C++语言培训教材

（下册）应用篇

● 刘峻桐　刘　豫　著

URL:http://www.phei.co.cn

PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
電子工業出版社

C++语言培训教材（下册）应用篇

C++语言培训教材

（下册） 应用篇

刘峻桐 刘 淳 编著
高永泉 贾宝金 审校

電子工業出版社·

内容简介

全书分上、中、下三册。上册为基础篇，讲述了C++语言的特点及优越性和C++程序设计的风格，C++的操作符、语句、函数、数组、变量作用域、模块化程序设计和指针等。读者通过上册的学习，基本上可以掌握C++语言的基础。

中册为提高篇。让一步对指针和数组进行深入的研究，进而提出类、对象以及继承性，并围绕类和对象进一步阐述了C++语言面向对象的特点。随后又对构造函数、析构函数、虚函数以及函数重载等作了详细的讲述，培养读者用类机制解决实际问题的能力，掌握面向对象的程序设计方法。

下篇为应用篇。重点讲述C++语言的输入输出(I/O)及介绍4个应用例程，通过解剖分析这4个C++实用程序，培养读者运用所学的C++语言编写程序时解决实际问题的能力。

本书适用于初中级教材，也可作为有关的训练和自学教材。

JS350/16

C++语言培训教材
（下册） 应用篇
刘峰桐 刘 俊 编著
高永泉 周宪余 审校
责任编辑 毛兆余
特约编辑 张成全
×
电子工业出版社出版
北京市海淀区万寿路173号信箱(100036)
电子工业出版社发行 各地新华书店经销
北京科技大学印刷厂印刷
×
开本 787×1092毫米 1/16 印张 9.5 字数 228千字
1996年9月第一版 1996年9月第一次印刷
印数 5000册 定价 12.00元
ISBN 7-5053-3822-2/TP·1698

前　言

C++代表着程序设计语言领域的一个重要进步。目前，学习C++程序设计语言的人越来越多，许多以前用C语言编程的人员现在开始转向用C++，C++代表了C语言的发展趋势，C++采用了许多重要技术，这使它替代C语言只是一个时间问题了。

C++是C语言的增强型版本。在C语言的基础上，C++做了功能扩充以支持面向对象的程序设计，这些扩充极大地增强了C语言的能力，C++语言的强大功能及其通用性，决定了它必将成为继C之后的主要程序设计语言之一。

事实上，C语言在应用上所形成的广泛基础已经决定了C++语言在未来程序设计语言中的主导地位。从本质上讲，C++与C有着不同的编程风格。学习C++语言，不仅要学习语言要素本身，而更重要的是学会如何用C++的思维方式进行程序设计。为了使读者能够尽早地养成用C++思维方式来考虑问题的习惯，本书力求一开始即给读者展现以C++的编程风格，而不是传统的C语言风格。

在语言学习中，单调的语法讲解不仅无助于有经验的程序设计人员提高编程技巧和能力，而且往往会使初学者感到迷惑。因此，本书在讲解C++语言要素的同时，结合大量精短的程序，以帮助读者理解和掌握C++语言，在阅读本书时读者将会发现，用这种方法讲解语言，不仅可以使你非常容易地掌握C++语言要素，与此同时还能学到许多编程技巧。

本书分上、中、下三册。上册为基础篇，基础篇共十七章，首先讲述了由C语言演变为C++语言的发展过程，C++语言的特点以及与C语言的主要区别，阐述了C++语言的优越性，进而讲述了C++程序设计的风格及C++的操作符、语句、函数、数组、变量作用域、模块化程序设计及指针等。因此可以说上册是学习C++语言的基础，读者通过上册的学习，基本上可以掌握C++语言的基础。

中册为提高篇，共九章（第十八章至第二十六章），是在上册的基础上进一步对指针和数组进行了深入的研究讨论，进而提出类、对象以及其继承性，并围绕类和对象进一步阐述了C++语言面向对象的特点，随后又对构造函数、析构函数、虚函数以及函数重载等作了详细的讲述，以培养读者用类模拟解决实际问题的能力，进而掌握面向对象的程序设计方法。

