面向对象程序设计攻关辅导

刘振安 尹雷
王岐捷 葛愿 编著

全国高等教育自学考试《面向对象程序设计》教材辅导参考书

中国科学技术大学出版社

面向对象程序设计攻关辅导

全国高等教育自学考试

《面向对象程序设计》教材辅导参考书

面向对象程序设计攻关辅导

刘振安　尹　雷

王岐捷　葛　愿

编著

相互对应的同步练习

深入浅出的例题分析

结合考试的模拟题型

加深理解的编程训练

走向实用的综合应用

简单实用的上机操作

详细清晰的实验示范

攻关考试的良师益友

中国科学技术大学出版社

2000·合肥

内 容 简 介

全书共分 13 章。本书的第一章至第八章的章节安排按自考指定教材《面向对象程序设计》一书的相应结构安排。每章先给出典型例题分析，然后按自考题型编制各种习题。各章的答案均附在章尾，以方便查阅。

应用实例两章的目的是带领读者进入实战状态，一方面进一步帮助读者了解应用方法，一方面也大大拓宽了知识面，不仅为课程设计打下良好的基础，还可以系统训练思考问题和解决实际问题的能力。它们对在校大学生及社会工程技术人员也大有裨益。BC 和 VC 上机指南则集中介绍了两种常见的语言工具的基本使用方法。实验指导则给出分别使用 BC 和 VC 编程环境对实验题目进行编辑并编译运行的实例，从而帮助读者尽快掌握编程工具。

本书可作为计算机自学考试的辅助学习教材，也可作为大专院校本科生的课外参考书及计算机培训教材；本书不仅能满足自学考试训练的需要，也能满足大专院校及社会上计算机专业培训学习 C--语言的需要。

图书在版编目(CIP)数据

面向对象程序设计攻关辅导/刘振安 等编著. 合肥：中国科学技术大学出版社，2000.11
ISBN7-312-01240-X

Ⅰ. 面… Ⅱ. 刘… Ⅲ. 面向对象语言-程序设计-高等教育-自学考试-自学参考资料
Ⅳ. TP312

中国版本图书馆 CIP 数据核字（2000）第 55003 号

中国科学技术大学出版社出版发行
(安徽省合肥市金寨路 96 号，230026)
中国科学技术大学印刷厂印刷
全国新华书店经销
开本：787×1092/16　　印张：18　字数：438 千
2000 年 11 月第 1 版　　2000 年 11 月第 1 次印刷
印数：1—3 000 册
ISBN7-312-01240-X/TP·263　　定价：22.00 元

前　言

面向对象方法的产生，是计算机科学发展的要求，和传统的程序设计方法相比，面向对象的程序设计具有抽象、封装和多态性等特征。其实，“面向对象”不仅仅作为一种语言、更作为一种方法论贯穿于软件设计的各个阶段。面向对象的技术在系统程序设计、数据库及多媒体应用等领域都得到广泛应用。专家们预测，面向对象的程序设计思想将会主导今后程序设计语言的发展。

为了适应新世纪计算机教学的要求，全国高等教育自学考试指导委员会决定在计算机及应用专业（独立本科段）开考《面向对象程序设计》课程。笔者有幸根据面向对象程序设计自学考试大纲主编了该教材和自学辅导书，并精心设计了本习题集作为教学参考书与之配套。原因是学习C++的关键是面向对象的思维方法，我们必须从原来已经比较熟悉的，面向过程的设计方法中转变思想，接受面向对象的编程思想。这种思想方法的转变是需要花大力气的，尤其需要下定决心抛弃原来的编程习惯和思考方法。而且自学考试的特点是试题题型固定，要求掌握内容的深度明确，但试题覆盖面广。为了帮助考生更好地掌握教学内容，理解考试大纲的要求及规定的考试题型，适当增加试题类型训练也是很有必要的。在当前的素质教育中，更要求学生学会应用所学去解决实际问题。因此，我们不能仅仅从教科书中学习，从考试的角度学习，而应明确我们学习的目的最终是为了应用。因此，能有一本适当开拓视界的例题练习参考书，必然会在帮助读者深入理解教科书内容的过程中起到重要作用。

