-15 中主函数的语句：

```
girl g("Stacy",15);
```

5.8.3 友元类

1．友元类的概念

当一个类作为另一个类的友元时，称这个类为友元类。当一个类成为另一个类的友元类时，这个类的所有成员函数都成为另一个类的友元函数，因此，友元类中的所有成员函数都可以通过对象名直接访问另一个类中的私有成员，从而实现了不同类之间的数据共享。

2．友元类的声明

友元类声明的形式如下：

```
friend class <友元类名>; 或 friend <友元类名>;
```

友元类的声明可以放在类声明中的任何位置，这时，友元类中的所有成员函数都成为友元函数。

例 5-16 使用友元类的例子。

```
#include<iostream.h>
#include<string.h>
class boy;
class girl
{
  char *name;
  int age;
  public:
```

```
    girl(char *n,int a)
    {
      name=new char[strlen(n)+1];
      strcpy(name,n);
      age=a;
    }
    void prt(boy &b);
    ~girl()
    {  delete name;  }
};
class boy
{
  char *name;
  int age;
  friend  girl;                    //声明类 girl 为类 boy 的友元类
  public:
    boy(char *n,int a)
    {
      name=new char[strlen(n)+1];
      strcpy(name,n);
      age=a;
    }
    ~boy()
    {  delete name;  }
};
void girl::prt (boy &b)
{
  cout<<"girl\'s name:"<<name<<"     age:"<<age<<"\n";
  cout<<"boy\'s name:"<<b.name<<"     age:"<<b.age<<"\n";
}
void main()
{
  girl g("Stacy",15);
  boy b1("Jim",16);
  g.prt(b1);
}
```

