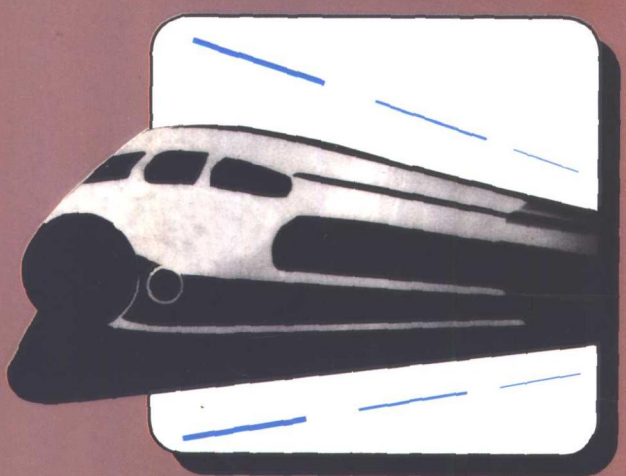
电脑知识直通车系列丛书

C++语言直通车

张 蕊　刘振安　编著

西安电子科技大学出版社



C++ C++ C++ C++ C++ C++ C++ C++ C++ C++

● 电脑知识直通车系列丛书

C++语言直通车

张　慈　刘振安　编著

西安电子科技大学出版社
1998

内 容 简 介

从C语言转向C++语言，对于初学者来说，最重要的是如何尽可能快地达到“入门”水平，即理解面向对象的基本原理，能够读懂C++ 程序，并能够编写简单的C++ 程序。本书目的是带领读者快速步入C++ 的殿堂。

本书语言生动活泼，比喻通俗易懂，概念清楚，重点突出类的概念及设计方法，从而使初学者尽快得到完整的C++编程概念和实际应用技巧的锻炼。为了方便学习，本书还介绍了上机及程序调试方法。

本书适合作为各类业余计算机培训班的C++ 语言教材及广大计算机爱好者的自学读本。

图书在版编目(CIP)数据

C++语言直通车/张蕊等编著. —西安：西安电子科技大学出版社，1998.8
(电脑知识直通车系列丛书)
ISBN 7-5606-0637-7

Ⅰ.C… Ⅱ.张… Ⅲ.C语言-基本知识 Ⅳ.TP312

中国版本图书馆CIP数据核字(98)第15347号

责任编辑 马乐惠
出版发行 西安电子科技大学出版社
(西安市太白南路2号)
邮 编 710071
电 话 (029)8227828
经 销 新华书店
印 刷 西安长青印刷厂
版 次 1998年8月第1版
1998年8月第1次印刷
开 本 787×1092毫米 1/16 印张 8.75
字 数 198千字
印 数 1～6 000册
定 价 11.50元
ISBN 7-5606-0637-7/TP·0322

操作简单就是享受

（代　序）

□ 乔　岗

自电脑诞生后，计算机软件就伴随着硬件的进步越来越向用户透明。也就是说，用户能够更简单、方便地使用软件。如果让计算机家电化，使操作电脑像使用电视机一样简单，那么电脑的普及便是指日可待的了，这也是电脑工作者孜孜以求的目标。

软件朝着多功能化方向发展的目的就是要简单化。比如说，低版本的DOS在删除子目录时，必须从底层向上逐级进行，相当繁琐，而在高版本DOS中使用DEL TREE外部命令就可以干净利落地解决问题。再如初期的视窗95软件对硬盘的要求是2 GB，超过2 GB就要对硬盘分区，而视窗98就突破了这个限制。“酒吧”(WIN98)因此比“酒屋”(WIN95)功能更强。

但是，并非软件功能越多使用起来就一定简单。微软的IE4.0是一个十分了得的“小软件”，其所占存储空间居然超过了10 MB，其实普通网络用户很少用到“网络电话(NetMeeting)”等等硬塞进来的功能。软件的功能既多又强固然好，但如果让用户一头雾水地找不着北，还不如选择功能虽弱但操作简单一些的。眼下电脑遇到了与一些家电一样的尴尬——新产品功能很多但使用起来却很令人头痛。有些流行的“家庭影院”功能多的让人无所适从，操作起来就像在做数学习题。凡此种种让那些狂热的电脑迷和追新者伤透了脑筋，也倍感无奈。