为了能满足自考的需要，本书的第一章至第八章的章节安排按自考指定教材《面向对象程序设计》一书的相应结构安排。每章先给出典型例题分析，然后按自考题型编制各种习题。为了加深对课文的理解和应用，增加一定数量的编程题。各章答案均附在章尾，以方便查阅。第九章是简单应用实例，第十章是综合应用实例。这两章的目的是带领读者进入实战状态，一方面进一步帮助读者了解应用方法，一方面也大大拓宽了知识面，不仅为课程设计打下良好的基础，还可以系统训练思考问题和解决实际问题的能力。它们对在校大学生及社会工程技术人员也大有裨益。第十一章是BC上机指南，第十二章是VC上机指南，这两章集中介绍了两种常见的语言工具的基本使用方法。第十三章的实验指导，则给出分别使用BC和VC编程环境对实验题目进行编辑并编译运行的实例，从而帮助读者尽快掌握编程工具。

本书不仅能满足自学考试的需要，也能满足大专院校及社会上计算机专业培训学习C++语言的需要，更是帮助训练动手能力的好助手。

我校95,96,97和98届一些本科生及研究生均为本书花费了大量的时间和精力，精心组织习题、仔细进行校对、修改及程序验证。特别是张蕊、刘大路、李丰君和蒋里，在这段时间里工作任劳任怨，认真负责。

本书在编写过程中还参考了大量书籍及文献资料，在此向被引用资料的作者及给予帮

助的所有学者再次表示感谢。

由于本人水平有限，不妥之处在所难免，希望同行及读者指正。

刘振安

2000年11月28日于合肥

中国科学技术大学

目　　次

第一章 面向对象及C++基础知识

因为考试大纲要求考生学习过C语言，所以教材中没有再介绍与C语言雷同的知识。为了方便学习，我们将在本章的习题集中给出一定数量的例题，以便先复习并巩固一下最基本的C++知识，为掌握指针及引用等问题打下基础。

1.1 例题分析

【例 1.1】 编写一个输入两个整数，输出其和的典型 C++示例程序，并通过这个程序说明C++程序的典型构造形式。

```
#include <iostream.h>                                    // 系统头文件
void main( )
{
  int a, b;
  cout << "Enter two integer:";
  cin >> a >> b;
  int result;
  result = a + b;
  cout << " \ nThe sum of " << a << "+" << b
  << " = " << result << endl;
}
```

假设我们要求 25 + 50 的和，则运行过程及结果如下：

Enter two integer:25 50

The sum of 25 + 50 = 75

现在，我们结合这个程序对其程序结构说明如下：

1.1.1 注释

程序中的符号“//”是注释符号，与 C 语言中的符号“/* */”的作用相同，都是使编译器跳过程序中的注释，不处理注释的内容。使用注释来描述程序的功能，完全是为了增强程序的可读性并提供理解程序的线索。

符号“ // ”告诉编译程序，本行“// ”之后的所有内容都是注释；而符号“ /* ”和 “*/ ”告诉编译器在“/*”和“*/”之间的内容都是注释。注释符号“// ”适合短的只占一行的注释，而“/* ...*/”适合于长的占用多行的注释。

程序文件的书写格式是自由的，空格起分隔单词的作用(多个空格符仅被当作一个空格符看待)。在上面的程序中，通过恰当地使用空格和空行，也可增强程序的可读性。

1.1.2 包含文件及头文件

C++语言包含文件的格式有两种。以 Borland C++为例，第一种为：

```
#include <文件名.扩展名>
```

编译器并不是在当前目录查找，而是根据目录出现的顺序分别查找各路径。尖括号中的空格被认为是文件名的一部分。这种包含方法常用于标准头文件。例如 stdio.h、string.h 等，均是 Borland C++的组成部分。第二种为：

```
#include "文件名.扩展名"
```