下册为应用篇，由本书的和第二十七章至第三十一章以及两个附录组成，重点叙述C++语言的输入输出（I/O）及介绍4个应用例程，并通过举例解剖分析4个C++实用程序，来培养读者运用所学的C++语言解决实际编程问题的能力。

本书在编写过程中，高永泉、贾宝金、王程兰、李桂萍等给予了大力支持，并做了大量工作，在此表示衷心的感谢。但因作者水平有限，虽然从事编程多年，亦恐书中仍有错误不妥之处，欢迎读者批评指正。

目　录

第二十七章　面向对象的I/O系统基础

C++除了完全支持ANSI C标准I/O系统外，还定义了一套面向对象的I/O系统。我们知道，ANSI C标准I/O系统是一种内容非常丰富，灵活，功能非常强大的I/O系统。那么，既然如此，为什么C++又要定义另外一套I/O系统呢？

其根本原因是ANSI C标准I/O系统一点也不适用于对象。因此，C++为了能够对用户创建的对象进行I/O操作，定义了一套面向对象的I/O系统。

C++面向对象的I/O系统不仅广泛地用于对象操作，而且在用于其它类型变量的操作时也有一些间接的好处。尽管C++完全支持ANSI C标准I/O系统，然而，C++面向对象的I/O系统所具有的良好灵活性，已使ANSI C标准I/O系统在C++中的使用具有很小的吸引力。

本章将介绍C++的基本流类，怎样格式化数据，怎样重载>>和<<运算符，使其用于类操作，如何创造特殊的操作符I/O函数。

主要介绍的内容有：

- 关于流
- 基本的流类库
- C++的预定义流
- 流I/O操作符<<和>>
- 使用ios成员格式化I/O
- 用操纵符格式化I/O
- 重载<<和>>
- 创建自己的操纵符

27.1　关于流

C++的I/O系统也是通过流操作的。前面我们已经介绍过，流实际上是一种既产生信息同时又消费信息的抽象设备。它通过C++I/O系统和物理设备相关联。因此，无论实际的物理设备有多大的差别，流都以相同的方式起作用。C++与ANSI C标准I/O一样，其I/O系统流不受硬件的限制，可以适用于各种不同型号的机器。也就是说，相同的C++I/O函数实际上可以操作任何类型的物理设备。

27.2　基本的流类库

C++在头文件iostream.h中定义了用以支持I/O的流类库。在该类库的类层次中，最低层为两个平行的类streambuf和ios。

类 streambuf 中提供了基本的输入和输出操作，主要用作其它类的基类，提供对缓冲区的低级操作，如设置缓冲区，对缓冲区指针操作，从缓冲区取字符，向缓冲区存贮字符等。如果用户不派生自己的 I/O 类，那么就不能直接使用 streambuf。下图表明了类 streambuf 和它的两个派生类 filebuf 和 strstreambuf：

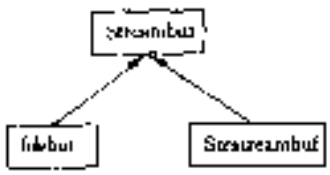

图 27.1

ios 类是类库中另一个类体系中最低层的类。类 ios 以及所有从 ios 派生来的类中都有一个指针指向类 streambuf。类 ios 主要用于提供输入输出中的格式、错误检测和状态信息。下图表明了类 ios 以及其派生类：

