面向对象程序设计案例精析

（C++语言）

庄波　著

山东大学出版社

面向对象程序设计案例精析
(C++语言)

庄　波　著

山东大学出版社

图书在版编目(CIP)数据

面向对象程序设计案例精析(C++语言)/庄波著. —济南:山东大学出版社,2009.9
ISBN 978-7-5607-3937-3

Ⅰ.面...
Ⅱ.庄...
Ⅲ.C语言—程序设计
Ⅳ.TP312

中国版本图书馆CIP数据核字(2009)第156473号

山东大学出版社出版发行
(山东省济南市山大南路27号　邮政编码:250100)
山东省新华书店经销
潍坊彩源国标印刷包装有限公司印刷
787×1092毫米　1/16　14.5印张　251千字
2009年9月第1版　2009年9月第1次印刷
定价:22.00元

前　言

对于一个初学者而言，建立面向对象的思维方式和提高面向对象的分析、设计能力无疑是十分重要的。但是，在多年的C++面向对象程序设计教学中，我发现这不是一个容易实现的目标。学生往往是空学了一身语法，解决实际问题时还是无从下手。当然，原因可能是多方面的，但其中比较重要的一点是：在教学中偏重语法知识的传授，不能很好地把面向对象的知识和解决实际问题结合起来。

2003年，我给2002级学生讲授C++。对于C++面向对象程序设计庞杂的内容，学生们有些摸不着头脑，一个学生甚至发出了类“真累”的感慨。我尝试引入一些案例，让学生在解决实际问题的过程中学习面向对象程序设计的知识。第一个引入的案例就是设计一个电子宠物猫。当时的效果很好，出乎我的意料，学生们甚至沉迷其中，乐此不疲，很快掌握了定义和使用类的方法。此后，我又设计了更多的案例，经过不断地调整和丰富，逐步形成了本书的内容。

本书以案例为载体，将面向对象的思维、方法融入解决实际问题的过程中，逐步培养和提高面向对象的分析和设计能力。本书收录了大小不等的10个案例，每个案例都完整地解决一个问题，通常包含多个知识点，但往往又有不同的侧重点，层层递进、逐步提高，涵盖了C++程序设计中的面向过程、面向对象、泛型编程以及标准模板库(STL)等各个方面，其中也涉及到内存管理、数据结构、测试驱动开发和设计模式等较深入的内容。

案例1　Score。一个学生成绩处理程序。可以学习自顶向下的设计方法、问题分析图(PAD)、结构体数组、冒泡排序和排名次算法、使用函数以及自定义头文件和多文件编译等知识。

案例2　Calendar。编写一个打印月历的程序。继续学习自顶向下的设计方法，重点是设计自定义函数的思路和方法。

案例3　Cat。有趣的电子宠物猫程序。重点学习类的设计、构造函数和成员函数的定义与实现。采用测试驱动的设计方法，变被动学习为主动思考，可谓是：小猫帮你学习“类”，一点也不“累”。

案例4　String。编写字符串类。学习构造函数、析构函数、拷贝构造函数、重载赋值运算符和输出运算符、深拷贝与浅拷贝等知识，重点是学习动

态内存管理技术。

案例 5 School。一个诙谐的校园故事。重点学习公有继承、基类和派生类的定义等，在轻松愉快的说笑中，带你跨入继承的世界。

案例 6 Vector。编写向量模板（类似 STL 中的 std::vector）。学习范型编程，动态存储管理，迭代器，插入、删除算法等，深入理解 std::vector。

案例 7 List。编写双向链表模板（类似 STL 中的 std::list）。学习范型编程，动态存储管理，链表插入、删除算法等，重点是学习自定义迭代器，进而加深对 STL 容器和算法的理解。

案例 8 Shape。一个命令交互式绘图程序。让学生学习简单绘图的同时，重点学习继承、虚函数、抽象类、运行时多态、文件流等知识的综合运用。用一种看得见的方式理解虚函数和运行时多态。

