全国高等教育自学考试应试指导丛书
中国计算机函授学院图书编写中心 组编

计算机及应用专业（本科）

面向对象程序设计自考应试指导

主　编　刘振安
副主编　胡学联

南京大学出版社

中国计算机函授学院图书编写中心　组编

全国高等教育自学考试应试指导丛书

计算机及应用专业(本科)

面向对象程序设计自考应试指导

主　编　刘振安

副主编　胡学联

作者　刘振安　胡学联

徐　菁　孙　忱　戴翌斐

南　京　大　学　出　版　社

内容简介

全书共分两大部分。第一部分的第1章至第9章是根据自考指定教材《面向对象程序设计》一书的相应结构来安排的。每章针对大纲中的考核点，高度凝炼出教材中的知识点，并组织了大量典型例题进行分析解答，同时给出了教材中的所有习题答案。第10章的实验指导，可以帮助读者尽快克服试验环节的困难，通过上机编程加深对概念的理解。第二部分中，编者组织了一套模拟试题供读者自测。

本书旨在帮助考生充分认识考试内容和题型，做好应试准备，取得理想成绩。当然，它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学习C++语言的需要，更是帮助训练动手能力的好助手。

图书在版编目(CIP)数据

面向对象程序设计自考应试指导/刘振安编著.—南京：南京大学出版社，2001.3
ISBN 7-305-02185-7

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(2001)第10733号

书　　名　面向对象程序设计自考应试指导
主　　编　刘振安
副 主 编　胡学联
丛书主编　牛允鹏
责任编辑　史德芬
出版发行　南京大学出版社
地　　址　南京汉口路22号　邮编　210093　电话　025-3593695
印　　刷　合肥学苑印刷厂
经　　销　全国各地新华书店
开　　本　787×1092(mm) 1/16　印　张12.5　字　数　300千字
版　　次　2001年3月第1版　2001年3月第1次印刷
定　　价　18.00元
ISBN 7-305-02185-7/TP·215

组编前言

国家教育部考试中心于2000年开始，正式执行自学考试新计划，同时使用新编的大纲和教材。

为适应新调整的考试计划及密切配合新大纲新教材开展助学辅导，中国计算机函授学院利用多年积累的自考教学辅导资源和经验，全面系统地剖析了本专业各门专业课程新大纲和教材的内容体系，重新组织编写了一套“全国高等教育自学考试计算机及应用专业应试指导”丛书，推向全国，以满足考生之急需，适应社会之需要。

这套丛书堪称“通关必读”，其主要特征是：

首先，担纲编写应试指导丛书的作者基本上都是该专业全国自考指定教材及大纲的主编。

其次，自考应试指导丛书的作者，都在书中融入了自己多年从事该专业自考教学辅导的直接经验。他们既是本专业的教授，又是自考辅导的专家，二者集于一身，有些作者就是当年在中央电视台担任自考辅导教学讲座的教授。

最后，精心组织、细心筹划、用心编撰，是这套丛书的又一质量保证。

编写该套丛书的指导思想是，切实解决考生自学应试中的三个问题：

(1)在自学过程中起到答疑解惑作用，帮助考生顺利阅读、掌握教材内容；

(2)帮助考生抓住课程重点、难点，不入迷津；

(3)帮助考生理清课程主线，建立清晰的知识结构体系，在掌握知识点的前提下，沉着应战，顺利过关。

较之其它专业而言，计算机及应用专业自学考试是有一定难度的，因此，请一位好“教师”，找一位好“辅导”，尤为重要。这套“自学考试指导”丛书，可望成为你攻克一门又一门课程，克服一个又一个难关的良师益友；帮助你扫清学习中的障碍，增强你的必胜信心，伴随你走向成功的彼岸。

我们真诚地为计算机及应用专业的广大考生奉献这份精品、真品。愿广大考生早成夙愿。

2000年1月

编者的话

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，特编写了本书。

本书章节结构和辅导内容均与教材一致，重点是结合考试大纲的要求，先介绍知识点，然后针对自学难点和容易混淆的概念，精选典型例题加以说明。为了使考生熟悉考试大纲规定的考试题型，给出了教材习题的分析和参考答案。第10章的实验指导，可以帮助读者尽快克服试验环节的困难，通过上机编程加深对概念的理解。

学习C++的关键是面向对象的思维方法，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，必须牢固掌握C++语言的基本结构和主要特征，而C++语言的独有特征及其与C语言的区别则是学习过程中的重点和难点。