运行结果：

```
girl's name: Stacy   age:15
boy's name: Jim    age: 16
```

关于友元，还有两点要注意：

① 友元关系是不能传递的。类 B 是类 A 的友元，类 C 是类 B 的友元，类 C 与类 A 之间，除非特别声明，没有任何关系，不能进行数据共享。

② 友元关系是单向的。类 B 是类 A 的友元，类 B 的成员函数可以访问类 A 的成员，反之，类 A 的成员函数却不可以访问类 B 的成员。

5.9 本章任务实践

5.9.1 任务需求说明

利用面向对象编程方法设计一个学生成绩单管理系统，要求实现以下功能：

（1）录入（添加）学生信息：学号、姓名、平时成绩和考试成绩，系统自动计算总评成绩（平时成绩占 20%，考试成绩占 80%）。可以一次录入多名学生的信息。

（2）查询学生成绩：输入要查询的学生的学号，查询该学生的信息并显示。

（3）显示学生成绩单：按学号顺序显示学生成绩单。

（4）删除学生信息：输入要删除的学生的学号，得到用户确认后，删除该学生的信息。

（5）修改学生信息：输入要修改的学生的学号，显示该学生的原有信息，用户输入修改后的信息。

（6）对成绩进行统计分析：可以对总成绩进行统计分析，分别统计出各个成绩段的人数和比例，本课程班级平均成绩等。

各运行界面如图 5-1～图 5-7 所示。

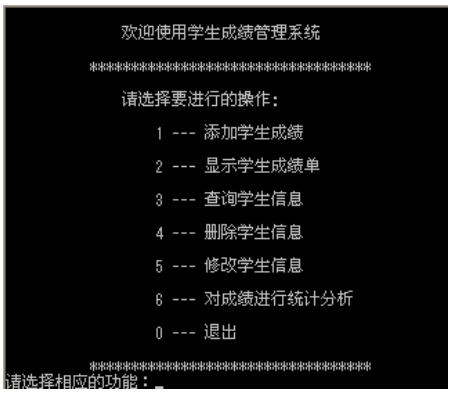

图 5-1 成绩管理系统主界面

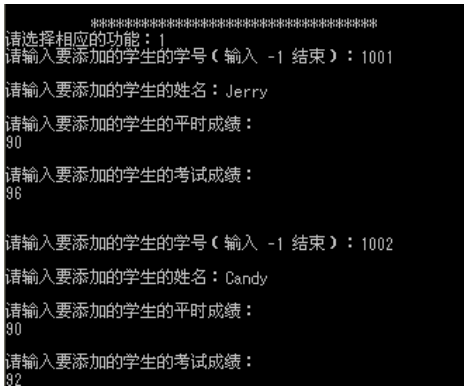

图 5-2 增加学生信息

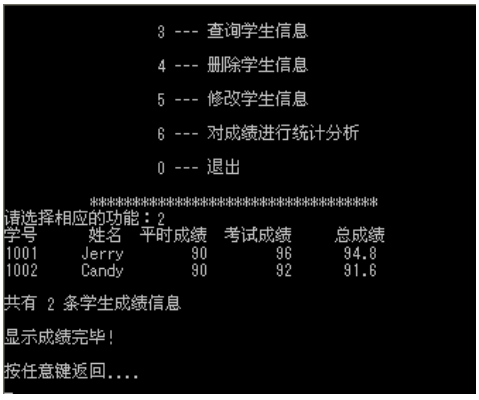

图 5-3 显示学生信息

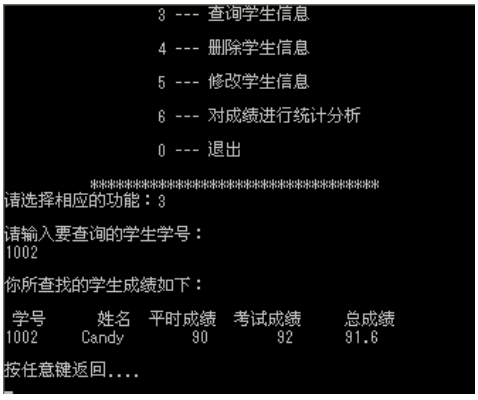

图 5-4 查询学生信息

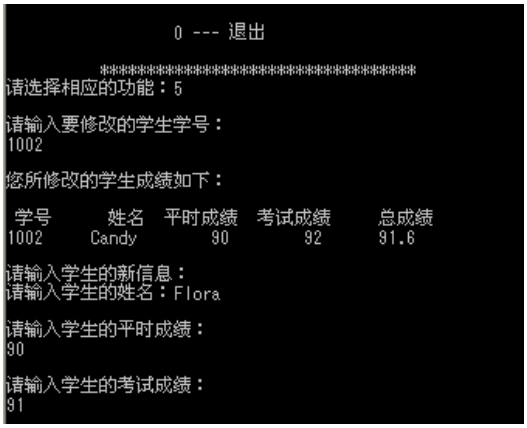

图 5-5　修改学生信息

图 5-6　成绩统计分析

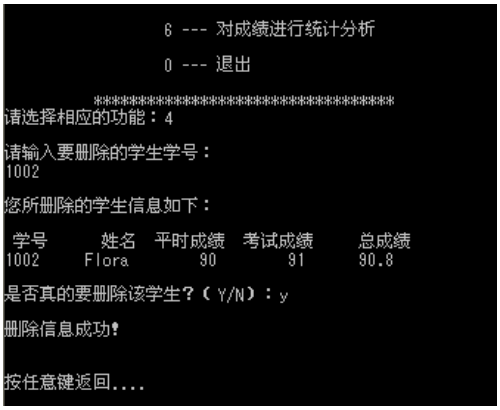

图 5-7　删除学生信息

5.9.2　技能训练要点

要完成以上项目，读者需要学会使用面向对象的方法进行编程，掌握类与对象的定义和使用，学会使用对象数组来存储和操作数据，熟练掌握构造函数、拷贝功能的构造函数和析构函数的概念和用法。

