21世纪高等院校计算机教材

C++
程序设计教程
（第三版）

Leen Ammeraal 著

刘瑞挺 等译

中国铁道出版社
CHINA RAILWAY PUBLISHING HOUSE

C++

程序设计教程

（第三版）

Leen Ammeraal 著

刘瑞挺 等译

中国铁道出版社

2003·北京

（京）新登字 063 号

北京市版权局著作权合同登记号：01-2002-5413 号

版 权 声 明

图书在版编目（CIP）数据

C++程序设计教程：第 3 版/（美）安莫罗（Ammeraal,L.）著；刘瑞挺等译. 一北京：中国铁道出版社，2003.4

（21 世纪高等院校计算机教材）

ISBN 7-113-05205-3

Ⅰ.C… Ⅱ.①安…②刘… Ⅲ.C 语言-程序设计-高等学校-教材 Ⅳ.TP312

中国版本图书馆 CIP 数据核字(2003)第 026726 号

书　　名：C++程序设计教程（第三版）

作　　者：Leen Ammeraal

译　　者：刘瑞挺等

出版发行：中国铁道出版社（100054，北京市宣武区右安门西街 8 号）

策划编辑：严晓舟　魏　春

责任编辑：苏　茜　袁秀珍

封面设计：孙天昭

印　　刷：河北省遵化市胶印厂

开　　本：787×1092　1/16　印张：25.25 字数：551 千

版　　本：2003 年 6 月第 1 版　2003 年 6 月第 1 次印刷

印　　数：1～5000 册

书　　号：ISBN 7-113-05205-3/TP・917

定　　价：47.00 元

前　言

自从本书第一版、第二版相继问世以来，C++语言及其标准程序库都有了很大的改进。因此，本书第三版根据 ANSI/ISO C++标准所完善的 C++语言进行了全面的修订。例如，在第 1 章引入了新的 string 类的重要概念，并在本书中用它代替了 C 语言中的字符串。同时，由于从现有的函数程序库来看，编程人员在实践中经常会用到 C 语言中的字符串，因此我们也对它做了简要的说明。我们还在第 1 章中介绍了标准模板库（STL）的某些概念，并在第 9 章中对其进行了详细的讨论。

所有的样本程序和习题答案都可以从网址 http://home.wxs.nl/~ammeraal/ 或 http://www.tqbooks.net/“下载专区”免费下载。对于那些带星号的习题，则可在本书的附录 C 中找到答案。

在此，我对 C++语言的著名专家 Bjarne Stroustrup，Dan Saks，尤其是最先引起我关注 STL 的 Francis Glassborow 表示衷心的感谢，感谢他们通过 E-mail，特别是在牛津举办的欧洲开发者论坛上给我个人的直接指导和帮助。

Leen Ammeraal
l.ammerall@consunet.nl

译者序

C++正在成为许多程序，特别是大型软件开发的重要工具，因此它理所当然地成为许多大学进行计算机教学的首选课程，同时也成为社会广大进修人员进入编程世界的主要入口。

但是，国内外大学对如何进行C++的教学一直都存在着分歧。概括起来有两种意见：一种意见认为应该先学会C，再进一步学习C++，因为后者是从前者发展而来，这样既可以学会面向过程的程序设计，又可以学会面向对象的程序设计，可谓一箭双雕、事半功倍。另一种意见认为此言差矣！C是面向过程的，而C++是面向对象的，采取上述学法不仅事倍功半，而且简直是一无所成。不应该再学习C语言了，直截了当从C++开始学习，从一开始就打好面向对象的思想基础。

本书作者提供了第3种意见：C++是混合语言，而不是纯粹的面向对象的语言。学习C++既可以编写面向过程风格的程序；也可以编写面向对象风格的程序。如果要学习面向过程的编程，就干脆用C语言；要学习面向对象的编程，就彻底学习C++语言。因此，他并不把C/C++混合起来写书，而是分别写《C++ for Programmers》和《C for Programmers》。不难看出，作者基本上是支持第二种意见，而不完全同意第一种意见。所以本书开门见山地从一开始就按C++的新标准来讨论C++的重要编程技巧。