案例 9 Lottery。设计一个彩票过滤程序。学习面向对象的常见思维方法和基本设计原则，包括使用抽象类，理解“开放—封闭”原则和“单一职责”原则等。

案例 10 Payroll。一个面向对象的薪金发放程序。其中大量使用了继承和多态，还运用了命令模式和策略模式。重点是借助用例分析、测试驱动、设计模式等手段，采用面向对象的思维和设计方法分析和解决较复杂的问题。

案例的取材力求有用、有趣和有效。其中的成绩处理、月历打印、彩票过滤、工资发放等案例，与实际生活紧密相结合，内容实用。而宠物游戏、校园故事、图形绘画等都是十分有趣的项目，能够极大地激发学习的兴趣。另外，每一个案例都有深入细致的分析过程，最后都附有源程序，便于自学。而且在讲解过程中，特别突出了思维方法和设计方法，力求“授之以渔”，以达到提高学生能力的效果。

对于初学者，建议按顺序阅读其中的讲解部分，因为一方面很多知识点需要在做的过程中学习和领会；另一方面，我们往往不是一下子就能拿出最好的解决方案，而是在逐步改进的过程中得到的。在前后对比之下，学生会更容易理解和掌握，相反，蜻蜓点水式地阅读，很可能欲速则不达。

如果你已经具备一定的 C++基础，书中仍有一些设计思路等内容可能对你有所帮助，所以你可以有选择地阅读。

在本书的编写过程中，滨州学院的谭业武教授审阅了本书；计算机科学技术系的老师们阅读了部分案例，提出了许多宝贵的意见和建议；张士国教授对本书的编写工作给予了很多支持和帮助。本书也得到了滨州学院教研项目的支持。由我授课的 2005 级至 2008 级学生参与和实践了书中的大部分案例。在此对他们一并表示谢意！同时，感谢父母的养育和教诲，感谢岳父、岳母长期以来无私的支持，感谢妻子和女儿的陪伴，让我的生活永远充满了乐趣。

庄　波

2009 年 8 月 1 日

目　录

案例 1　Score

百川东到海，何时复西归。
少壮不努力，老大徒伤悲。
——《乐府诗集·长歌行》

1. 前　言

我们通过解决一个分析考试成绩的问题对 C/C++语言中基本控制结构、函数、数组、结构体以及条件编译和源代码组织等问题作一个回顾。你将会学习到自顶向下(top-down)的设计方法(在这方面我们将借助问题分析图(PAD)进行描述)、冒泡排序、使用函数、自定义头文件、快速输入大量数据等知识和技巧。

我们要解决下面这个老生常谈(但愿我能讲出一点新意)的考试成绩分析的问题。

输入 30 个学生的学号、姓名和 5 门课程的成绩，计算总分并按照总分排出名次，最后按照学号顺序打印成绩单。

我们采用循序渐进的方式来讨论，分为三个版本，每个版本集中于某一方面的问题。

- V0.1 一个 main()函数解决所有问题：自顶向下的设计，冒泡排序等。(2～7 节)
- V0.2 使用函数使程序结构清晰，使用全局变量。(第 8 节)
- V0.3 自定义头文件，隐藏实现细节。(第 9 节)

如果你认为某个版本对你来说比较简单，则可以略读甚至跳过，你也可以直接阅读附录中的源程序代码，遇到问题再回头寻找有关的讨论。

让我们开始吧！

2. 数据结构的设计

经常听到有人说感觉编程很困难，也经常看到许多人不假思索地写出错误的程序。其实，这都是缺乏设计或不良的设计导致的。如果设计充分的话，任何人都能写出专业的程序。

首先，我们进行数据结构的设计。我们需要描述30个学生的信息，很自然地，我们可以定义数组：

```
Student stu [30];
```

为了描述每个学生的信息，我们使用结构体类型 Student，因为我们无法使用简单的 int 或 char 之类的数据来描述一个学生的所有信息。为了描述一组相关数据组成的实体，我们最好使用结构体(struct)。这里为了描述一个学生的学号、姓名、5门课成绩、总分和名次，我们定义结构体类型 Student 如下：

```
struct Student {
    string sid;        //学号
    string name;       //姓名
    int score[5];      //成绩
    int total;         //总分
    int rank;          //名次
};
```

