高等教育自学考试计算机及应用专业（独立本科段）自学辅导丛书

面向对象程序设计自学辅导

刘振安　主编

清华大学出版社
http://www.tup.tsinghua.edu.cn

高等教育自学考试计算机及应用专业(独立本科段)自学辅导丛书

面向对象程序设计自学辅导

刘振安　主编

清华大学出版社

(京)新登字 158 号

内 容 简 介

本书依据考试大纲编写，全书共分 12 章。第 1 章至第 9 章的章节结构与考试大纲相对应，每章均先介绍知识点和学习方法，再进行重点与难点分析，最后给出自测练习及其答案。第 10 章是复习应试指南，对全书进行系统复习。为了帮助读者提高应用水平，第 11 章设计了一个成本核算程序以加深对文件操作的理解，为进一步编制实用程序打下基础。第 12 章是模拟试题及参考答案。本书的磁盘中有全部程序的源代码以及另外几套模拟试题及答案。

本书是高等教育自学考试计算机及应用专业(独立本科段)自学辅导丛书之一，旨在帮助考生充分把握考试内容和题型，做好应试准备，取得理想成绩。本书不但可供参加自学考试的考生学习课程使用，也能满足大专院校及社会上 C++语言学习者的需要。

书　名：面向对象程序设计自学辅导
作　者：刘振安　主编
出版者：清华大学出版社(北京清华大学学研大厦，邮编 100084)
http:// www. tup. tsinghua. edu. cn
印刷者：北京市清华园胶印厂
发行者：新华书店总店北京发行所
开　本：787×1092 1/16　印张：15.75　字数：361 千字
版　次：2001 年 8 月第 1 版　2001 年 8 月第 1 次印刷
书　号：ISBN 7-302-04668-9/TP・2772
印　数：00001～10000
定　价：25.00 元(含盘)

前　言

学习C++的关键是面向对象的思维方法，读者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。必须熟悉数据封装的特性与应用，抓住面向对象程序设计的实质。应该深刻理解C++语言的对象和类，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，它必须同时考虑类的数据封装与继承关系。

为了提高应试能力，我们编写了包含各种题型的模拟试题。考生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。只要按照要求吃透并掌握各种题型的分析解题方法，就能具有应变能力，顺利地通过考试。

参加本书编写的有刘振安、孙忱、尹雷、彭程、葛愿、郭小敏、范长杰、章守信、田钢、马杰华、吴倩、颜廷荣、刘胜璞、陈志雄、蒋里、潘剑峰、叶刚和戴笠斐等。本书第1章至第9章的结构均与大纲一致，每章先介绍知识点和学习方法，再进行重点与疑难点分析，最后给出自测练习及其答案；第10章是复习应试指南，对全书进行了系统复习；第11章是综合实例，因为文件操作是应用程序的基础，所以本章使用类的继承和文件设计了一个成本核算演示程序，以加深对文件操作的理解，为进一步编制实用程序打下基础；第12章是模拟试题。本书的程序均在磁盘中，磁盘中还另外提供多套模拟试题及答案。

由于这门课程刚刚开设，所以对社会考生的水平还难以推测。尽管如此，只要考生能够按考试大纲脚踏实地地准备，是不难考出好成绩的。

刘振安

2001年5月8日于合肥

目　录

第1章　面向对象及C++基础知识

本章的主要要求是理解面向对象程序设计的思想及C++语言中的新思想，目的是为学习C++的类打下基础。

1.1　知识点和学习方法

本章具体要求如下：

“识记”层次：(1) 面向对象程序设计思想；(2) C++语言与C语言的关系。

“领会”层次：(1) const修饰符的使用方法和作用；(2) 编译指令的几种形式及其作用；(3) 在程序中应用编译指令。

“简单应用”层次：(1) new的作用；(2) delete的作用。

“综合应用”层次：(1) 程序的基本结构、作用及正确的书写；(2) C++语言的注释、换行、标准输入及输出语句；(3) 头文件及常量的定义；(4) 不使用宏来定义常量的意义；(5) 主函数的基本作用；(6) 函数原型的作用、意义及使用方法；(7) 内联函数的作用及使用方法；(8) 引用的作用及使用方法。