本书包括11章内容和三个附录。分别对C++概述，表达式和语句，其他运算符，函数与程序结构，数组、指针与串，类与对象，模板，异常处理，标准模板库，输入输出流以及早期的库函数进行了详尽的讨论。

参加翻译工作的有：韩毅刚、宋乃宁、刘瑞挺、宋杏珍、侯立新、宋学泽、刘淑芳、赵永欣、肖文俭、侯冬梅、谷新胜、王玉华、于浩、陈兰芳、程瑞芬、崔仙翠等。全国高等院校计算机基础教育研究会副理事长、《CHIP新电脑》总顾问、南开大学刘瑞挺教授对全书进行了修订和润色。

由于时间仓促，作者水平有限，错误在所难免，敬请读者批评指正，我们也会在适当的时候进行修订和补充，并发布在天勤网站：http://www.tqbooks.net“图书修订”栏目中。

译者
2003年4月15日
liu_ruiting@chip-china.com

目 录

1

绪论

1-1 C++及其他语言

C++编程语言是B.Stroustrup设计的，并于1986年出版了他的著作《C++程序设计语言》。C++由众所周知的C语言发展而来，C++这个名称来自C（++）程序中对变量C进行增值的表达式C++。

C++为面向对象的程序设计（OOP）提供了良好的工具，但作为一种混合式语言，它也支持传统的程序设计风格。因此，如果编程人员觉得有必要，可以在是否使用面向对象的程序设计上进行切换。从这一点看，C++又与纯的面向对象语言不同，如Smalltalk, Eiffel以及Java。从大多数C编程人员的角度看，C++仅是“一个更好的C”。除了OOP中不可或缺的、重要的类概念外，C++还包含了许多有用的语言概念和功能，它们在C中是没有的。在这里只简单涉及与函数有关的功能，其中包括函数重载、内联函数、缺省变量、类型的安全连接、函数模板以及在调用函数之前先要声明的简单要求。未声明的函数可以在C中使用，但在C++中不行。

以上所述的及其他内容使C++比C更安全。但是与一些别的语言不同，C++可提供与C同样的灵活性。这里所用的安全一词是指程序出现错误时发生的情况。从这一点看，汇编语言是极其“不安全的”，但这并不意味着用汇编语言编写的程序就不能完全正确和可靠。它们是可以做到的，同时C程序也是可以的。大多数有经验的程序员都希望尽可能多地控制计算机系统，因此与Basic等语言相比，他们更喜欢使用C或C++。如果一种语言既安全，同时又不受限制，大家就感觉很好。C++比C更安全，但并不比C更受限制。最老的专业编程语言是著名的Fortran和Cobol。它们的生命力很强：修订过的版本至今仍被应用程序员广泛地使用。

Fortran和Cobol在1960年以前就已存在，这一年诞生了Algol 60，它可以说是科学编程语言领域的一个里程碑。在20世纪60年代，正如Fortran在工业界普遍使用那样，Algol 60作为一种非常典雅而时髦的语言广泛用于学术界中。许多年来，用Algol 60编写的新算法发表于《ACM 通信》和许多其他科学期刊中。这种语言及其定义报告促使许多专家编写其后继的语言。其中最著名的是PL/1（同样以Fortran和Cobol为基础）、Pascal、Simula 67和Algol 68。由于它的定义报告极其复杂，Algol 68远不如Pascal应用得广泛，而且Pascal非常好的编译程序和相应书籍很快就应运而生了。

C++中通过使用类来隐藏数据的概念来源于Simula 67；而将运算符定义为新类型的可能性，作为C++的另一个重要特征，也早就出现在Algol 68中。比较新的语言是N. Wirth设计的Modula和来自美国国防部的Ada，我们知道Cobol也同样来自国防部。Modula和Ada都以Pascal为基础，而C、C++则与Algol 60、Algol 68存在更多的相似之处。例如，Algol 60及Algol 68中存在条件表达式（第3-1节中将详细讨论），而Pascal、Modula和Ada中不存在。C和C++中的简写符号也与Algol 68中的相似。Algol 60、Algol 68、C和C++这四种语言的另一个相似之处在于函数的返回值可被忽略，从而可将这样的函数作为过程来使用。从这些语言在学术界以外被接受的情况看，那些完全起源于美国的语言（Fortran，Cobol，PL/1，Basic，C，C++，Ada）要比大多数其他语言（Algol，Simula，Pascal，Modula）普遍得多。