5.9.3　任务实现

```
#include <iostream.h>
#include <string.h>
#include <iomanip.h>
#include <conio.h>

class CStudent {
public:
CStudent(char * id="",char *na="",int us=0, int ts=0);     //有参构造函数
CStudent(const CStudent &s);                               //拷贝构造函数
```

```
~CStudent();
char* GetID();                    //获取学生的学号
double GetTotalScore();           //获取总评成绩
static void TableHead( );         //输出表头
void Display( );                  //显示学生信息
private:
char ID[5];                       //学号
char name[10];                    //姓名
int UsualScore;                   //平时成绩
int TestScore;                    //考试成绩
double TotalScore;                //总评成绩
void CalcTotalScore();            //计算总评成绩
};
int num;                          //学生人数
class CStuDatabase {
public:
CStuDatabase();                   //构造函数
~CStuDatabase();                  //析构函数
 void ListScore( );               //显示成绩单，输出所有学生信息
 void SelectStuInfo( );           //查询学生信息
 void AddStuInfo( );              //添加学生成绩
 void DelStuInfo( );              //删除学生信息
 void EditStuInfo( );             //修改学生信息
 void AnalyScore( );              //对成绩进行统计分析
 void StuDBM( int );              //成绩库维护
 int FunctionMenu();                  //功能菜单
private:
   CStudent stu[51]; //学生数组,stu[0]不用，一个类的对象作为另外一个类的数据成员
   int SearchStu(const char* id);   //查找指定学号的学生
   void SortStu( );                 //按学号从小到大对成绩单排序
};
int InputScore( )                   //输入百分制成绩
{     int score;
      cin>>score;
while ( score<0 || score>100 )
{   cout<<"成绩超出范围，请重新输入百分制成绩（0---100 分）:";
    cin>>score;
}
return score;
}
CStudent::CStudent( char * id, char *na,int us, int ts )   //构造函数
{
strcpy(ID,id);
strcpy(name,na);
   UsualScore=us;
```

```
    TestScore=ts;
    CalcTotalScore();
}
CStudent::CStudent( const CStudent &s )          //拷贝构造函数
{
strcpy( ID, s.ID );
strcpy( name, s.name );
UsualScore=s.UsualScore;
TestScore=s.TestScore;
TotalScore=s.TotalScore;
}
CStudent::~CStudent()
{  }
char* CStudent::GetID()                          //取得学生的学号
{   return ID; }
double CStudent::GetTotalScore()                 //获取总成绩
{   return TotalScore;}
void CStudent::TableHead( )                      //输出学生信息表头
{
  cout<<setw(4)<<"学号"<<setw(10)<<"姓名"<<setw(10)<<"平时成绩"<<setw(10)
<<"考试成绩"<<setw(12)<<"总成绩\n";
}
void CStudent::Display( )                        //显示学生信息
{   cout<<setw(3)<<ID<<setw(10)<<name<<setw(10)<<UsualScore
            <<setw(10)<<TestScore<<setw(10)<<TotalScore<<endl;
}
void CStudent::CalcTotalScore()                  //计算总成绩
{   TotalScore= UsualScore*0.2 + TestScore*0.8;   }

CStuDatabase::CStuDatabase()
{
}

CStuDatabase::~CStuDatabase()
{
}

int CStuDatabase::SearchStu(const char * id)   //查找指定学号的学生
{
for ( int i=1; i<=num; i++ )
    if ( strcmp(stu[i].GetID(),id)==0 )