这就是我们设计的数据结构，如图1.1所示。

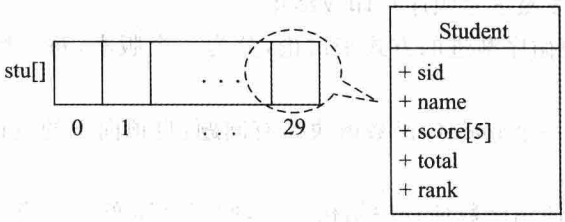

图1.1 数据结构的设计

这样一来，我们就可以用程序语言描述问题了。比如，“第 i 个学生”就是 stu[i]，“第 i 个学生的学号”就是 stu[i].sid，“第 i 个学生的第 j 门课成绩”就是 stu[i].score[j]，等等。

3. 自顶向下的算法设计

下面我们着重探讨一下算法的设计思路。所谓算法，就是解决问题的步骤。那么，如何设计解决问题的步骤呢？

有些人在遇到编程问题时会首先不假思索地写出下面的代码：

```
#include <iostream>
using namespace std;
int main()
{ }
```

接下来，光标停在最后一行，开始思考下一步怎么办。其实，这是缺乏设计的表现，在解决比较复杂的问题时，这种方法很难奏效，甚至可以说，用这种方法正确地写出一个 60 行的程序都是困难的。只有在充分设计的基础上，才能对解决问题的步骤有一个从整体到细节的把握，才能写出正确、漂亮的程序来。

那么，应该如何进行算法的设计呢？最常见的就是“自顶向下(Top-down)，逐步细化”的设计方法。也就是说，首先把待解决的问题分为几个大的步骤，然后针对每个步骤进一步细化为更小的步骤，直到能够用程序解决为止，最后，综合起来就形成了完整的算法。在此过程中可以借助问题分析图(PAD)等工具辅助我们进行直观的设计。

设计往往伴随着对问题的深入分析甚至是重新认识。“输入 30 个学生的学号、姓名和 5 门课程的成绩，计算总分并按照总分排出名次，最后按照学号顺序打印成绩单。”这其中已经说明了输入什么，作何处理，输出什么，大致可以分为 4 个大的步骤(这里没有固定或唯一的划分方法，视具体情况而定)：(1)输入 30 个学生的学号、姓名和 5 门课程的成绩；(2)计算总分；(3)按照总分排出名次；(4)按照学号顺序打印成绩单。

我们借助 PAD 图描述这 4 个步骤(见图 1.2)，为描述方便对其中每个步骤进行编号，如 1,2 等等。

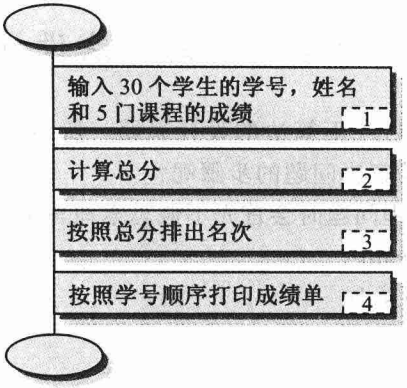

图 1.2　主程序的顶层设计

图 1.2 描述了最顶层的处理过程，接下来，我们要逐步地把每一步进行细化的设计。

比如其中的第 1 步“输入 30 个学生的学号、姓名和 5 门课程的成绩”，我们可以用一个 30 次的循环来完成（见图 1.3），同样，对每一个待细化的步骤也进行编号，如 1.1、1.2 等。