使用方便的实际程序通常并不一定容易读懂。在修改复杂的程序时也很难保证不犯错误。当使用C++时，这种情况就要比使用一些别的语言好得多，这是因为我们可以定义自己

语言的扩展，即类。我们将这些写在不同的模块中，并且仅在主程序中使用该语言的扩展，或者说，将它用于应用模块中，从而使后者保持更为简单方便，而不是相反的情况。例如，编写一个执行很大数字的算术运算的程序时，若编程所使用的语言并不支持如此大的数字及相关运算符号+、-、*、/ 所具有的数据类型，那么该程序就会非常复杂。虽然 C++中也不具备这样内建的数据类型，但我们可以通过对应用程序的精确编程来解决这个问题，就好像该语言自身就支持这种数据类型一样。因此，我们可以说通过这种方法扩展该语言。用户定义的语言扩展，它的优点是更大的灵活性。这种取代内建实用程序的方法会为我们提供更多的东西。在详细探讨这些之前，我们必须尽量熟悉该语言的基本内容，其中许多也包含在 C 语言中。

如果你没有其他编程语言的经验，那么你的好处是不会为语言之间的区别而感到困惑，但你必须知道要想成为一名熟练的编程人员需要花很长的时间，而且只有通过不断的实际编程才能学会编程，因此不要忽视做练习。当然，这些也适用于那些有相关经验的人。本书包括大量的练习，你也可以选择其他的练习或自己开发一些程序，练得越多，掌握得越好。

1-2　非实用的程序

本书出现的大多数程序举例都不是真实的程序，这是因为每个例子都只对我们所讨论的某个对象进行说明。因此，它们比实际程序要短，实际程序往往要结合许多其他的东西。本节中我们将涉及一个与大多数其他程序相比更加不实际的程序，因为它所描述的每个操作都仅被执行一次。对于那些已经或多或少熟悉编程的人来说，就等于该程序中不包含任何条件语句或循环。下面给出这个程序，即从键盘读取两个整数 a 和 b，计算 $u=(a+b)^2$ 和 $v=(a-b)^2$。

```
// example1.cpp: A program to compute the squares of both
//               the sum and the difference of two given
//               integers.
#include <iostream>
using namespace std;

int main()
{  cout << "Enter two integers: ";  // Displays input request.
   int a, b;
   cin >> a >> b;                  // Reads a and b.
   int sum = a + b, diff = a - b,
   u = sum * sum, v = diff * diff;
   cout << "Square of sum      : " << u << endl;
   cout << "Square of difference: " << v << endl;
   return 0;
}
```

如果我们使用 MS-DOS 环境下的编译程序，如 Microsoft 和 Borland 所提供的某些编译程序的话，那么当我们使用程序编辑器编写完这个程序后，将它保存为 example1.cpp 文件。若使用 UNIX 操作系统下运行的 C++编译程序，则该文件的扩展名通常用.C 代替.cpp。实际上 PC

使用的大多数C++编译程序包，既包括一个C语言的编译程序，又包括一个C++语言的编译程序，而使用其中哪一个取决于文件扩展名。由于我们所写的程序包含了许多C++语言特有的要素，所以必须使用C++编译程序；如果使用普通C语言编译程序，则会显示许多错误信息。

经过编译和连接后，我们就可以执行这个程序。接下来屏幕上就会显示下列文字：

```
Enter two integers:
```

我们现在就可以输入，例如：

```
100 10
```

按下回车键后，出现下面的文字：

```
Square of sum          : 12100
Square of difference   : 8100
```

我们容易对这些值进行验算：a=100、b=10，那么和=110、差=90，将它们乘方后得u和v的值分别是12100和8100。

现在我们就结合所举的例子，简要地讨论一些C++语言的规则。将注释写在程序的开头是一种很好的习惯。可以通过两种方法来完成：第一种方法是在某一恰当的位置输入一对斜线（//），那么这对斜线和后面的程序行将被编译程序所忽略，例如：

```
// example1.cpp : 计算两个给定整数的和与差，并计算其平方值的程序
```