        return i;
return -1;
}
int CStuDatabase::FunctionMenu()                 //功能菜单
```

```
{   int FuncNum;    //保存操作编号
cout<<"\n";
   cout<<setw(14)<<' '<<"欢迎使用学生成绩管理系统\n\n";
cout<<setw(10)<<' '<<"*********************************\n\n";
cout<<setw(14)<<' '<<"请选择要进行的操作:\n\n";
cout<<setw(18)<<' '<<"1 --- 添加学生成绩\n\n"
    <<setw(18)<<' '<<"2 --- 显示学生成绩单\n\n"
    <<setw(18)<<' '<<"3 --- 查询学生信息\n\n"
    <<setw(18)<<' '<<"4 --- 删除学生信息\n\n"
    <<setw(18)<<' '<<"5 --- 修改学生信息\n\n"
    <<setw(18)<<' '<<"6 --- 对成绩进行统计分析\n\n"
    <<setw(18)<<' '<<"0 --- 退出\n\n";
   cout<<setw(10)<<' '<<"*********************************\n";
cout<<"请选择相应的功能：";
    cin>>FuncNum;
while ( FuncNum<0 || FuncNum>6 )
{
    cout<<"请重新选择要进行的操作："<<endl;
    cin>>FuncNum;
}
   return FuncNum;
}
void CStuDatabase::StuDBM( int FuncNum )   //成绩维护
{
switch ( FuncNum )
  {
    case 1: AddStuInfo();  break;           //添加学生成绩
    case 2: ListScore( ); break;            //显示成绩单
    case 3: SelectStuInfo( );  break;       //查询学生信息
    case 4: DelStuInfo( ); break;           //删除学生信息
    case 5: EditStuInfo( );  break;         //修改学生信息
    case 6: AnalyScore( );  break;          //对成绩进行统计分析
  }
}
void CStuDatabase::SelectStuInfo( )         //查询学生信息
{
char no[5];   //临时保存学号
cout<<"\n 请输入要查询的学生学号："<<endl;
cin>>no;
int i=SearchStu(no);
if ( i==-1 )
{   cout<<"\n 你查找的学生不存在！\n"; }
else
   {    cout<<"\n 你所查找的学生成绩如下：\n\n ";
    CStudent::TableHead( );    //输出表头
```

```
    stu[i].Display();
}
cout<<"\n 按任意键返回...."<<endl;
   getch();
}
void CStuDatabase::ListScore( )     //显示成绩单
{
if ( num == 0 )
{   cout<<"当前还没有学生成绩! \n";    }
else
{
    SortStu( ); //按学号对成绩单排序
    CStudent::TableHead( );         //输出表头
    for ( int i=1; i<=num; i++ )
        stu[i].Display();
    cout<<"\n 共有 "<<num<<" 条学生成绩信息\n";
}
cout<<"\n 显示成绩完毕!\n\n 按任意键返回...."<<endl;
getch();
}
void CStuDatabase::AddStuInfo( )   //添加学生成绩
{
    char no[5];   //临时保存学号
cout<<"请输入要添加的学生的学号（输入 -1 结束）: ";
cin>>no;
while ( strcmp(no,"-1")!=0 )
{
    int i=SearchStu( no );
    while ( i!=-1 )
    {   cout<<"\n 你添加的学生已存在! \n 请重新输入学号（-1 结束）: ";
    cin>>no;
        if ( strcmp(no,"-1")==0 )
        {
            cout<<"\n 本次操作完成!\n\n 按任意键返回...."<<endl;
            getch();
            return;
        }
        i=SearchStu( no );
    }
   num++;
    char na[10];
    cout<<"\n 请输入要添加的学生的姓名: ";
    cin>>na;
    cout<<"\n 请输入要添加的学生的平时成绩: \n";
    int us = InputScore();
```

```
    cout<<"\n 请输入要添加的学生的考试成绩：\n";
    int ts = InputScore();
    CStudent s(no,na,us,ts);
    stu[num]=s;
    cout<<"\n\n 请输入要添加的学生的学号（输入 -1 结束）：";
   cin>>no;
   }
   cout<<"\n 本次操作完成!\n\n 按任意键返回...."<<endl;
getch();
}
void CStuDatabase::DelStuInfo( ) //删除学生信息模块
{
   char no[5];   //临时保存学号