在自学时要注意与教材同步，注意每一章的考核知识点以及各种题型的解题能力。必须熟悉数据封装的特性与应用，才能抓住面向对象程序设计的实质。对C++语言的对象和类必须深刻理解，重点掌握。掌握构造函数与析构函数在对象创建及撤消过程中的作用，掌握对象的各种初始化方法。

继承在面向对象程序设计中可以说是最为重要的一环，学习面向对象中的继承特征，不仅要学习其观点与方法，更重要的是学习如何在实际编程中加以应用。虚函数与多态性是学习的重点，也是难点，难点在于它必须同时考虑类的数据封装与继承关系。

以上内容知识点多，也是试题的精华之处。不仅应掌握辅导材料给出的题型变化，还要掌握分析问题的方法。为了提高应试能力，我们还编写了包含各种题型的模拟试题，只要按照要求吃透并掌握各种题型的分析解题能力，就能具有应变能力，顺利通过考试。

参加本书编写的主要有刘振安、胡学联、徐菁、孙忱、戴翌斐、章守信、田钢、蒋里、颜廷荣、尹雷、葛愿、彭程、刘胜璞、武昕、陈志雄、弓岱伟、吴倩和李佳等。

由于这门课程还没有开考，而且又是新开设的课程，所以对社会考生的水平还难以推测。尽管如此，我们还是要按考试大纲，要求考生脚踏实地准备应试，只要准备充分，也是不难考出好成绩的。预祝每一位自考者成功！

刘振安

2000.12.8 于合肥

目　录

第一部分　知识点与典型题解 …… (1)
第 1 章　面向对象及 C++ 基础知识 …… (2)
1.1　知识点 …… (2)
1.2　典型例题分析与解答 …… (4)
1.3　本章习题解析 …… (12)
第 2 章　类和对象 …… (15)
2.1　知识点 …… (15)
2.2　典型例题分析与解答 …… (19)
2.3　本章习题解析 …… (29)
第 3 章　构造函数与析构函数 …… (40)
3.1　知识点 …… (40)
3.2　典型例题分析与解答 …… (42)
3.3　本章习题解析 …… (52)
第 4 章　继承和派生类 …… (59)
4.1　知识点 …… (59)
4.2　典型例题分析与解答 …… (61)
4.3　本章习题解析 …… (70)
第 5 章　多态性和虚函数 …… (81)
5.1　知识点 …… (81)
5.2　典型例题分析与解答 …… (83)
5.3　本章习题解析 …… (92)
第 6 章　进一步使用成员函数 …… (101)
6.1　知识点 …… (101)
6.2　典型例题分析与解答 …… (102)
6.3　本章习题解析 …… (115)
第 7 章　运算符重载及流类库 …… (126)
7.1　知识点 …… (126)
7.2　典型例题分析与解答 …… (127)
7.3　本章习题解析 …… (137)
第 8 章　模板 …… (152)
8.1　知识点 …… (152)
8.2　典型例题分析与解答 …… (154)
8.3　本章习题解析 …… (161)
第 9 章　面向对象程序设计基础 …… (169)

9.1 知识点 …… (169)
9.2 典型例题分析与解答 …… (170)
9.3 本章习题解析 …… (171)
第10章 实验指导 …… (175)
10.1 使用BC环境的实验方法 …… (175)
10.2 使用VC环境的实验方法 …… (180)
第二部分 模拟试卷分析与解答 …… (186)
模拟试卷 …… (187)

第一部分

知识点与典型题解

本部分紧扣全国高等教育自学考试委员会公布的考试大纲，以考核知识点与考核要求为主线，结合作者多年来的教学及辅导经验，从整体上对本课程的结构及内容进行规划、编排。该部分的每章又分为三个小部分：知识点、典型例题分析与解答、本章习题解析。

知识点部分最能体现作者对教材的认识与理解，是作者多年教学辅导工作的结晶，以简洁、通俗的语言，对难以理解的内容进行了归纳与总结。典型例题分析与解答部分反映了作者对自考大纲的准确把握。这里并不注重题量的多少，但在题目的典型性、题型选择、难易程度的把握等方面，力求与自考大纲贴近。通过对这些内容的学习，可以培养考生融汇贯通、举一反三、灵活运用的能力，从而达到增强考生应试能力的目的。习题解答部分对自考指定教材上的习题进行了详细分析与解答，使考生不仅能够知其然也能知其所以然。