1.2　重点与难点分析

因为本章仅仅是初步建立一些有关C++程序、new和delete、内联函数及引用等的概念，所以重点应放在对新概念的理解和简单应用上，然后在以后的学习中不断加深理解并扩大应用的深度。

【例1.1】　编写一个输入半径，输出其面积和周长的C++示例程序。

```
#include <iostream.h>                    // 系统头文件
const double pi=3.14159
void main( )
{
   double r;
   cout <<"r=";                          // 输出提示信息
   cin >> r;                             // 通过键盘给半径赋值
   double l = 2.0 * pi * r;
   double s = pi * r * r;
   cout <<"\nThe long is: "<< l <<endl;
   cout <<"The area is: "<< s <<endl;
}
```

假设输入 5.6，则运行过程及结果如下：

```
r=5.6
The long is: 35.1858
The area is: 98.5203
```

这个程序演示了系统头文件、自带的输入和输出、新的注释方式和换行符号及类型修饰符 const 的使用方法，应该结合教材熟练地掌握它们的使用方法。

程序中使用了 const 修饰符修饰 pi，编译器将 pi 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 3.14159 取代它。

【例 1.2】 结合下面的程序，简要说明 sub( )函数声明、调用和定义的方法。

```
#include <iostream.h>
void main( )
{
   float a,b;
   float sub(float, float);                    // 用原型声明 sub( ) 函数
   a=12.5;
   b=6.5;
   float c=sub(a,b);                           // 调用 sub 函数
   cout << c << endl;
}

float sub(float x, float y)                    // 定义 sub( )函数
{
   float temp;
   temp = x-y;
   return temp;
}
```

(1) 函数定义和函数原型

函数定义的一般形式如下：

```
类型 函数名(参数表)
{
函数体
}
```

函数 sub()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：

```
float sub (int x, y)          // 错误
```

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。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。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：

```
float sub (int, int) ;
```

(2) 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，表明函数不需要从调用者那里接收数据。虽然参数表可以为空，但函数名后的一对圆括号不能省略。缺省参数就是不要求设定该参数，而由编译器在需要时给该参数赋予预先设定的值。

(3) main 函数和 return 语句

任何一个 C++程序都必须定义一个 main 函数，默认返回类型总是 int 类型。这个函数由操作系统来调用，在 main 函数执行完以后，程序随之终止。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，其他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 1.3】 分析下列程序段中 const 的作用及程序的输出。

```
#include <iostream.h>
void main()
{
  const float num=45.5;
  float & x=num;
  float & y=x;
  ++x;
  cout<<"x="<<x<<",num="<<num<<", y="<<y<<endl;
  ++y;
  ++x;
  cout<<"x="<<x<<",num="<<num<<" ,y="<<y<<endl;
```

```
}
```

程序输出如下：

```
x=46.5,num=45.5,y=46.5
x=48.5,num=45.5,y=48.5
```

为引用提供的初始值，必须是一个也具有初始值的变量。如果是一个常量或是一个使用 const 修饰的变量，则编译器首先建立一个临时变量，然后将该常量的值置入临时变量中，对引用的操作就是对该临时变量的操作。所以，上例中 num 不变，因 y 是对 x 的引用，因而它们同步变化。

应该从现在开始就养成使用 const 和引用的习惯，尤其是定义常数上，更应该马上抛弃使用 define 定义常数的习惯，而改用 const 定义它们。

另外，应养成使用头文件的习惯，并通过编辑上机程序以加强对条件编译指令的理解。

面向对象的程序设计方法和传统的程序设计方法相比，具有抽象、封装、继承和多态性等特征。但要真正认识这些特征的内在含义，则要随着知识的展开而逐步了解。

注意，引用常量问题比较复杂，这里所说的复杂，不是说它难于理解，而是指不同的编译器大相径庭。例如，早先的教科书大都是以 Borland C++为例，Borland C++ 3.1 可以引用常量，而现在流行的 Microsoft Visual C++则不能引用。本例的程序只能在 Borland C++下编译运行，所以在实际使用时，一定要注意使用的具体环境。

1.3 自测练习

1. 分析下面程序的输出结果。

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

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

2. 一个标识符的作用域指的是什么？作用域分为哪 5 种？试举例说明文件作用域。

3. 改正程序的错误，使它的输出为

```
The values are 8,15
The values are 8,815
The values are 8,815
```