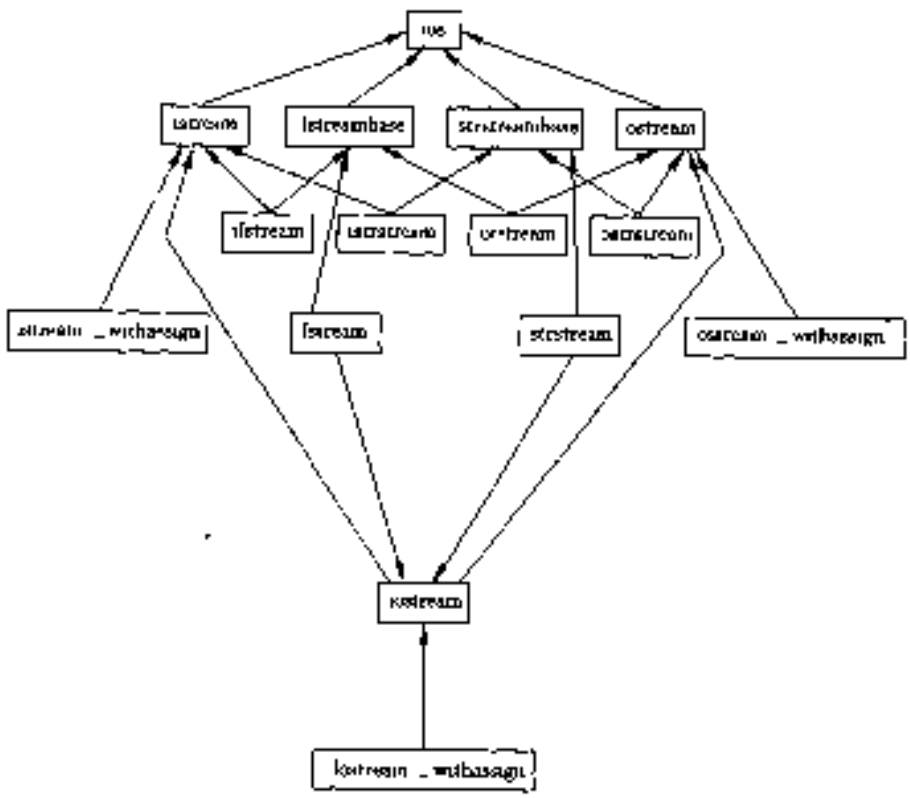

图 27.2

图中箭头标明了继承关系，例如，类 ifstream 是从类 istream 和 fstream 派生而来的。从图中还可以看出，其中有些类的派生不是简单的派生，而是经过多个基类、多层次

派生而得到的。例如，派生类 fstream，它直接或间接地继承了类 iostream、istream、ostream、fstreambase 和 ios。因此，这些类的成员函数和成员变量，都包含在派生类 fstream 中。

从类 ios 派生出的类 istream、ostream 和 iostream 分别用于创建输入流、输出流和输入输出流。

类 ios 及其所有派生类都使用一个 streambuf(或 filebuf 和 strstreambuf)作为其所创流的源和目的，或者同时为源和目的。

类 ios 中包含的许多成员函数和变量，用于控制或监视流的基本操作。本章和下章所介绍的内容中将会用到许多类 ios 的成员。这里首先提醒一点，如果以通常方式使用 C++的 I/O 系统，任何流都可以使用 ios 的成员，因为它处在原始基类的地位。

27.3 C++的预定义流

C++程序开始运行时，自动打开四个内部流。其定义(声明)如下：

```
extern istream cin;
extern ostream cout;
extern ostream cerr;
extern ostream clog;
```

由此可见，所谓的内部标准流，也就是用前面介绍的流类声明了四个流类类型的对象 cin、cout、cerr 和 clog。它们的含义如下：

流	含义	缺省设备
cin	标准输入	键盘
cout	标准输出	屏幕
cerr	标准错误输出	屏幕
clog	cerr 的缓冲形式	屏幕

由此我们可以看出，流 cin、cout 和 cerr 对应于 ANSI C 中的 stdin、stdout 和 stderr。

这些流称为标准流，缺省时它们用于控制台输入输出。我们在前面已经介绍和使用过它们。在支持 I/O 重定向的环境中(如 DOS、UNIX 等)，还可以把标准流重定向到其它设备或文件。

27.4 流输入输出操作符

流的输出用操作符<<实现。在 C++中，标准左移位操作符被重载，用于流输出操作。

流输出操作符<<的左操作数是流类 ostream 类型的对象，右操作数为流输出已定义的任何类型。也就是说，在默认情况下，流输出操作符<<可用以输出任何内部类型，也可以由用户对其重载(后面将介绍重载<<和>>)，以把它用于由用户自己定义的类型的对