第二种方法是将注释写在一个程序行的结尾，来解释该程序行的前面部分，例如：

```
cin >> a >> b;                    // 读入a和b
```

在第1-8节中，我们将给出一种不同的、比较早的编写注释的方法，该方法基于一对字符：/*出现在注释的开始处，*/ 出现在注释的结尾处。

在程序开始处的注释后面接着：

```
#include <iostream>
```

我们说<iostream>是这个程序行中包含的一个头文件，即标题。虽然不需要在文字上过分推敲，但我们可以想象存在一个文件iostream，其内容在逻辑上代替了#include程序行。通常情况下，每个#include程序行必须占有一整个程序行。例如，我们不可在#include程序行结尾处写入程序文字int main( )。

由此看来，#include程序行就成为下面这条普遍规则的例外，该规则说：就C++编译程序而言，我们可以随意地将程序文本分成若干个程序行。例如，我们可将下面这个程序行：

```
int sum = a + b, diff = a - b;
```

替换为如下的两个程序行：

```
int sum = a + b,
diff = a - b;
```

我们可将这些程序行再进一步划分，但那样做显然不会提高其可读性。当将一个程序行划分为两个新的程序行时，我们就说插入了一个新行字符。类似的字符还有空格键和【Tab】键。将这三种字符共同称为white-space字符。

程序行：

```
using namespace std;
```

是必需的，这是因为我们使用了C++标准程序库中的名称cout、cin以及endl（很快将会涉及到）。如果我们用它们的完整写法std::cout、std::cin、std::endl来代替这里的名称缩写，

那就可以省略该程序行。而另外一种可能就是用以下三行程序代替上面的一行程序：

```
using std::cout
using std::cin
using std::endl
```

每个 C++程序都包含一个或多个函数，其中的一个称为 main（请注意我们这里并没有使用“函数”一词的抽象的数学意义，取而代之的是一个函数代表一个具体的程序段，即一个由字符组成的序列！）。在我们上面所举的例子中，它只含有 main 函数，其形式为：

```
int main ( )
{ ...
  ...
  return 0;
}
```

函数可以有，也可以没有参数。如果有参数，我们就将它写在括号中，这点在后面会讲到。如果没有，我们将继续使用空的括号，如上例所示。每个函数的“主体”都由一对大括号{ }括起来，如上例中所示的将这对大括号的两个括号分别写在（垂直方向上）同一列上，并将其中的内容缩格排列，这样写是一种很好的习惯。

在一个函数的开始大括号{ 后，我们写入所谓的语句。只要我们没有使用复合语句（将在第 2-4 节中进行讨论），那么每一条语句都要以分号作为结尾。在 example1.cpp 程序的 main 函数中，我们可以数出有 7 个分号，每个分号都表示一条语句的结束。通常一条语句正好写在一行上。然而，一行也可以同时存在多条语句，而且一条语句也可以占用多行，正如下面这条语句所示：

```
int sum = a + b, diff = a - b,
u = sum * sum, v = diff * diff;
```

这类语句称为一个声明。它说明变量 sum、diff、u 及 v 都属于 int 类型，说明它们代表的都是整数。每个变量和常量都具有一种类型（或称数据类型），事实证明类型是一个非常重要的概念。int 类型变量的值是有一定范围的：对于大多数电脑来说，该值的范围从-2^{31}（=-2147483648）到$+2^{31}-1$（=2147483647）。在上面的语句中，我们不但声明了变量 sum 、diff、u 以及 v，而且使用 a 和 b 的值来对它们进行初始化赋值。正如编程语言中经常见到的，星号*表示相乘。变量 a 和 b 在下面的语句中进行本身的声明：

```
int a, b;
```

但并未初始化赋值：它们的值在此声明后仍是不确定的。一个完全不同的语句是：

```
cout << "Enter two integers:";
```

这是 C++中典型的屏幕输出文字的方式。我们把 cout 称为标准输出流，并且可以通过操作符<< 向它发送字符。请注意该操作符由一对指向左边的箭头组成，因此它表明"Enter two integers:"双引号内的字符被发送到 cout 流中。有时，我们说打印这些字符，来代替将字符发送到 cout 输出流（或屏幕）的说法。类似的语句：

```
cin >> a >> b;
```