总之，以上三部分内容各有特色，相互配合，相互补充，基本上能满足考生在学习上的需要。同时，更为考生解决了无师可求、难题不能解决的问题。

第1章　面向对象及C++基础知识

本章重点是面向对象设计的基础知识、C++语言对C语言的修改所呈现的新面貌及突出的一些新特点和新思想，应掌握基本程序结构、类型修饰符、函数原型、内联函数、引用、编译指令和一些其它基础知识。

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，我们在本章中给出了一定数量的例题，以便复习并巩固一下最基本的C++知识。

1.1　知　识　点

1.面向对象程序设计基础知识

面向对象的程序设计方法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、继承和多态性等特征。C++是在标准C语言的基础上，引入“面向对象”概念而扩充形成的混合型面向对象语言。C++语言不是一种纯面向对象的语言，即C++程序中还要使用像main( )这样的全局函数。

(1)抽象

面向对象语言鼓励程序员以抽象的观点看待程序，即程序是一组抽象的对象组成的。程序员以一组对象为起点，抽取公共的行为将其放入到一个类中，这基本上是抽象分类的观点，不同类的对象具有不同的行为。

(2)封装

封装要求一个对象应具备明确的功能，并具有接口以便和其它对象相互作用。同时，对象的内部实现(代码和数据)是受保护的，外界不能访问它们，只有属于这个对象的代码才可以访问该对象的内部数据。对象内部数据结构的不可访问性称为数据隐藏。封装使得一个对象可以像一个部件一样用在各种程序中，同时也切断了不同模块之间数据的非法使用，减少了出错的可能。

(3)继承

简单地说，继承就是一个对象可以获得另一个对象的特征。例如，国光苹果继承苹果的特征。

(4)多态性

不同的对象可以调用相同名称的函数，并可导致完全不同行为的现象称为多态性。利用多态性，程序中只需进行一般形式的函数调用，函数的实现细节留给接受函数调用的对

象。这大大提高了我们解决复杂问题的能力。

2. C++基本程序结构

下面是一个简单的C++程序：

```
#include <iostream.h>
void main( )
{
  char * name;
  cout << "Please input your name: ";                    // 输出信息
  cin >> name;                                           // 输入名字
  cout << "Hello," << name << "!" << endl;               // 使用 endl 换行
}
```

(1) 头文件

第一行包含了一个头文件 iostream.h。有了它，我们就可以使用C++风格的输入/输出。

(2) 输入/输出

C++自带输入和输出，并且可以根据数据的类型，自动地使用合适的输出方式。C++自带的输入和输出分别是"cin >>"和"cout <<"。如果把"<<"和">>"看作表示方向的符号，我们就会发现，数据确实是在按照一定的方向流动，这就是"流"这个名称的由来。

(3) 注释

行注释方式从"// "开始，直到行尾。例如：

```
i++;                    // i=i+1
```

一般情况下，多行注释仍旧使用"/*……*/"，而短的注释则较多使用行注释方式"//"。

(4)使用新的换行符

在语句"cout << "Hello," << endl;"中，C++使用 endl 换行，这个换行符号也可以用在其它位置。例如：

```
cout << "I" << endl << "You" << endl << "He" << endl;
```

上面语句的输出如下：

```
I
You
He
```

3. 类型修饰符 const

类型修饰符 const 声明的变量必须在声明时被定义(即初始化)，并且它的值在程序中不能被改变，只能被读取。也就是说，一旦加上 const 修饰符，编译器就将其视为一个常量，不再为它分配内存，并且每当在程序中遇到它时，都用在说明时所给出的初始值取代它。

4. 使用函数原型和缺省参数

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。任何函数，如果缺少了函数原型，C++都将无法编译。函数原型使得C++能够提供更强的类型检

查，将函数调用表达式中可能存在的问题发现在编译阶段。

缺省参数就是不要求程序员设定该参数，而由编译器在需要时给该参数赋予预先设定的值。

5. 使用新的动态内存分配函数

运算符 new 用于进行内存分配，运算符 delete 释放 new 分配的内存。

6. 内联函数

将一个函数定义为内联函数，只要定义时在函数名前加上关键字 inline 即可，例如：

```
inline float Myabs(float x)
{
  return x<0 ? (-x):x ;
}
```