象的输出。

可直接用流输出操作符<<输出内容的类型有：

char (signed 和 unsigned)、short (signed 和 unsigned)、int (signed 和 unsigned)、long (signed 和 unsigned)、float、char * (当作串处理)、double、long double 和 void *。除非设置各种 ios 标志改变规则外，整类型根据 printf 的默认规则转换。例如：

```
int x;
int y;
```

则下面两个语句的输出结果相同：

```
cout<<x<<","<<y;
printf("%d,%d", x,y);
```

同样，浮点类型是根据 printf 默认的规则转换的。例如，假如说明一个浮点型变量 double d，则下述两个语句结果相同：

```
cout<<d;
printf("%g", d);
```

指针(void *)插入符也是预定义的，例如：

```
int i=10;
cout<<&i;          //用十六进制显示地址
```

流输入与流输出相同，在C++中，标准右移位操作符被重载，用于流输入操作。

流输入操作符>>的左操作数为流类 istream 类型的对象，同样，其右操作数可以是输入流已定义的任何类型。

流输入操作符在默认时跳过空白，然后读入对应于输入对象类型的字符。对于类型 char *，则被当作串看待，操作符>>从一个非空白字符开始读取直到遇到一个空白字符为止，然后在后面追加一个'\0'。在用流输入操作符>>为串赋值时，应注意不要溢出。下面的语句可有效地避免溢出：

```
char array[SIZE];
.........
//初始化 array
.. .... .
cin.width(sizeof(array));          //width()将在后面讲到
cin>>array;
```

对于所有内部类型的输入，如果在遇到非空白字符之前结束，则 istream 状态置为失败，这时如果右边的操作数是个未初始化的操作数，它们保持未初始化。

27.5 使用 ios 成员格式化 I/O

类 ios 包含着许多控制或监视流的基本操作的成员函数和变量。使用它们，用户可以设置域宽，指定数字基数或确定显示到十进制小数点后多少位等。

前面我们学过，用函数 printf()和 scanf()可以完成各种格式化 I/O。从根本上讲，用面向对象的 I/O 系统所提供的操作符>>和<<可以完成 printf()和 scanf()所能完成的所有 I/O 操作。

面向对象的 I/O 系统中，有两种相关的但概念不同的格式化数据的方法。第一种是我们这一节将介绍的，通过设置 ios 类中定义的各种格式状态位或调用各种 ios 的成员函数格式化数据。第二种方法我们将在下一节中介绍的通过作为表达式一部分的操纵符函数格式化数据。

27.3.1 ios 格式化标志集合

ios 格式化标志集合对应于每一个流，对它们进行操作，可以通过流格式化信息。

在 ios 中定义了下面所列出的枚举，枚举中定义了各个元素(标志)及其值，用来设置或清除格式化标志。

```
public:
enum{
    skipws          =0x0001,
    left            =0x0002,
    right           =0x0001,
    internal        =0x0008,
    dec             =[illegible],
    oct             =0x0020
    hex             =0x0040,
    showbase        =0x0080,
    showpoint       =0x0100,
    uppercase       =0x0200,
    showpos         =0x0400,
    scientific      =0x0800,
    fixed           =0x1000,
    unitbuf         =0x2000,
    stdio           =0x4000,
};
```

其中各标志的含意如下：

(1)设置 skipws 标志时，在流上进行输入操作时丢弃所有空白字符(空格、制表符和换行符)；清除 skipws 标志时，则保留空白字符。

(2)设置 left 标志时，输出左对齐。设置 right 标志时，输出右对齐。

如果上述两个标志都未设置，则按缺省时的右对齐输出。如果设置 internal 标志时，对于数值则通过在符号或基字符间插入空格来填充域。

(3)缺省时数值以十进制形式输出，设置 oct 标志使输出以八进制显示，设置 hex 标志使输出以十六进制显示。要使输出转换回十进制，则需设置 dec 标志。

(4)设置 showbase 标志时，显示数值的基。例如，如果以十六进制显示数值，则值 11 就显示为 0x11。如果这时设置标志 uppercase 时，则显示科学记数时"E"为大写。同样，显