这使得编译器首先在当前目录中查找，然后像上一种方式那样在标准目录中查找。因为在引号中的空格被认为是文件名的一部分，所以应避免可能引起错误的空格。

iostream.h 是 C++的系统文件，经常称其为头文件。而

```
#include <iostream.h>
```

行的作用是指示 C++编译器将文件 iostream.h 的内容插入到程序中#include 指令所在的这一行的后面，这使得程序可以使用在文件 iostream.h 中定义的标准输入和标准输出操作。只有这样，语句 cout 和 cin 才能被正确编译。

类标识符在使用之前，必须先进入作用域，这通常是用类声明来实现的。使用类的公有成员时，类标识符不仅要在活动作用域中，而且必须被完全声明。因为每个类都有自己的声明部分，而且各种模块都可以使用它，所以常把声明置于头文件中，用到该类的每个模块再包含相应的头文件。

假设一个典型的程序是由大量类和头文件组成的，常常会遇到在同一个文件中，一个头文件可能被包含多次。例如：

```
                                                    // file HEADER.HPP
class EssentialClass { /* ... */ };
                                                    // file DESKTOP.HPP
#include "HEADER.HPP"
class DeskTop { /*  ...  */ };


                                                    // file DRAWER.CPP
#include "HEADER.HPP"
#include "DESKTOP.HPP"                              // file HEADER.HPP included  twice !
class Drawer { /*  ... */ };
```

为避免这种错误，C++头文件用一些预处理指令产生下面的简单而有效的分离构造：

```
                                                    // file HEADER.HPP
#ifndef HEADER.HPP
```

```
#define HEADER.HPP
class EssentialClass { /    ...    / }
...
#endif
```

所有的头文件都采取相同的机制。通常在#define 语句中使用头文件的文件名或其省略形式。这样试图再次包含同一文件的错误就可以被预处理器发现并处理。

1.1.3 语句

在C++语言中，以分号结尾的句子称为语句。例如：

```
int a,b;
```

分号代表这个语句的结束。如果这一行的结尾没有分号，则这一个语句还没有结束。有时一个长的语句可以占不止一行，为了提醒读者注意这些行属于一个语句，应该以醒目的方式断行并在下一行往右缩排。例如：

```
hWnd =   CreateWindow(
           szClassName,
           szTitle,
           ...,
         NULL
         );
```

对于短的语句，也可以一行写多个语句。例如：

```
int a; float fp; char sc;
```

1.1.4 标准输入与输出

cout 和 cin 称作标准输出 / 输入流，在 iostream.h 中定义，它表示标准输出 / 输入设备，标准输出一般指的是屏幕，标准输入指的是键盘。

运算符<< 把它的右边内容在屏幕上显示出来(变量与表达式的值或由双引号括起的字符串)。运算符>>将键盘中输入的一个数，送到它右边的变量中保存起来。上述程序运行之后，在屏幕上可以看到下面这条信息：

```
Enter two integer:
```

光标在冒号后面闪烁，等待输入两个数。例如输入第一个数 25，按一下空格键，再输入第二个数 50，然后按回车键，这时程序就读入所输入的数，25 送给 a，50 送给 b，然后开始执行以后的语句。

1.1.5 新行

在程序中，“\n”称为新行符。C++系统还提供一个操纵算子 endl，它的功能和新行符一样，也是开始一个新行。

1.1.6 主函数

这个程序中以 main 开始的部分定义了一个函数，该函数规定了该程序的功能。main 是函数名，在函数名之后紧跟一对圆括号。所有的 C++程序都必须有一个名为 main 的主函数。这是程序员和 C++系统之间的约定：程序执行的开始点是 main 函数中的第一条语句。

