C＋＋语言学习与实验指导

秦　军　主编

河 海 大 学 出 版 社

图书在版编目(CIP)数据

C＋＋语言学习与实验指导/秦军主编. —南京:河海大学出版社,2006.1
ISBN 7-5630-2198-1

Ⅰ.C... Ⅱ.秦.... Ⅲ.C语言—程序设计—高等学校—教学参考资料 Ⅳ.TP312

中国版本图书馆CIP数据核字(2006)**第**002187**号**

书　　名/C＋＋语言学习与实验指导
书　　号/ISBN 7-5630-2198-1/TP·103
责任编辑/代江滨
封面设计/杭永鸿
出　　版/河海大学出版社
地　　址/南京市西康路1号(邮编:210098)
电　　话/(025)83737852(总编室)　(025)83722833(发行部)
印　　刷/
开　　本/787毫米×1092毫米　1/16　16.875印张　330千字
版　　次/2006年1月第1版　2006年1月第1次印刷
定　　价/25.00元

编　委　会

前　言

C++程序设计语言具有强大的功能，它是从 C 语言发展演变而来的，是 C 语言的超集，是一种面向对象的程序设计语言。目前它已经在各个领域得到广泛的应用，成为开发大型软件时首选的一种程序设计语言。

在多年的程序设计语言的教学过程中，作者了解学生在学习该课程时的困惑，即上课听得懂，下课看书也能理解，但是自己编程时却无从下手。这里关键是对程序设计的基本方法和基本思想没能很好地掌握。在长期的教学过程中有很多学生向作者提出“如何学好程序设计语言课程”的问题。要想把书本上的知识变为自己所具有的能力，主要是要实践。但对于初学者在上机实践的过程中，由于对 Visual C++环境不熟悉，没有很好地掌握基本思想和基本方法，再加上缺乏调试方法等等，花了四五个小时也调不通一个程序，由此产生畏难情绪，对该课程失去兴趣。作者编写本书的目的就是企图在提高学生编程能力方面助上一臂之力，使初学者能较为轻松地进入 C++的世界，享受成功的乐趣。

本书分为两部分：第一部分为“学习指导”，首先是“知识要点”，强化对各章节的知识点的认识；其次是“例题解析”，通过一些典型的例题的解析，帮助读者掌握 Visual C++程序设计的方法和技巧，最后通过精心组织的精选习题让读者进行练习，以巩固对已学的知识点的掌握；第二部分为“实验指导”，包括 9 个精心设计的实验。每一个实验都包括“实验目的”、“实验实例”、“实验内容”、“实验指导”和“思考题”。此外，书后还有一个附录“常用系统函数”列出了 C++中的常用库函数及其说明，可以作为简明函数手册使用。

作者建议读者在学习 C++时尽量自己完成习题和实验程序，经过独立思考之后再参考本书中的参考答案。只有通过自己的思考和实践，在错误中不断总结经验，才能真正提高编程能力。

本书在编写过程中，编者参阅了许多 C++的参考书和相关资料，在此谨向这些书的作者表示衷心的感谢。

本书中的全部程序都在 Windows 环境下 Visual C++开发环境中测试通过。

本书第一部分中，俞琼编写第 1 章到第 3 章，朱立华编写第 4 章，朱建编写第 5 章到第 7 章。第二部分，何丽萍编写实验 1 到实验 4，秦军编写实验 5，吴伟敏编写实验 6 到实验 9。全书由秦军统稿。

由于编者水平所限和时间仓促，书中难免有错漏和不足之处，恳请广大读者对本书的内容提出批评和修改建议，我们将不胜感谢。作者的电子邮件：qj@njupt.edu.cn。

作　者

2006 年元月

目　　录

第一部分：C++语言学习指导

第 1 章　面向对象程序设计概述…… 1
第 2 章　C++概述 …… 4
第 3 章　类与对象 …… 23
第 4 章　派生类与继承 …… 68
第 5 章　多态性…… 107
第 6 章　模　板…… 138
第 7 章　C++的 I/O 流类库 …… 156

第二部分：C++语言实验指导

实验 1　熟悉 Visual C++6.0 集成开发环境 …… 175
实验 2　C++概述 …… 184
实验 3　类和对象(一) …… 192
实验 4　类和对象(二) …… 197
实验 5　继承与派生 …… 203
实验 6　重载运算符 …… 208
实验 7　多态性 …… 212
实验 8　模板 …… 216
实验 9　C++的 I/O 操作 …… 221