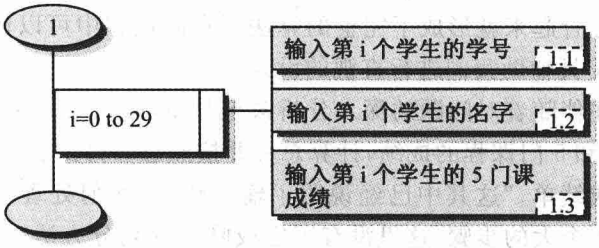

图 1.3　步骤 1 输入 30 个学生的学号、姓名和 5 门课程的成绩

图 1.3 中的 1.3 步“输入第 i 个学生的 5 门课成绩”也可以用一个循环来完成，如图 1.4 所示。

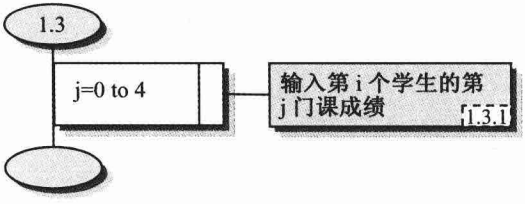

图 1.4　步骤 1.3 输入第 i 个学生的 5 门课成绩

这样的逐步细化可以进行到你能够容易地用程序实现为止。当然，我们只是对第 1 步的设计过程进行了详细展示，你应该领会到如何进行“自顶向下，逐步细化”的设计过程了。采用这种设计方法，可以使我们始终把注意力集中到一点上，从而把大问题分解为小问题，并逐步解决。对于第 2、4 步你可以尝试着自己解决(可参考图 1.5 把第 2 步分解得更详细些)。

最后，我们可以把逐步细化的结果代入到顶层设计中，产生详细设计的结果。比如，我们将 1.3 的细化结果代入步骤 1，再将步骤 1 代入顶层设计，其余各步骤以此类推，可以得到下面的参考结果。我们尽量使用程序语言来描述，并且用梯形框表示输入和输出，这样看起来更简洁一些。如图 1.5 所示。

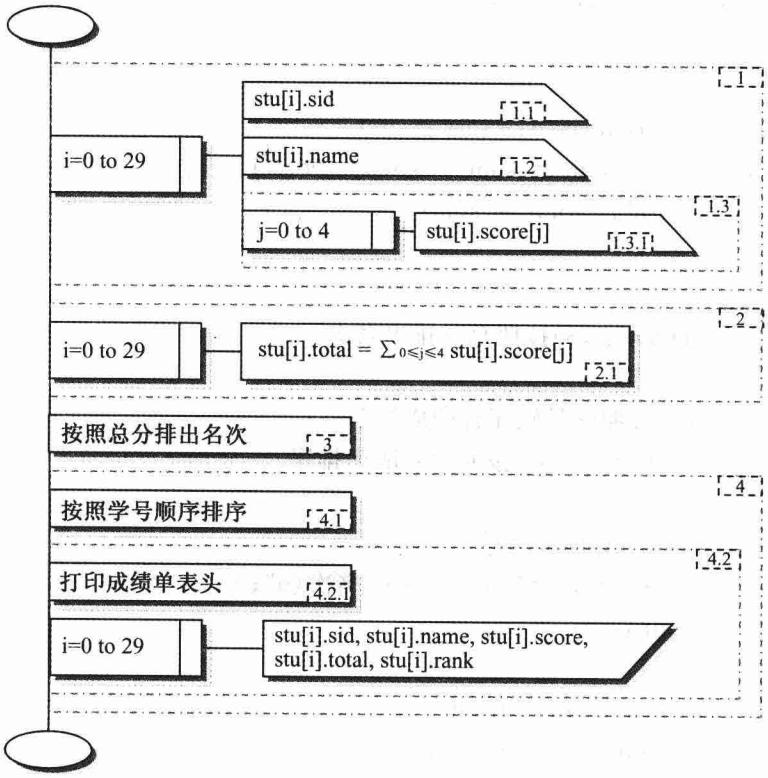

图 1.5 主程序的详细设计

下面，我们把这个图转换成程序代码。

```
int main()
{
```

```
//(1)输入 30 个学生的学号,姓名和 5 门课程的成绩
for(int i=0; i<30; i++)
{
    cin >> stu[i].sid;
    cin >> stu[i].name;
    for(int j=0; j<5; j++)
        cin >> stu[i].score[j];
}

//(2)计算总分
for(int i=0; i<30; i++)
{
    int sum = 0;
    for(int j=0; j<5; j++)
        sum = sum + stu[i].score[j];
    stu[i].total = sum;
}

//TODO:(3)按照总分排出名次

//(4)按照学号顺序打印成绩单
//TODO:(4.1) 按照学号顺序排序

//(4.2) 打印成绩单
cout <<"学号 姓名 成绩 总分 名次\n";
for(int i=0; i<30; i++)
{
    cout << stu[i].sid;
    cout << stu[i].name;
    for(int j=0; j<5; j++)
        cout << stu[i].score[j];
    cout << stu[i].total;
    cout << stu[i].rank << endl;
}
```