cout<<"\n 请输入要删除的学生学号："<<endl;
cin>>no;
int i=SearchStu( no );
if ( i==-1 )
{   cout<<"\n 你要删除的学生不存在！\n";    }
else
  {   cout<<"\n 您所删除的学生信息如下：\n\n ";
CStudent::TableHead( );    //输出表头
stu[i].Display();
char anser;
cout<<"\n 是否真的要删除该学生？（Y/N）：";
cin>>anser;
if ( anser=='y' || anser=='Y')
{
for ( int j=i+1; j<=num; j++ )
stu[j-1]=stu[j];
num--;
cout<<"\n 删除信息成功！"<<endl;
}
    }
    cout<<"\n\n 按任意键返回...."<<endl;
    getch();
}
void CStuDatabase::EditStuInfo( ) //修改学生信息模块
{
   char no[5];   //临时保存学号
    cout<<"\n 请输入要修改的学生学号："<<endl;
cin>>no;
int i=SearchStu( no );
if ( i==-1 )
{   cout<<"\n 你要修改的学生不存在！\n";    }
else
```

```
    {    cout<<"\n 您所修改的学生成绩如下：\n\n ";
    CStudent::TableHead( );    //输出表头
    stu[i].Display();
       cout<<"\n 请输入学生的新信息：";
    cout<<"\n 请输入学生的姓名：";
    char na[10];
    cin>>na;
    cout<<"\n 请输入学生的平时成绩：\n";
    int us = InputScore();
    cout<<"\n 请输入学生的考试成绩：\n";
    int ts = InputScore();
    CStudent s(no,na,us,ts);
    stu[i]=s;
cout<<"\n 修改信息成功！"<<endl;
}
cout<<"\n\n 按任意键返回...."<<endl;
getch();
}
void CStuDatabase::AnalyScore( )    //对成绩进行统计分析
{
    int c[5]={0};                   //用来保存各个分数段的人数
    double AveScore=0;              //用来保存所有学生的平均成绩
double ts;                          //临时保存总评成绩
for ( int i=1; i<=num; i++ )
{
    ts=stu[i].GetTotalScore();
    AveScore+=ts;
    switch ( int( ts/10 ) ){
    case 10:
    case 9:   c[0]++;   break;      //90（含 90）分以上人数
       case 8:    c[1]++;   break;  //80（含 80）---90（不含 90） 分人数
       case 7:    c[2]++;   break;  //70（含 70）---80（不含 80） 分人数
       case 6:    c[3]++;   break;  //60（含 60）---70（不含 70） 分人数
       default:  c[4]++;   break;   //不及格人数
        }
    }
    AveScore/=num;
cout<<"\n 学生成绩分布情况如下：\n\n";
cout<<"优秀（90 分---100 分）人数："<<c[0]<<", \t 占 "
<<double(c[0])/num*100<<" %\n\n";
cout<<"良好（80 分--- 89 分）人数："<<c[1]<<", \t 占 "
<<double(c[1])/num*100<<" %\n\n";
    cout<<"中等（70 分--- 79 分）人数："<<c[2]<<", \t 占 "
<<double(c[2])/num*100<<" %\n\n";
cout<<"及格（60 分--- 69 分）人数："<<c[3]<<", \t 占 "
```

```
<<double(c[3])/num*100<<" %\n\n";
cout<<"不及格（ 60 分以下 ）人数："
<<c[4]<<", \t 占 "<<double(c[4])/num*100<<" %\n\n";
cout<<"学生总人数为："<<num<<endl;
cout<<"\n 班级平均成绩为："<<AveScore<<endl;
cout<<"\n 按任意键返回...."<<endl;
getch();
}
void CStuDatabase::SortStu( )       //按学号从小到大对成绩单排序
{   int i, j, k;
for ( i=1; i<num; i++ )
{
    k=i;
    for ( j=i+1; j<=num; j++ )
        if ( strcmp( stu[j].GetID(),stu[k].GetID() )<0 )
            k=j;
       CStudent temp=stu[i];
stu[i]=stu[k];
stu[k]=temp;
}
}

void main()
{
CStuDatabase stuDB;                 //生成成绩单对象
int FuncNum;                        //保存操作编号
FuncNum=stuDB.FunctionMenu();       //显示功能菜单
while ( FuncNum!=0 )
{
    stuDB.StuDBM( FuncNum );        //学生库管理
    FuncNum=stuDB.FunctionMenu();
}
}
```