附录一　实验报告格式及内容…… 225
附录二　常用系统函数…… 226
参考文献…… 263

第一部分：C++语言学习指导

第1章　面向对象程序设计概述

1.1　知识要点

1.1.1　什么是面向对象程序设计

面向对象程序设计　是一种通过为数据和代码建立分块的内存区域，以便提供对程序进行模块化的程序设计方法，这种模块可以被用作样板，在需要的时候再建立其副本。

1.1.2　对象与类

对象　是将描述物体属性的数据以及对这些数据所进行的相关操作封装在一起而构成的统一体。是抽象类的具体实现。

对象具有静态和动态的属性，静态属性又称为对象的状态，通常在面向对象程序设计中用数据成员的形式来表现；动态属性又称为对象的操作或功能，通常在面向对象程序设计中用成员函数来表现。

类　是对一组具有共同的属性特征和行为特征的对象的抽象。在面向对象程序设计中，总是先声明类，再由类生成对象。

对象与类的关系就是具体与抽象的关系，就像以前所学习的类型与变量之间的关系一样。在面向对象程序设计中，一个类只在源程序的代码中出现，而不会在一个正在内存运行的程序中出现，即类只在编译时存在，对象才是运行时存在的实体。所以类是一种新的数据类型，由类定义的不同对象在内存中表现为功能上相对独立的内存区域。

1.1.3　数据的抽象与封装

抽象　是通过对数据实例的分析，抽取其共同性质得到的结果。它要求编程者注重于事物的本质特征，而不是具体的细节。

封装　就是将一组数据和与这组数据有关的操作集合组装在一起，形成一个能动的实体。封装的目的在于将对象的使用者与设计者分开。

抽象是在确定类时强调对象的共同点而忽略它们的不同点的结果，封装则是隐藏了抽象的内部实现细节的结果。抽象和封装是互补的。好的抽象有利于封装，而封装的实体则帮助维护抽象的完整性。

1.1.4　继承性

继承　表达的是对象类之间相关的关系，即一个类自动获得来自另一个类的数据和与这组数据相关的操作。目的在于：避免公用代码的重复开发，减少冗余；通过增加一致性来

减少模块间的接口和界面。

1.1.5 多态性

多态性 指不同的对象收到相同的消息时产生多种不同的反应。

C++语言支持两种多态性,即编译时的多态性和运行时的多态性。编译时的多态性是通过重载来实现的;运行时的多态性则是通过虚函数来实现的。

1.1.6 面向对象程序设计的语言

C++是对传统C语言进行面向对象的扩充,是在C语言的基础上增加了对面向对象程序设计的支持。它不是纯的面向对象语言。

1.2 例题解析

例 1.2.1 面向对象程序设计具有_____等特征。

解答:抽象、封装、继承和多态性

例 1.2.2 在面向对象程序设计中,对象具有封装性,对象之间的联系只能通过_____来完成。

解答:消息传递

例 1.2.3 继承和封装具有相似性,就是都提供_____,因而增加了代码的重用性。

解答:共享代码的手段

继承提供的代码共享是静态的,派生类对象在成为活动的实体以后,自动地共享其基类中定义的代码段,从而使基类对象与其派生类对象共享一段代码。

封装提供的代码共享是动态的,当我们在一个类中说明了一段代码,那么属于该类的多个实例(对象)在程序运行时共享在类中说明的这段代码。

例 1.2.4 模拟一图书管理系统,分析如何对图书、读者、管理人员抽象出相应的静态属性。

解答:针对图书、读者、管理人员的共性分析,所涉及的静态属性

图书:图书条码,书名,作者,出版社,出版日期,数量,借出标识

读者:读者条码,姓名,学号,最多借书数,停借标识

管理人员:管理人员ID,姓名,密码

1.3 习题及参考答案

1.3.1 C++语言是在标准_____的基础上,引入"_____"概念而扩充形成的_____语言。

1.3.2 面向对象程序设计具有的显著特点是 ( )

A. 抽象 B. 继承 C. 多态性 D. 函数调用

1.3.3 面向对象程序设计将数据与_____放在一起作为相互依存、不可分割的整体来处理。 ( )