需要改错的程序为：

```
#include <iostream.h>
void comp(const int&, int&);
int main()
{
   int count = 8, index = 15;
   cout << "The values are ";
   cout<<count<<","<<index<<endl;
   comp(count, index);
   cout << "The values are ";
   cout<<count<<","<<index<<endl;
   return 0;
}

void comp(const int &in1, int &in2)
{
   in1=in1 * 100;
   in2 = in2 + in1;
   cout << "The values are ";
   cout<<in1<<","<<in2<<endl;
}
```

1.4 自测练习答案

1. 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，n 的初始化是在程序运行中第一次经过它时进行的，即 n=25，所以程序的输出是：

```
n = 24
n = 23
```

2. 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他作用域。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;
      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，在文件作用域中说明的变量称为全局变量，在块作用域中说明的变量称为局部变量。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库块作用域内进行函数说明。

3. 程序错在对常量进行操作。将 comp 函数里的两条语句用如下语句代替即可：

```
in2=in2+in1 * 100;
```

第2章　类和对象

如何正确建立类的概念，是学好面向对象程序设计的关键。对熟悉C程序过程设计方法的读者而言，要特别注意体会类的特点及其真正含义，做好转变思想的工作。本章是全书的切入点，概念多，容易与C程序混淆的地方也不少，需要建立的新概念也具有一定的抽象性，所以要注意通过习题进一步加深理解，以便为后续的学习打下良好的基础。另外，希望能初步理解使用对象成员建立新类的思想。

2.1　知识点和学习方法

对本章的内容要求较高，除了结构和联合的内容只要求达到“识记”层次之外，其他的均要求达到应用层次。要求达到“简单应用”层次的有成员函数重载的作用和重载方法，空类、嵌套类和this指针。而要求达到“综合应用”层次的有：(1) 类的作用域、对象的存取及类的实例化；(2) 说明类的方法、类的标识符和类体；(3) 作用域运算符及成员函数的实现；(4) 对象说明的方法；(5) 访问类对象成员的方法；(6) 封装的实现方法和类的存取控制方法；(7)内联成员函数的作用及其使用方法。

本章的主要要求是：理解并掌握C++语言中类的基础知识及其使用，能够熟练利用类的概念进行编程。学习本章的最好方法是上机编程，通过不同语句理解类成员的存取权限及实现封装性的方法。尤其是应通过大量习题加深对基本概念的理解，为以后学习继承及兼容性规则打下坚实基础。

应该初步建立使用对象成员建立新类的思想，以便为后续的学习打下基础。

2.2　重点与难点分析

本章的重点是学会初步建立类并使用类的对象，掌握类的抽象和封装的概念并真正认识私有成员的保护性。难点是理解C++语言中的this指针、成员对象和重载的概念。本节将通过实例进一步说明这些问题。

应先建立使用成员对象作为另一个类的数据成员的概念，然后再把这一概念推广到函数成员。

【例2.1】　找出并改正下面程序中的错误。

```
#include <iostream.h>
class Point
{
   public:
```

```
        int y;
    protected:
        int x;
        int Getx(){return x;}
    public:
        void init(int a,int b){ x=a; y=b;}
        void Gety(){return y;}
        int Setx(int a){x=a;}
        int Sety(int b){y=b;}
      }A, B;

void main()
{
    A.init(25,27);
    B.init(29,31);
    cout<<"A.x="<<A.Getx()<<endl;
    cout<<"B.x="<<B.x<<endl;
    cout<<"A.y="<<A.y<<endl;
    cout<<"A.y="<<A.Gety()<<endl;
    cout<<"B.y="<<B.y<<endl;
    cout<<"B.y="<<B.Gety()<<endl;
}
```

分析：这个程序的错误较多，现分别说明如下：

(1) 不能把类的成员函数 Getx()定义为保护的。因为 protected 的性质与私有的(private)一样，这就使得 main()函数中的对象 A 和 B 不能使用函数 Getx()。此时，Getx()函数变成不可访问的。如果我们把 protected 改为 private，则情况相同。由此可知，一个类必须至少提供一个用 public 定义的函数，才能给外界提供接口。