这样 C++ 编译器在遇到对函数 Myabs 进行调用的地方，就用这个函数的函数体进行替换。

7. 引用

所谓引用，就是给变量起一个别名，换句话说，是使新变量和原变量共用一个地址。这样，无论对哪个变量进行修改，其实都是对同一地址的内容进行修改。因此变量和对它的引用总是具有相同的值。

通过引用传递方式，我们不用指针也能改变实参的值。

8. 编译指令

在 C++ 源程序中，可包含各种编译指令，以指示编译器对源代码进行翻译之前先对其进行预处理。所有的编译指令都以 # 开始，每条指令单独占用一行，同一行不能有其它编译指令和 C++ 语句(注释例外)。编译指令不是 C++ 的一部分，但扩展了 C++ 编程环境的使用范围，从而改善程序的组织和管理。

条件编译指令是 # if, # else, # elif 和 # endif，它们构成类似于 C++ 的 if 选择结构。其中编译指令 # if 用于控制编译器源程序的某部分有选择地进行编译，endif 表示一条指令结束。该部分从 # if 开始，到 # endif 结束。如果 # if 后的常量表达值为真，则编译这部分，否则就不编译该部分，这时，这部分代码相当于被从源文件中删除掉。

关键字 defined 用于判定一个标识符是否已经被 # define 定义。如果标识符 identifier 已被 # define 定义，则 defined(identifier)为真，否则为假。C++ 鼓励程序员使用 defined 进行标识符的测试。

1.2 典型例题分析与解答

1 编写一个输入两个整数，输出其和的典型 C++ 示例程序。

```
# 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 << "\n The sum of " << a << "+" << b
  << " = " << result << endl;
}
```

【解析】假设我们需要计算 35 + 20 的和,则运行过程及结果如下:

```
Enter two integer: 35 20
The sum of 35 + 20 = 55
```

注意:不要忘记包括系统头文件 iostream.h。

2 下面程序使用了 static 变量,试分析程序的输出结果。

```
# include <iostream.h>
void fun ( )
{
  static int i = 25 ;
  i++ ;
  cout << "i=" << i << endl ;
}
void main ( )
{
  for (int j = 0 ; j < 2 ; j++ )
    fun ( ) ;
}
```

【解析】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
```

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 < < " " ;
}
```

【解析】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

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;
}
```

【解析】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。但即使参数表为空,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。下面的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与函数定义中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 ( );          // 函数调用运算符不能缺省
}
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，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。

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。

【解析】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。整个程序清单如下：

```
# 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( );
```

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5种？试举例说明文件作用域。

【解析】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，其作用域是从说明点开始，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变量。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;
      cout << "i=" << i <<endl;                 // 输出 7
    }
  cout << "i=" << i ;                           // 输出 1
```

```
}
```

变量可以在文件作用域中说明，也可在块作用域中说明，在文件作用域中说明的变量称全局变量，在块作用域中说明的变量称局部变量。函数只能在文件作用域中进行定义，但可以在文件作用域或块作用域内进行函数说明。

7 分析下面程序的输出结果。

```
#include <iostream.h>
void main( )
{
  int a[2][3] = { { 2, 4, 6}, { 8, 10, 12 } };
  int *pa[2];
  pa[0] = a[0];
  pa[1] = a[1];
  for( int i=0; i<2; i++ )
    for(int j=0; j<3; j++, pa[i]++ )
      cout << "a ["<<i<<"]["<< j <<"]="
        << *pa[i]<<endl;
}
```

【解析】当 i=0 时，第二层的 for 循环可简写成：

```
for (j = 0; j< 3; j++, pa[0]++ )
    cout << *pa[0] ;
```

在这个循环中，j 仅起到计数的作用，使 pa[0]仅指向 3 个元素。第一次循环结束后，程序输出为 2，然后 pa[0]++使 pa[0]指向下一个元素，2 的下一个元素是 4，所以输出为 4。然后 pa[0]++又使 pa[0]指向下一个元素，4 后面是 6，所以输出为 6。

当 i=1 时，第二层 for 循环可简写为：

```
for (j = 0; j < 3; j++, pa[1]++ )
    cout << * pa[1];
```

这个循环输出数组 a[1]的三个元素 8，10，12。

程序的运行结果是：

```
a[0][0]=2
a[0][1]=4
a[0][2]=6
a[1][0]=8
a[1][1]=10
a[1][2]=12
```

8 分析使用引用程序的输出。

```
#include <iostream.h>
void addsub(int&, int&);
```