A. 对数据的操作 B. 信息 C. 数据隐藏 D. 数据抽象

1.3.4 在C++中,封装是借助于_____达到的。 ( )

A. 结构 B. 类 C. 数组 D. 函数

1.3.5　面向对象的程序中,对象的特点有　　(　　)

A. **多态性**　　B. **抽象性**　　C. **封装性**　　D. **继承性**

参考答案:

1.3.1 C **语言　面向对象(或类)　混合型面向对象**

1.3.2 A、B、C　**1.3.3** A　**1.3.4** A、B　**1.3.5** A、B、C

第 2 章　C++概述

2.1　知识要点

2.1.1　C++的起源和特点

起源　C++是美国贝尔实验室的 Bjarne Stroustrup 博士在 C 语言的基础上，弥补了 C 语言存在的一些缺陷，增加了面向对象的特征，于 1980 年开发出来的一种过程性与对象性结合的程序设计语言。

特点　与 C 相兼容，从开发时间、费用到形成的软件的可靠性、可重用性、可扩充性和可维护性上，都比 C 语言表现出明显的优越性。

2.1.2　C++源程序的构成

C++几乎保留了 C 的所有的特性，由下面的比较我们来认识 C++程序：

	C	C++
文件的扩展名	.c	.cpp
注释	/* */	//(一行有效)
头文件	<stdio.h>	<iostream.h>
标准输出	printf	cout <<
标准输入	scanf	cin>>

关键字 cin、cout 及运算符“<<”、“>>”在 C 语言中是没有的。它们是 C++提供的新的输入输出方式。其中 cin 称为标准输入流，cout 称为标准输出流，它们都是流对象。

2.1.3　C++在非面向对象方面的一些特性

局部变量说明

C++允许在代码块的任何地方说明局部变量，其变量的作用域范围为从说明点开始到最小分程序(指包含该说明点的复合语句、函数体等)末。

const 修饰符

当 const 与指针一起使用的时候，应注意 const 修饰的是地址还是地址中存放的内容。

(1) 指向常量的指针　此时 const 修饰的是地址中存放的内容，即当有定义 const char * name="chen"时，“chen”作为存放在 name 指针变量所指向的地址中的内容，其组合是固定的，即为常量，而 name 作为指针变量是可以变化的。

(2) 常指针　此时 const 修饰的是地址，即当有定义 char * const name="chen"时，“chen”作为存放在 name 指针变量所指向的地址中的内容，其组合是可以变化的，而 name 作为指针变量是固定的，即 name 的值不可以变化，为常指针。

(3) 指向常量的常指针　此时指针变量的值不能改变，所指向地址中的内容也不能改变，即当有定义 const char * const name="chen"时，“chen”作为存放在 name 指针变量所指向的地址中的内容，其组合是固定的，即为常量，而 name 作为指针变量也是不可以变化的。

内置函数

在进行定义时就给出函数体或用“inline”关键字标识的成员函数，叫做内置函数。其作用是为了提高程序的运行效率，类中简单的成员函数一般使用内置函数的形式。

- 内置函数在被调用前必须进行完整的定义，通常出现在主函数前面
- 内置函数类似于 C 中的宏定义，但消除了宏定义的不安全因素
- 内置函数中一般不能有循环语句和开关语句
- 类说明体内的函数都是内置函数
- 内置函数是一种空间换时间的措施

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。其作用在于让编译程序进行检查以确定调用函数的参数与事先定义的原型是否相符，以及返回值是否与原型相符，以维护程序的正确性。其语法形式为：

返回类型　函数名(参数表)

note：原型说明中没有指出返回类型的，默认返回类型为 int 型。如果函数没有返回值，应将返回类型设为 void 型。

带有默认参数的函数

函数的形参可直接赋值为默认值，作为缺省参数。如：

void f1(float x, int y, char z='B')