示十六进制时"X"也为大写，缺省时，这些字符都为小写。

(5)设置 showpos 标志时，则在正数前显示加号。

(6)设置 showpoint 时，所有浮点输出，都显示十进制的小数点和尾 0。

(7)设置 scientific 标志时，浮点数字值将以科学记数法显示。当设置 fixed 时，浮点值以常规记数法显示。缺省时，显示六位十进制位。这些标志都没有设置时，编译程序选择一种适当的方式。

(8)当设置了 unitbuf 时，输出部分地用缓冲实现，这样就提高了 C++ 的 I/O 性能。缓冲区在每个插入操作之后被清空。

(9)当设置 stdio 时，每次输出之后清空每个流。清空一个流使输出流实际上写入与流相联系的物理设备，

格式标志以一个长整型数存储。

27.5.2 设置格式标志

ios 的成员函数 setf()用来设置格式标志，其一般格式为：

```
long setf (long flags);
```

函数 setf()返回前一次设置的格式标志的值，并打开由 flag 指定的那些设置。对于未指定的其它标志则不产生影响，例如，下述语句打开 skipws 标志：

```
stream.setf (ios ::skipws);
```

其中，stream 是要影响的流。

下面的程序以八进制和十六进制分别显示 100 并指出其基。

```
# include "iostream.h"
void main()
{
    cout setf (ios::oct);
    cout setf (ios::showbase);
    cout<<100;
    cout.setf (ios::hex);
    cout.setf (ios::showbase);
    cout<<100;
}
```

可以用或(OR)把所希望设置的标志联在一起，从而使得只调用一次 setf()函数一次设置几个标志。例如，上述程序可以改写如下：

```
void main()
{
    cout.setf (ios::oct | ios::showbase);
    cout<<100;
    cout.setf (ios::hex , ios::showbase);
    cout<<100;
```

```
}
```

记住 setf()是类 ios 的成员函数这一点是非常重要的，它被类 ios 的所有派生类继承。因此，会体对 setf()函数的调用都关系到一个特定的流。

setf()函数被它自己调用是没有意义的。也就是说，C++没有全局格式状态的概念，每一个流都维持它自己的格式状态。

最后还必须注意一点，由于各标志都是在类 ios 中定义的，因此在访问这些标志时必须用 ios 和作用域分辨操作符，尽管这里函数 setf()是类 ios 的成员函数。(请读者考虑为什么这样，以及函数 setf()怎样才能改变标志)。

27.5.3 清除格式标志

与成员函数 setf()对应，ios 的另一个成员函数 unsetf()用于清除一个或多个格式标志，其一般格式为：

```
long unsetf (long flags);
```

其中，flags 是要清除的标志。该函数只清除 flage 所指出的标志，除此之外，对其它标志不产生影响，函数返回上一次设置的标志。

下面程序用于说明成员函数 unsetf()的使用：

```
#include "iostream.h"
void main()
{
    cout.setf (ios::hex | ios::showbase);
    cout<<100;              //display 0x64
    cout.unsetf (ios::hex | ios::showbase);
    cout.setf (ios::dec);
    cout<<100;              //display 100
}
```

上述短程序中首先设置标志用十六进制和基格式输出 100，然后清除十六进制标志，设置回十进制标志，并输出 100。

27.5.4 函数 setf()的重载形式

函数 setf()有重载形式，其一般格式为：

```
long setf (long flags1, long flag2);
```

在函数 setf()的重载形式中规定，只有 flag2 指定的标志起作用，函数使标志 flag2 复位，然后使 flag1 所指示的标志置位。

由于函数中只有 flag2 指定的标志起作用，因此，flag1 通常需包含在 flag2 中。注意，尽管 flag1 可能包含 flag2 中没有指定的其它标志，但是只有那些在 flag2 中指定的标志才起作用，函数返回上一次设置的标志。

下面的程序在输出流 cout 中设置 showpos 和 showpoint，然后复位这两个标志并设置 showpoint 标志。

```
# include "iostream.h"
void main()
{
    cout.setf(ios::showbase  ios::hex);
    cout<<100<<"\n"       //display 0x64
    cout.setf (ios::oct, ios::hex | ios::oct);
    cout<<100;            //display 0144
}
```