一个 C++函数中的任何成分被括在一对花括号（“{"和"}”）中，在函数 main 的后面的右圆括号后紧跟一个左花括号，表示“这个函数从这里开始”，最后的右花括号表示“这个函数在这里结束”，花括号括起来的部分称作函数体，而函数名 main 和它后面的一对圆括号称为函数头。函数体由一系列的 C++语句组成，这些语句描述这个函数怎样实现它的功能。

【例 1.2】分析下面程序的输出结果。

```
#include <iostream.h>
void fun ( )
{
    static int i = 25 ;
    i++ ;
    cout << "i=" << i << endl ;
}

void main ( )
{
    for (int j = 0 ; j < 2 ; j++)
    fun ( ) ;
}
```

静态变量可以是局部变量或全局变量，但都具有全局寿命。在说明静态变量时，如果未指定初始化表达式，则其初始化表达式为 0。 静态全局变量的初始化是在程序开始执行主函数 main ( )之前完成，静态局部变量的初始化则在程序运行中第一次经过它时进行。在上述程序中，i 的初始化是在程序运行中第一次经过它时进行的，即 i=25,所以程序的输出是：

```
i = 26
i = 27
```

【例 1.3】分析下面程序的输出结果。

```
# include <iostream.h>
void main ( )
{
    int i = 0 ;
    while ( i++ < 3 )
    for (int j = 0 ; j < 5 ; j + +)
    cout < < j < < " " ;
}
```

程序是在一个 while 循环语句的循环体中使用另一个 for 循环语句，构成嵌套循环。程序输出结果为：0 1 2 3 4 0 1 2 3 4 0 1 2 3 4

【例 1.4】 结合下面的程序，说明 sum( )函数声明、调用和定义的方法。

```
#include <iostream.h>
void main( )
{
    int a,b,c;
    int sum(int, int);                          // 用原型声明 sum( ) 函数
    a=25;
    b=36;
    c=sum(a,b);                                 // 调用 sum 函数
    cout << c << endl;
}

int sum(int x, int y)                           // 定义 sum( )函数
{
  int temp;
  temp = x+y;
  return temp;
}
```

下面结合该例说明如下：

1. 定义函数

函数定义的一般形式如下：

```
类型  函数名（参数表）
{
    函数体
}
```

函数定义指定了一个函数名和函数类型，函数名是一个有效的 C++标识符，函数的类型规定为该函数返回值的类型，它可以是除数组和函数之外的任何有效的 C++数据。参

数表可以为空，这表明函数不需要从调用者那里接收数据。既使参数表为空，函数名后的一对圆括号也不能省略。

花括号内括起的函数体是一个语句序列，用于描述这个函数所要执行的操作。当函数返回一个值时，在这个函数体中必须有一个 return 语句。

需要注意的是，函数 sum()的定义在参数表中进行变量说明时，每个变量必须分别指定类型和名字。下述说明方法是不正确的：

```
int sum (int x, y)                    // 错误
```

2. 函数原型

函数原型标识一个函数的返回类型，同时也标识该函数参数的个数和类型。C++ 编译器从一个函数定义中抽取该函数的函数原型。程序员也可以在程序中使用函数说明语句来说明一个函数的原型。函数说明语句的一般形式为：

类型　函数名（参数类型说明列表）;

其中“列表”是用逗号隔开的一个个类型说明，其个数和指定的类型必须和函数定义中的参数的个数和对应类型一致。例如:

```
int sum (int, int) ;
```

函数原型的重要作用是可以使编译器检查一个函数调用表达式中可能存在的问题。在函数说明中也可以给出参数名，例如：

```
int sum (int first, int second);
```

名字 first 和 second 对编译器没有意义，但如果取名恰当的话，这些名字可以起到说明参数含义的作用，以帮助程序员正确掌握函数的使用方法。

3. 函数调用

函数调用是由函数名和函数调用运算符（ ）组成的一个表达式，其一般形式为：

函数名（实参表）

实参表是用逗号隔开的一个表达式列表，其中的每个表达式的值称为实参。在函数调用时，实参的值传给形参；在实参表中，实参的个数必须和形参的个数相同，实参的类型必须和对应的形参的类型一致。函数调用表达式的类型为在函数定义中为该函数指定的类型， 如果其类型不为 void 的话，这个表达式值就是函数定义中的 return 语句所返回的值，否则它表示一个无值表达式。