调用时，若相应形参 z 指定了实参，则使用实参的值；否则，形参使用缺省值'B'。

note：缺省定义按从右到左顺序进行。编译程序在进行实参与形参的结合时按照从左到右的顺序。

函数重载

一个函数名可以对应多个函数的实现，即多个函数可以拥有同一个名字，C++允许同一函数名定义多次。

重载的实现方法可以是参数类型不同或参数个数不同。

note：(1) 只有返回值不同的两个函数不可以重载；

(2) 采用参数个数不同的方式进行重载时，要注意可能因为参数指定缺省值而引起二义性；

(3) 在调用时，编译器将根据不同的参数类型、参数个数、参数顺序自动选择一个合适的函数。

new 和 delete

new：运行时根据程序要求为变量分配内存空间。

格式：(1) 指针变量 = new 类型 (初始值列表)；//单个存储单元的分配

例：int *p;　int *pi;

p=new int;　　==　　int *p=new int;

pi=new int (249);

(2) 指针变量 = new 类型[整型表达式]；//连续存储空间的分配，实现可调数组

例：int *pq;

pq= new int [10];

delete:释放已分配的内存空间。

格式:(1) delete 指针变量;　　//释放单个存储单元

(2) delete [] 数组名;　　//释放连续存储空间

使用 new 和 delete 来动态操作内存的好处是:

(1) 可以实现可调数组。

(2) 可以灵活使用内存,提高内存的使用率。

note: new 与 delete 的配套使用,可避免不及时释放空间而造成内存空间的浪费。

引用

就是将一个新标识符和一块已经存在的存储区域相关联,它是一个别名,是某一个目标对象的替代名,因此,使用引用时没有分配新的存储区域,对所有加于引用上的操作实际上就是加于引用的目标对象之上。在声明引用时必须进行初始化。

(1) 为什么要用引用?

- 引用可以像地址一样完成对所代表的对象的操作
- 对引用的操作不同于对指针的操作,它可以直接作用于变量,所以更方便,更易于理解

(2) 引用与指针的区别。

- 引用是某一对象的别名,指针是某一变量的地址变量
- 引用在定义时就必须初始化,是一已定义的对象的别名,不可以重新赋值为其他变量。指针可在使用时初始化,且可以重新赋值

(3) 引用的应用。

- 传递参数
- 引用返回值:函数可以返回一个引用,目的是该函数可以用在赋值运算符的左边

void 型指针

表示不确定的类型。即任何类型的指针值都可以赋给 void 类型的指针变量。但已经获得值的 void 类型指针再次进行处理时需进行显式的类型转换。

2.2 例题解析

例 2.2.1　在 C++中,表达式 cout<<endl 还可表示为_____。

解答:cout<<'\n';cout<<'\12',cout<<"\xA"或其他等价形式。

因为在 C 语言中曾讲过用"\"引导转义字符。"\12","\xA"分别为八进制和十六进制。参看 C 语言课本。

例 2.2.2　下面是一个 C 程序,改写它,使它采用 C++风格的 I/O 语句。

```
#include <stdio.h>
main()
{
  int a,b,c;
  int temp;
  printf("Enter a,b,c:");
```

```
    scanf("%d%d%d",&a,&b,&c);
    temp=a;
    a=b;
    b=c;
    c=temp;
    printf("a=%d b=%d c=%d\n",a,b,c);
}
```

解：

```
#include <iostream.h>          //C++风格的I/O语句是以流对象为特征的,所以
                               //在选择头文件时应包含"iostream.h"。
void main()                    //C++的主函数一般要求在前面写上返回类型,若函数
                               //没有指出返回类型,C++默认该函数的返回类型
                               //为int。而本题中无返回值,故应为 void 型。
{ int a,b,c;
  int temp;
  cout<<"Enter a,b,c:";        //cin,cout 及运算符<<,>>在 C 语言中是没有的,
                               //它们正是 C++提供的新的输入输出方式。其中
                               //cin 称为标准输入流,cout 称为标准输出流,它们都
                               //是流对象;<<是输出运算符,>>是输入运算符。
  cin>>a>>b>>c;
  temp=a;
  a=b;
  b=c;
  c=temp;
  cout<<"a="<<a<<" b="<<b<<" c= "<<c<<"\n";
                               //当需要连续输出不同变量值时,注意使用进行分
                               //割,避免出现 cout<<a, b, c;这样的错误。
}
```

例 2.2.3　下面是一个 C++程序,不能运行,试指出其错误并改正。

```
main()
{
int a,b;
cin>>a>>b>>endl;
b+=a;
cout<<b<<endl;
}
```