本章小结

本章主要讲解了面向对象程序设计的思想和方法，详细介绍了类和对象的概念及它们之间的关系，在此基础上对构造函数、拷贝功能的构造函数、析构函数、友元函数与友元

类等内容做了进一步的介绍，并在应用方面与指针、函数、数组等内容结合起来，使读者会用面向对象的思想解决更为复杂的问题，为后续的学习奠定基础。

课后练习

1．下列关于类的定义格式描述中，错误的是_____。
   A．类中成员有三种访问权限
   B．类的定义可分说明部分和实现部分
   C．类中成员函数都是公有的，数据成员都是私有的
   D．定义类的关键字通常用 class
2．下列关于对象的描述中，错误的是_____。
   A．定义对象时系统会自动进行初始化
   B．对象成员的表示与 C 语言中结构变量的成员表示相同
   C．属于同一个类的对象占有内存字节数相同
   D．一个类所能创建对象的个数是有限制的
3．下列关于成员函数的描述中，错误的是_____。
   A．成员函数的定义必须在类体外
   B．成员函数可以是公有的，也可以是私有的
   C．成员函数在类体外定义时，前加 inline 可为内联函数
   D．成员函数可以设置参数的默认值
4．关于构造函数，以下正确的说法是_____。
   A．定义类的成员时，必须定义构造函数，因为创建对象时，系统必定要调用构造函数
   B．构造函数没有返回值，因为系统隐含指定它的返回值类型为 void
   C．无参构造函数和参数为缺省值的构造函数符合重载规则，因此一个类中可以含有这两种构造函数
   D．对象一经说明，首先调用构造函数，如果类中没有定义构造函数，系统会自动产生一个不做任何操作的缺省构造函数
5．关于析构函数，以下说法正确的是_____。
   A．析构函数与构造函数的唯一区别是函数名前加波浪线~，因此，析构函数也可以重载
   B．当对象调用了构造函数之后，立即调用析构函数
   C．定义类时可以不说明析构函数，此时系统会自动产生一个缺省的析构函数
   D．类中定义了构造函数，就必须定义析构函数，否则程序不完整，系统无法撤销对象
6．执行以下程序后，输出结果依次是_____。

```
class test
```

```
{   int x;
    public:
      test(int a){  x=a;  cout<<x<<"  构造函数";  }
      ~test(){  cout<<x<<"  析构函数";  }
};
void main()
{   test x(1);  x=5;  }
```

A．1　构造函数　1　构造函数　5　析构函数　5　析构函数
B．1　构造函数　5　构造函数　5　析构函数　5　析构函数
C．1　构造函数　5　析构函数　5　构造函数　5　析构函数
D．1　构造函数　1　析构函数　5　构造函数　5　析构函数

7．下列表达方式正确的是_____。

A．
```
class P{
    public：
    int x=15；
    void   show(){cout<<x；}
     }；
```

B．
```
class P{
    public：
    int x；
    void show(){cout<<x；}
  }
```

C．
```
class P{
   int   f;
    }；
  f=25；
```

D．
```
class P{
  public：
    int a；
      void Seta (int x) {a=x;}
```

8．以下拷贝构造函数具有的特点中，错误的是_____。

A．如果一个类中没有定义拷贝构造函数时，系统将自动生成一个默认的
B．拷贝构造函数只有一个参数，并且是该类对象的引用
C．拷贝构造函数是一种成员函数
D．拷贝构造函数的名字不能用类名

9．下列关于友元函数的描述中，错误的是_____。

A．友元函数不是成员函数
B．友元函数只可访问类的私有成员
C．友元函数的调用方法同一般函数
D．友元函数可以是另一类中的成员函数

10．下列关于类型转换函数的描述中，错误的是_____。

A．类型转换函数是一种成员函数
B．类型转换函数定义时不指出类型，也没有参数
C．类型转换函数的功能是将其函数名所指定的类型转换为该类类型
D．类型转换函数在一个类中可定义多个