注意：flag1 中只有那些在 flag2 中指明的标志才起作用。

如果把上述程序改写成如下形式，将会出现错误：

```
# include "iostream.h"
void main()
{
    cout.setf(ios::showbase | ios::hex);
    cout<<100<<"\n";  //display 0x64
    cout.setf(ios::oct, ios::hex); //error: oct not set
    cout<<100;            //display 100
}
```

在这个程序中，函数 setf()中 flag2 中指出只有 hex，因此只有 hex 标志才是有效的。flag1 的值 oct 没有在 flag2 中指定，因而无效。因此程序中 setf()复位 hex 标志，但不能置位 oct 标志。

在 flag2 中引用 oct、dec 和 hex 时，可以用 ios::basefield 替代，

同样，引用 left、right 和 internal 可以用引用 ios::adjustfield 实现。scientific 和 fix 可以用 ios::floatfield 替代。

下面我们用这种替代把上述程序改写如下：

```
# include "iostream.h"
void main()
{
    cout.setf(ios::showbase | ios::hex);
    cout<<100<<"\n";
    cout.setf(ios::oct, ios::basefield);
    cout <<100;
}
```

由此可以看到，setf()的重载形式同时起到了 unsetf()和 setf()的作用。但应注意，大多数情况下使用 unsetf()清除标志，用带有一个参数的 setf()设置标志，setf()函数的重载形式只用在一些特定的场合。

27.5.3 检查格式标志

成员函数 setf()和 unsetf()用于设置或清除标志，而有时用户需要知道现行标志的设

置情况，而不希改变现行设置情况，ios 的另一成员函数 flags()提供了这一功能。其一般格式如下：

```
long flags();
```

函数 flags()返回一个长整数，该长整数既反应了当前设置的每个标志的情况。例如，下面的程序检验函数 flags()的返回值，查看 skipws 标志是否置位，如果置位则显示 on，否则显示 off。

```
# include "iostream.h"
void main()
{
    long f;
      f=cout.flags();
    if (f&ios::skipws) cout<<"on \n";
    cout<<"off\n";
}
```

下面程序中编写了一个函数 displayflags()它将显示出所有标志的设置情况，请注意函数 displayflags()的定义。

```
# include "iostream.h"
void displayflags();
void main()
{
  displayflags();          //call displayflags show default flags
}
void displayflags()
{
    long f;
    int i;
    char * name [15][12]={
                          "skipws",
                          "left",
                          "right",
                          "internal",
                          "dec",
                          "oct",
                          "hex",
                          "showbase",
                          "showpoint",
                          "uppercase",
                          "showpos",
                          "scientific",
                          "fixed",
```

```
                                "unitbuf",
                                "stdio"
    };
    f=cout.flags();
    for (j=1, j=0; j<=0x4000; j<<=1, j++)
    if (f&j) cout <<f_name[j] <<"is on\n";
    else cout<<f_name[j] <<"is off\n";
    cout <<"\n";
}
```

程序将输出如下结果：

```
skipws is on
left is off
right is off
internal is off
dec is off
oct is off
hex is off
showbase is off
showpoint is off
uppercase is off
showpos is off
scientific is off
fixed is off
unitbuf is off
stdio is off
```

上述显示结果，正是缺省设置时各标志的设置情况。

27.5.6 用 flags()设置所有标志

成员函数flags()还有一种形式可以用来设置与流相关的、在 flags()的参数中指定的那些格式。其一般格式为：

```
long flags (long f);
```

当用函数 flags()的上述格式设置标志时，函数 flags()把参数 f 以位方式拷贝到流的格式标志变量中去。该函数返回上一次的设置。

由于 flags(long f)以位的方式把参数 f 的值拷贝到容纳相关流的格式标志中，因此，当需要置位某些标志时，可以把第一节中所列出的这些标志的值加在一起，作为参数 f 的值传递给函数 flags()，那么这些标志(值被加到一起传递给函数 flags()的标志)在对应的流中便被置位，其它标志将被复位。

例如，标志 showbase、showpoint、hex 和 left 的值分别为 0x0080、0x0100、0x0040 和 0x0002，把这些值加起来其和为 0x1c2。在下面的程序中我们把 0x1c2 传递给函数 flags()，