解：错误原因：在没有包含头文件"iostream.h"的情况下,如系统没有事先约定,cin、cout 将不被认为标准输入、输出流,却会被看做普通的变量名因无定义而不被识别。

endl 作为行结束符,在输出时使用,在输入时无法进行流提取操作,被看做普通变量名。

main()函数因没有返回值类型说明而出现编译警告。

应作如下改动：

```
#include "iostream.h"
void main()
{
int a,b;
cin>>a>>b;
cout<<endl;                    //想使得输入数据与输出数据分行显示,应单独输出 endl。
b+=a;
cout<<b<<endl;
}
```

例 2.2.4　下面是一个 C++程序，不能运行，试指出其错误并改正。

```
#include "iostream.h"
void main()
{
  for(int k=0;k<=5;k++)
   {
       int a,b;
       cin>>a>>b;
       if (a>k || b<k) break;
   }
    cout<<a<<b<<k<<endl;
}
```

解：错误原因是 C++中允许在代码块的任何地方说明局部变量，使用时应注意其作用域范围，a，b 两个局部变量是在 for 循环结构中说明的，其作用域范围仅限制在包含它的复合语句中，所以在 cout<<a<<b<<k<<endl；语句中 a，b 变量已经不存在，无法使用。

为了能使得输出正常进行，应将 a，b 两变量的说明语句放置在复合语句外。即：

```
#include "iostream.h"
void main()
{ int a,b;
  for(int k=0;k<=5;k++)
  {
       cin>>a>>b;
       if (a>k || b<k) break;
  }
  cout<<a<<b<<k<<endl;
}
```

例 2.2.5　下列定义中：

ptr 所指向的地址可更改，但 * ptr 不可更改的是_____。

ptr 所指向的地址不可更改，但 * ptr 可更改的是_____。

ptr 所指向的地址可更改， * ptr 也可更改的是_____。

ptr 所指向的地址不可更改， * ptr 也不可更改的是_____。

A. const int * ptr;　　B. int * const ptr;

C. const int * const ptr ;　　D. int * ptr;

解：A,B,D,C

在辨别 const 与指针一起使用时，简单的方法是从右往左看，例如 const int * ptr;**首先** ptr **是指针(*)，指向整型(int)的常量(const)，得出** ptr **所指向的地址可更改，但** * ptr **是不可更改的；再如** int * const ptr;**首先** ptr **是常(const)指针(*)，指向整型(int)变量，得出** ptr **所指向的地址不可更改，但** * ptr **是可更改的。**

例 2.2.6　C++中为什么提倡用 const 定义一个常量，这样做与用 #define 来定义有什么不同？

答：这两种写法的差别在于使用宏 #define 定义的常量没有数据类型，而使用 const 来定义的常量则有数据类型，当我们将这些常量当做函数的参数来传递时，可以帮助编译程序来检查参数间的数据类型是否正确。

例 2.2.7　下面是使用带参宏定义完成求平方的功能。其程序为：

```
#include "iostream.h"
#define sq(x) x*x
int main()
{ for(int k=2;k<=3;k++)
  cout<<k<<"  square is "<<sq(k)<<endl;
  cout<<"1+3 square is"<<sq(1+3)<<endl;
  return 0;
}
```

程序输出结果为

```
2 square is 4
3 square is 9
1+3 square is 7
```

请使用内置函数替代宏定义进行正确的运算。

解：

```
#include "iostream.h"
inline int   sq(int x)          //内置函数消除了#define的不安全因素
{return x*x;}
int main()
    {
        for(int k=2;k<=3;k++)
        cout<<k<<"  square is"<<sq(k)<<endl;
        cout<<"1+3 square is"<<sq(1+3)<<endl;
```

```
    return 0;
}
```

例 2.2.8　编写一个输入半径，输出其面积和周长的 C++示例程序。

解：

```
#include "iostream.h"                          //系统头文件
const double pi=3.14159;                       //定义双精度常量 pi
void main()                                    //main 函数无返回值
{ double r;                                    //定义局部变量 r
  cout<<"r=";                                  //输入界面提示信息
  cin>>r;                                      //通过键盘给半径赋值
  double l=2.0*pi*r;                           //定义局部变量 l
  double s=pi*r*r;                             //定义局部变量 s
  cout<<"\nThe long is:"<<l<<endl;             //\n 换行使得输入与输出分行显示
  cout<<"The area is:"<<s<<endl;
}
```