(2) 成员函数定义错误。Getx()和 Gety()虽然无参数，但均需要返回 int 类型的值，所以均应定义为 int 类型。而 Setx(int)和 Sety(int)函数虽然有参数，但并不返回一个值，因此应定义为 void 类型。

(3) 数据成员 x 是私有成员，在主程序里不能直接使用对象 B(B.x)来操作私有数据。

因为 y 定义为公有的，所以在 main()程序中既可以通过对象的成员函数 Gety()使用它，也可以直接通过对象使用它。

修改后的完整程序如下：

```
#include <iostream.h>
class Point
{
    public:
        int y;
```

```
    protected:
        int x;
    public:
        void init(int a,int b){ x=a; y=b;}
        int Getx(){return x;}
        int Gety(){return y;}
        void Setx(int a){x=a;}
        void Sety(int b){y=b;}
    }A, B;

void main()
{
    A. init(25,27);
    B. init(29,31);
    cout<<"A. x="<<A. Getx()<<endl;
    cout<<"B. x="<<B. Getx()<<endl;
    cout<<"A. y="<<A. y<<endl;
    cout<<"A. y="<<A. Gety()<<endl;
    cout<<"B. y="<<B. y<<endl;
    cout<<"B. y="<<B. Gety()<<endl;
}
```

程序输出如下：

```
A. x=25
B. x=29
A. y=27
A. y=27
B. y=31
B. y=31
```

对象 A 的数据成员和对象 B 的数据成员的取值不同，但都是使用类 Point 定义的数据成员，即重复使用了程序代码，称为代码复用。

由此可见，对公有数据的访问很简单，但它不能对数据提供保护作用。如上所示，可以在主函数里通过 A. y 和 B. y 访问对象的数据成员 y。一般情况下，我们应将数据成员说明为私有的，以便隐藏数据；而将部分成员函数说明为公有的，用于提供外界和这个类的对象相互作用的接口（界面），从而使得其他函数（例如，函数 main()）也可以访问和处理该类的对象。对于那些仅是为支持公有函数的实现而不作为对象界面的成员函数，我们也应将它们说明为私有的。

公有的成员函数是外界所能观察到（访问到）的对象界面，它们所表达的功能构成对象的功能，使同一个对象的功能能够在不同的软件系统中保持不变。这样，当以后数据结构发生变化时，我们只需要修改少量的代码（类的成员函数的实现代码）就可以保证对象

的功能不变。只要对象的功能保持不变，则公有的成员函数所形成的接口就不会发生变化。这样对对象的内部实现所做的修改就不会影响使用该对象的软件系统。这就是面向对象程序设计使用数据封装为程序员的程序开发活动所带来的益处。

数据封装就是使用类将数据和操作这些数据的代码连接在一起。在 C++ 中，封装可以由 struct、union 和 class 等关键字提供。

在这个程序中，将会碰到如下两种使用类中数据成员的情况：

(1) 类的函数成员。例如，void Setx(int a){x=a;}，不管数据成员是私有的、保护的、或是公有的，类的函数成员都可以使用它们。

(2) 类的对象。例如 A 可以使用类的公有数据成员 y，但不能使用类的私有数据成员 x。因为数据成员 y 是公有性质时，所以下式是正确的。

```
A. y=89;
```

当数据成员 y 是私有或保护性质时，它就是不正确的。这时，对象可以通过成员函数对私有数据进行存取。例如，

```
A. Setx(24);
```

以后还会出现在不同类系中访问这三种数据成员的问题。

【例 2.2】 以下面设计的类为例，再设计一个 void print( )函数，用它说明访问对象成员函数的方法。

可以使用指针及引用的方法使用类的成员函数。假设我们定义的类 Point 如下：

```
#include <iostream. h>
class Point
{
  private:
     int x,y;
  public:
     void Setxy(int a, int b){x=a; y=b;}
     void show(){cout<<"x="<<x<<",y="<<y<<endl;}
};
```

如果要通过指针调用成员函数，为它设计的 print()函数如下：

```
void print(Point * p)
{p->show();}
```

这时，可以使用以下三种方法：

(1) 使用对象的地址。因为是使用类 Point 定义的指针作为函数 print()的参数，所以在 main()函数中调用 print()时，可以使用对象的地址。

```
void main()
{
```