要求从标准输入系统（也就是从键盘）输入两个值，并将它们存储在变量 a 和 b 中。操作符>>表现为一对指向右边的箭头，因此数值从 cin 输入流发送到变量 a 和 b 中。当执行这条语句时，机器就会等待数据的输入，因此我们现在就可以按照要求输入两个整数。

当计算出 u 和 v 的值后，这些变量的值就处于一些文字和一个换行符之间，并通过下面的语句显示出来：

```
cout << "Square of sum          : " << u << endl;
cout << "Square of difference   : " << v << endl;
```

我们使用 endl 来表示已处于一行的结尾。在第 1-5 节中我们将会看到，也可以用'\n'或"\n"来代替 endl。而 endl 本身所代表的含义将在第 10-2 节中提到。同时，我们可自由地使用它。如果第一行中省略了 endl（或其类似的表示形式之一），那么输出结果将显示为：

```
Square of sum       : 12100Square of difference: 8100
```

在实际的输出结果中，如下所示，你会发现由于 8100 和 12100 第一个字母所处位置相同，而最后一个字母位置不同，它们并不是整齐的排列的。在第 2-5 和第 10-2 节中我们将会看到如何对此进行改善：

```
Enter two integers      : 100 10
Square of sum           : 12100
Square of difference    : 8100
```

1-3 存储器组织和二进制数

编译过的程序和数据都存储在计算机的存储器中。同时，计算机还有寄存器，可暂时存储数据。我们可以把存储器看成一个很长的字节序列，每个字节通常包括 8 个二进制数或位。每一位的二进制数只能有两个值：1 或 0。由于一个字节的长度不足以表示我们所要处理的大部分数字，因此将许多字节连起来形成一个机器码或简称为字，用来存放一个整数。现在实际上常用四个字节组成一个字，8 位二进制数组成一个字节，那么一个字就含有 32 位二进制数，我们将这些位从右向左依次标记为 0、...、31，如图 1-1 所示。

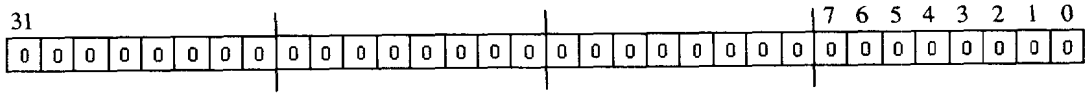

图 1-1 一个 32 位的字

该机器码包含下列二进制数的序列：

00000000 00000000 00000000 01010011

我们最少可以用两种办法来解释这个 32 位的二进制数：一是把它看作四个字符（每个由 8 位二进制数表示）；二是把它看作 32 位二进制数构成的一个整数。选用哪种解释方法取决于用来处理该字的程序。如果需将其看作一个整数，那么我们可用仅含 1 的位来计算它的值。忽略开始的 0，我们有 10110011，它的值是 83。该值可按以下公式计算：

$$1\times2^6+0\times2^5+1\times2^4+0\times2^3+0\times2^2+1\times2^1+1\times2^0=83$$

数字的二进制表示形式经常简化为更短（但不够准确）的二进制数，而后者更为常用，我们也常用此法表示。如本例所示，二进制数的值用 2 的 n 次幂来表示。我们就将 2 称为二进制数的基或基数。同样，10 是常用的十进制的基数。例如，我们可按照下面的式子计算十进制数 8241：

$$8\times10^3+2\times10^2+4\times10^1+1\times10^0$$

如果我们用同样的方法，使用一个（32 位二进制数）字的所有 32 位，我们就可将这些 2^{32} 个数字表示为：

00000000　00000000　00000000　00000000=0
00000000　00000000　00000000　00000001=1
00000000　00000000　00000000　00000010=2
00000000　00000000　00000000　00000011=3
…
11111111　11111111　11111111 11111111=2^{32}-1=4294967295

我们用“无符号整数”（unsigned int）一词（或更简单地，unsigned）来称呼这种表达式。许多实际应用中，我们也需要使用负数。最常用的方法是 2 的补数（two's complement）法。使用 32 位二进制数，我们可将整数-4294967296、…、+4294967296 表示如下：