本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，应该结合教材熟练掌握它们的使用方法。

例 2.2.9　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;
}
```

解：程序的错误在于对常量进行赋值操作。函数 void comp(const int &in1,int &in2) **中的形参** in1 **由修饰符** const **定义为常量，则在函数体中，不可以出现** in1=in1*100；**这样的语句，应将** in1=in1*100；in2=in2+in1；**这两句改为** in2=in2+in1*100。

例 2.2.10　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。

答：一个标识符的作用域是程序中的一段区域，用于确定该标识符是否可用（可见性）。作用域分为：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（全局作用域）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到文件结束处结束。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包含了该头文件的程序文件中，直到该程序文件结束。例如：

```
(1) #include "iostream.h"
(2) int i;                                    //全局作用域
(3) void main()
(4) {
(5)     i=5;
(6)    {
(7)         int i;                            //局部作用域
(8)         i=7;
(9)         cout<<"i="<<i<<endl;              //输出"7"
(10)    }
(11)    cout<<"i="<<i<<endl;                  //输出"5"
(12) }
```

第 2 行中的 i（**下面称为** i2）**为文件作用域（全局作用域），第 7 行中的** i（**下面称为** i7）**为块作用域（局部作用域），**i7 **屏蔽外面的** i2，**这时在包含** i7 **的复合语句（即** 7～10 **行）中无法存取** i2。

通过使用作用域运算符::**可以在块作用域中存取被屏蔽的全局作用域中的** i2。**例如：**

```
(1) #include "iostream.h"
(2) int i;                                    //全局作用域
(3) void main()
(4) {
(5)     i=5;
(6)    {
(7)         int i;                            //局部作用域
(8)         i=7;
(9)         ::i=1;
(10)         cout<<"i="<<i<<endl;             //输出"7"
```

```
(11)    }
(12)    cout<<"i="<<i<<endl;              //输出"1"
(13) }
```

变量可以在文件作用域中说明为全局变量，在块作用域中说明为局部变量。

例 2.2.11 如果在程序中，我们只用 new 进行了动态内存分配，而忘记了用 delete 来释放，会产生什么样的后果？使用 new 和 delete 来动态操作内存，有何好处？

答：若不及时用 delete 来释放用过了的内存，会产生大量的内存垃圾，随着垃圾的增多，计算机的速度会越来越慢，最后造成死机。

使用 new 和 delete 来动态操作内存的好处是：

① 可以实现可调数组。

② 可以灵活使用内存，提高内存的使用率。

例 2.2.12 在下面的语句中错误的是_____。

A. int n=5;　　int y[n];

B. const int n=5;　　int y[n];

C. in n=5;　　int *py=new int[n];

D. const int n=5;　　int *py=new int[n];

答：A，选项中 n 是变量，int y[n]企图定义一动态数组是不可行的。B 选项中 n 为常量，C、D 选项均为用 new 动态操作内存从而可正确定义可调数组。

例 2.2.13 在下列函数原型中错误的是_____。

A. int add(int X=3,int y=4,int Z=5)

B. int add(int x,int y=4,int z)

C. int add(int x,int y=4,int z=5)

D. int add(int x,int y,int z=5)

答：B，带默认参数的缺省定义按从右到左顺序进行，在出现缺省定义后不可以再出现非缺省参数。

例 2.2.14 在下列重载函数调用时可能发生错误的是_____。

A. int print(int X);
　　void print(float X);

B. int disp(Myclass A);
　　char *disp(Myclass A);

C. int show(int X);
　　int show (char *s);

D. int view(int X,int y);
　　int view(int X);

答：B

例 2.2.15 函数被说明为内置函数后，对哪些开销有影响？

答：内置函数的引入较好地解决了时间开销的问题。这是因为在程序编译时，编译器将程序中调用内置函数的表达式用内置函数的函数体来代替，类似于在 C 语言中讲过的带参宏定义。显然，这种做法没有函数调用时存在的保护现场和恢复现场的问题，但是，由于编译时用函数体替代了调用表达式，增加了目标程序的代码量，带来了空间开销的增加。由于计算机硬件的迅速发展，内存容量不断增大，空间开销的问题不再成为主要问题。

例 2.2.16 用 new 操作为一个包含 20 个整数的数组分配内存，输入若干个值到数组中，分别统计其中正数和负数的个数，再用 delete 操作释放内存。