当在函数定义中没有指定形参时，调用表达式中的实参表为空，但函数调用运算符不能缺省。例如:

```
int value( );
void main ( )
{
    cout < < value ( );
}

int value ( )
{
```

```
    return 55 ×8 ;
}
```

4. 函数 return 语句的作用

任何一个 C++程序都必须定义一个 main 函数，它的返回类型总是 int 类型。这个函数由操作系统来调用，在 main 函数执行完以后，程序也就终止了。main 也可以使用 return 向操作系统返回一个值。可使用操作系统的命令检查 main 的返回值。一般约定在 main 返回 0 时，表示程序运行过程中没有出现错误，其它非零值表示程序出现异常情况。如果没有为 main 指定返回值，则返回值是任意的。

对于不返回值的函数，函数返回类型指定为 void，这时，return 语句中不能带有表达式，但可以在其后加一个分号，形成一个调用表达式语句或将这一条 return 语句省略。

【例 1.5】 编写一个计算平方根的程序，要求能判断所输入的值是否合理。

我们编写一个函数 input，用来请求用户输入一个浮点数，然后使用函数 sqrt()求这个数的平方根，并将其值返回到调用者那里。该函数同时检测用户是否输入了一个负数，若是，显示一条错误信息，并使用函数 exit( )立即终止程序的执行。

```
#include <iostream.h>
#include <math.h>
#include <stdlib.h>
void main ( )
{
    float root;
    float input( );
    root=input( );
    cout << "The square root = " << root <<endl;
}

float input ( )
{
    float x;
    cout <<"Enter a real number:";
    cin >> x;
    if  ( x < 0.0) {
    cout << "Input < 0 " << endl;
    exit(1);
    }
    return sqrt(x);
}
```

函数 abort( )和 exit( )的作用一样，不同的是 exit( ) 函数在程序终止之前要作一些清理工作，例如，关闭该程序中已打开的文件，并调用有关的退出函数或析构函数；而函数

abort( )却不做这些工作，它仅有的作用是终止程序的运行。调用函数 abort 时不需要提供参数。它在 stdlib.h 中的说明为：

```
void abort( );
```

【例 1.6】我们说的一个标识符的作用域是什么意思？作用域分为哪五种？试举例说明文件作用域。

解答：一个标识符的作用域是程序中的一段区域，用于确定该标识符的可见性。作用域分为五种：块作用域（局部作用域）、文件作用域（全局作用域）、函数原型作用域、函数作用域和类作用域。

具有文件作用域的变量是在所有块、函数和类之外说明的标识符，其作用域从说明点开始，在文件结束处结束。如果标识符出现在头文件的文件作用域中，则它的作用域扩展到嵌入了这个头文件的程序文件中，直到该程序文件结束。文件作用域包含该文件中所有的其它作用域。在同一作用域中不能说明相同的标识符。标识符的作用域和其可见性经常是相同的，但并非始终如此。例如：

```
#include <iostream.h>
int i;                                          //文件作用域
void main ( )
{
    i=5;
    {
        int i ;                                 // 块作用域
        i = 7 ;
        cout << "i=" << i <<endl;               // 输出 7
    }
    cout << "i=" << i ;                         // 输出 5
}
```

在这个程序中，最外层的 i 有文件作用域，最内层的 i 有块作用域，最内层的 i 隐藏最外层的 i，这时在最内层无法存取文件作用域的 i。

通过使用作用域运算符 :: ，可以在块作用域中存取被隐藏的文件作用域中的名字。例如：

```
#include <iostream.h>
int i;                                          // 文件作用域
void main ( )
{
    i=5;
    {
      int i ;                                   // 块作用域
      i = 7 ;
      ::i=1;
```