10000000　00000000　00000000　00000000=-2^{31}= -4294967296
10000000　00000000　00000000　00000001=- (2^{31}-1)= -4294967295
…
11111111　11111111　11111111　11111110=-2
11111111　11111111　11111111　11111111=-1
00000000　00000000　00000000　00000000=0
00000000　00000000　00000000　00000001=1
…
01111111　11111111　11111111　11111111=2^{31}-1=4294967295

如果我们处理的数字与此例相同，也有负数，那么这些数字为带符号的整型数（signed int）。由于 int 型在缺省情况下是有符号的，因此我们通常用 int 代替 signed int。请注意，字长为 n 时，对于任意一个左边第一位为 1 的二进制数字序列，其没有符号的和有符号的形式转化后的结果相差 2^n。例如字长为 32，下面这个二进制数字序列。

10000000　00000000　00000000　00000000

若按 unsigned int 型转化结果为 2^{31}，而若按 signed int 型转化结果为-2^{31}，这两个值之间的差为 2^{31}-（-2^{31}）=2×2^{31}=2^{32}。

通常情况下，还有 16 位二进制数的相对较短的整数（有符号的和没有符号的）形式。于是，unsigned int 型的最大值为 2^{16}-1=65535，而 signed int 型的最大值为 2^{15}-1=32767。

在结束本节内容时，我们再提及另一种重要的二进制数字序列的表示方法，即转化为数字的十六进制表示形式。我们可将机器码分为若干个部分，每部分含 4 位二进制数。在十六进制系统中，基数是 16，因此它有 16 个数字，其二进制表示形式分别如下：

0000=0
0001=1
…
1001=9
1010=A（=10）
1011=B（=11）

1100=C（=12）

1101=D（=13）

1110=E（=14）

1111=F（=15）

现在考虑下面这个含有32位二进制数的字，其中每4位二进制数为一组，共分8组：

1000 0000 0000 0000 1111 0000 1010 0011

这个二进制数的序列转化成相应的十六进制表示形式为：

800F0A3

我们已知F=15，A=10，将其带入计算可得：

$8\times16^7+15\times16^3+10\times16+3=2147545251$

这是计算 32 位二进制数组成的字的另一种方法，当然这时认为它是不带符号的二进制数。当我们想将其作为带有符号的二进制数字进行转化时，将该值减去 2^{32}，得：

2147545251 – 42949672956= -2147422045

存储单元中的内容（字节或字）是经常变化的：它们可以在程序执行过程中不断地变化。与此相反的是，每个存储单元都有一个固定的数字，用来表明其位置的地址。在高级语言中，我们通常不知道变量的地址。这一点在C++中也适用，但是在这种语言中，我们用一些象征性的符号来表示这些地址。例如，我们将在第5章以及其他地方看到，用&a表示变量a的地址。

1-4 标识符和关键字

变量名在技术上称为标识符。一个标识符写作一个字符序列，其中只能出现字母、数字和下划线（_），而且第一个字符不能是数字。由于一些系统函数的名称往往使用下划线开始，以便与其他标识符相区别，因此最好选择以字母（而不是下划线）开头的标识符。字母的大小写是不同的，因此就有52个不同的字母。一个标识符中所有的字符都是有意义的。以下是几种标识符的正确表示形式：

```
a
largest_element
table1
```

下面的标识符是作为关键字而保留起来的，因此我们不可以将它们用于变量名或其他一些类似的用途。

```
Asm,auto,bool,break,case,catch,char,class,const,const_cast,continue,
default,delete,do,double,dynamic_cast,else,enum,explicit,export,extern,
false,float,for,friend,goto,if,inline,int,long,mutable,namespace,new,operator,
private,protected,public,register,reinterpret_cast,return,short,signed,
sizeof,static,static_cast,struct,switch,template,this,throw,true,try,
typedef,typeid,typename,union,unsigned,using,virtual,void,volatile,
wchar_t,while.
```

此外，对于以下这些作为特定运算符的替代标识符，我们也不能随便使用。在这些标识符后面的括号中列出了相应的运算符，我们将会对这些运算符加以讨论。

```
and (&&),and_ eq (&=), bitand (&),bitor (|), compl (~), not (!), not_ eq (!=),
or (||), or_ eq (|=), xor (^), xor_ eq (^=).
```