11．类体内成员有三种访问权限，说明它们的关键字分别是_____、_____和_____。

12．如果一个类中没有定义任何构造函数时，系统会_____。

13．静态成员是属于_____的，它除了可以通过对象名来引用外，还可以使用_____来引用。

14．友元函数是被说明在_____内的_____函数。友元函数可访问该类中的成员。

15．

```
#include<iostream.h>
class A
{  int x,y;
    public:
   A(int a,int b){x=a; y=b; cout<<"ABC"<<'\t';}
   A(){ x=3; y=4; cout<<"CBA"<<'\n';}
   void Show()
   { cout<<"x="<<x<<'\t'<<"y="<<y<<'\t';}
   ~A(){cout<<"XYZ"<<'\n';}
   };
void main(void)
{
   A *s1=new A(1,2),*s2=new A;  s2->Show();
   delete s1;  delete s2;
}
```

问题一：本程序的执行后输出的结果是_____。

问题二：如果将语句 s2->Show()改为 s1->Show()，则执行结果是_____。

16．

```
#include<iostream.h>
class A
{
  private:
     int x;
public:
   A(int a) { x=a; cout<<"x="<<x<<'\t'<<"class_A"<<'\n'; }
   ~A() { cout<<"class_~A"<<'\n'; }

};
class B
{  A y,z;  int s;
  public:
   B(int a,int b,int c): y(a+b+c),z(3-a)
    { s=c-b; cout<<"s="<<s<<'\t'<<"class_B"<<'\n'; }
   ~B() { cout<<"class_~B"<<'\n'; }
};
void main(void)
{  B s(1,2,3);  }
```

问题：本程序共输出_____行，其中第三、第四行分别是_____和_____。

17.

```
#include<iostream.h>
class node
{
    int x,y;
public:
    node(int a,int b)
      { x=a; y=b; cout<<"node_1"<<'\n'; }
    node( )
      { x=a.x; y=a.y; cout<<"node_2"<<'\n'; }
    void Show()
      {cout<<"x="<<x<<'\t'<<"y="<<y<<'\n'; }
};
void main(void)
{
   node f1(5,6);  node f2(f1);  f2.Show();
}
```

问题一：具有拷贝功能的构造函数 node()的参数表中缺少一个形参，这个形参的正确定义是_____。

问题二：node()中的形参被正确定义后，执行结果依次是_____。

18．以下 student 类用于查找考试成绩在 60 分以下的学生及其学号，并统计这些学生的总人数，请填空。

```
#include <iostream.h>
class student{
    int i,count;
    float *stu;
    int *num;
    int n,m;
public:
    student(int k){
        m=0;
        n=k;
        stu=new float[n];
        _______
        for(i=1;_______;i++)cin>>num[i]>>stu[i];
    }
    void stat(){
        for(i=1; i<n ; i++)
if(stu[i]<60)
                { _______;  Show();  }
    }
    void Show(){
        cout<<"number: "<<num[i]<<'\t'<<"grade: "<<stu[i]<<'\n';
```

```
    }
    void print(){
        cout<<"flunked total: "<<m<<endl;
    }
};
void main(){
    cout<<"请输入总人数: \n";
    int n; cin>>n;
    student a(n); a.stat();a.print();
}
```

19．按下列要求编程：

（1）定义一个描述矩形的类 Rectangle，包括的数据成员有宽（width）和长（length）；

（2）计算矩形周长；

（3）计算矩形面积；

（4）改变矩形大小。

通过实例验证其正确性。

20．编程实现一个简单的计算器。要求从键盘上输入两个浮点数，计算出它们的加、减、乘、除运算的结果。

21．编一个关于求多个某门功课总分和平均分的程序，实现一个有关学生成绩的操作，该类名为 Student。具体要求如下：

（1）每个学生信息包括姓名和某门功课成绩。

（2）假设 5 个学生。

（3）使用静态成员计算 5 个学生的总成绩和平均分。