}

上面程序中的(3)和(4.1)尚未设计出具体的算法，所以仅仅使用注释作了说明，我们将在下面几节中继续讨论。

4. 排 序

下面让我们先讨论一下排序的问题，然后应用冒泡排序解决问题中的第4.1步“按照学号顺序排序”。

排序(Sort)是一个很常见的问题，我们常常按照事物某一个或某一些属性的大小将这些事物进行排序。比如，体育课上排队时，可能根据身高排序；同一宿舍中的同学可能根据年龄及生日排序；玩扑克牌时可能同时根据花色和面值排序。

排序的结果可能是从小到大，也可能是从大到小。排序过程往往需要多次调整记录的顺序，把那些顺序不合适的记录移动或相互交换。

排序的方法有很多，在“数据结构与算法设计”中还会系统学习，我们在这里使用可能是你目前最熟悉的一种——冒泡排序(Bubble sort)。(如果对冒泡排序已经足够熟悉，可以直接跳到本节末尾)

有时，我听到一些人抱怨记不住冒泡排序的程序怎么写。其实，我们没有必要把这些方法奉为圣经，这些方法都是人发明的。我们暂且不管叫什么排序方法，让我们通过一个简单的实例，体验一下发明一种“新的”排序方法的过程——从具体到抽象，设计排序算法。最后，我们具体应用该排序算法解决4.1“按照学号顺序排序”。

为简化起见，我们把6个数{4,2,8,5,7,1}从小到大排序。我们使用一种十分简单的规则，“考察相邻的两个元素，如果顺序不合适，我们就把它们交换过来”。一次处理过程如图1.6所示。

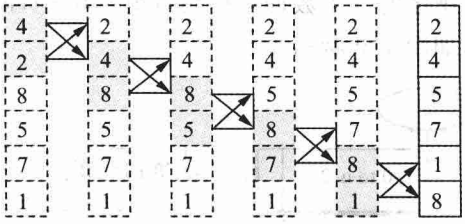

图1.6 第1趟排序过程

我们看到，经过这一处理，并未将所有元素按照从小到大的顺序排好。但是，也容易发现其中最大的元素已经排到了最后合适的位置。我们把这一处理过程称为“一趟排序”，可以说，经过一趟排序，至少可以排好最大的一个元素。

那么，接下来，我们对着还未排好的前 5 个元素进行一趟排序（见图 1.7）会怎样呢？

前 5 个中的最大元素 7 也排好了。

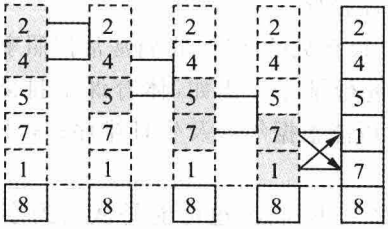

图 1.7　第 2 趟排序过程

于是，总共执行 5 趟排序过程就可以将整个序列从小到大排列好。下面列出了每一趟排序的结果，整个排序过程如图 1.8 所示。

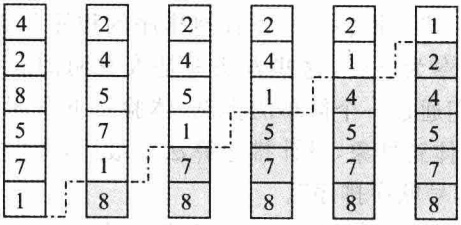

图 1.8　每趟排序过程

从这个具体的例子出发，我们总结一下整个排序方法。

我们不妨设待排序的 n 个数据保存在数组 a[0,…,n－1]中。整个排序过程需要 n－1 趟，如图 1.9 所示。

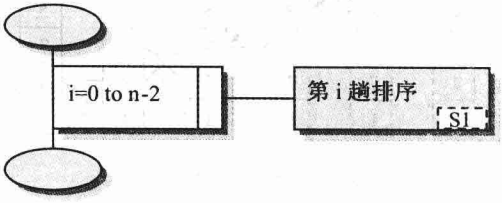

图 1.9　排序算法共 n－1 趟排序过程