人人都喜欢新东西，但不希望它太复杂，因为操作简单才是享受。

这套丛书编辑出版的目的，就是为了寻找一个新的平衡点：使用户既能及时享用优秀软件，又感到操作起来十分简单方便；既能顺利完成日常工作，又不必太费脑子。这是一个“鱼与熊掌”二者得兼的选题，极富挑战性。让人高兴的是，这套丛书的作者们在电脑图书写作方面经验多多，他们出版过多部有关计算机的图书，对中国读者的阅读兴趣和阅读习惯有较深的研究，因此敢于面对挑战。出版社和作者们联手奉献给广大读者的这套丛书，是一列精心设计的软件科技快车。只要上了车，哪怕是糊里糊涂，它也会带您抵达目的地——这些全是“直通车”。

其实，很多出版机构都会碰到这样的事：在已经编辑出版的众多计算机图书中，竟然找不到一本适合培训社内员工的教材。想一想也真是，又有哪一位

作者、哪一家出版社有底气地说：我们所出的每一部书都是精品，句句都经得起推敲，都适用于各自所面向的读者。笔者的书柜里摆了很多厚厚的计算机图书，从国内到国外，从软件到硬件，从PC到网络应有尽有。但说句真心话，我很少去动它们。这不是自己懒，只是实在没有必要。该知的早已烂熟于心，不知的工作上一时又用不上的就不想知道，要用时现学也不迟。在单位里，搞计算机的人常被同事尊称为电脑专家。我相信专家们多是靠“半部论语治天下”的，因为那些大部头的电脑书太厚，命令多得记也记不住，逼急了才去查资料。DOS有100多条命令，平时能用到一半的人恐怕没有，谁也不会傻背DOS命令玩儿。连专家都这样没有耐心，让普通读者捧着“砖头”受洋罪，多少有些“不人道”吧。

这套丛书就是冲着“简单、够用”四个字来的。作者们人道一些，把书写薄点儿，读者们就会轻松许多，否则没有面子的又是这些电脑专家们。我有一个朋友久闻Word大名，决心学一学，找来一本入门书刻苦攻关，但到最后自认不行，重拾WPS。对一个没有多少基础的初学者一上来就是几十页的新名词、几十页洋字码的安装过程，谁也受不了。多亏我这位朋友性子慢，一般人早就把书扔了。这不是因为Word不好，而是写书的作者领错了道，误导了读者，摧毁了他的自信心，同时也糟蹋了Word，这恐怕也是一种不人道行为。初学者的特点是最怵生僻的新名词，最怯繁琐的安装过程。“直通车”丛书在编写之前，就对每位作者提出了要求，统一了把读者放在第一位的思想：不要怕“婆婆妈妈”，不要怕“琐琐碎碎”，要站在读者角度，要细心地照顾读者，要周到……

对初学者，简单、细致、周到都是福音。

有了“直通车”，朋友们可以把使用电脑真正当成一种享受。

欢迎您“搭乘”电脑知识“直通车”！

前　言

本书以轻松的格调介绍C++程序设计，尤其适合于那些刚刚从C语言转向C++语言的非计算机专业应用人员，它将带领读者快速步入C++的殿堂。

近年来，计算机的广泛应用使得大批非专业人员坐到了计算机前，而形势的进一步发展又要求他们迅速从面向过程转向面向对象，从C语言转向C++语言。对于他们来说，最重要的是如何尽可能快地达到"入门"水平，即理解面向对象的基本原理，能够读懂使用C++语言编写的程序，并能够使用C++语言编写简单的C++程序。为了满足这一类读者的需要，我们编写了本书。

全书共9章。第1章和第2章介绍C++语言对C语言的一些改进；第3章和第4章介绍面向对象设计方法的一些基本概念和类的设计；第5章讨论类的派生，即如何通过已有的类来产生符合需要的新类；第6章介绍运算符重载；第7章介绍输入输出及文件设计；第8章讨论如何使用C++语言进行面向对象程序设计的基础知识，并结合设计实例介绍设计应用程序的基本方法；第9章给出C++程序的调试方法。

本书语言生动活泼，比喻通俗易懂，不涉及高深的理论知识，从而使读者能在轻松的气氛中学习C++程序设计，所以也很适合作为各类业余计算机培训班的C++语言教材。

作　者

1998年5月

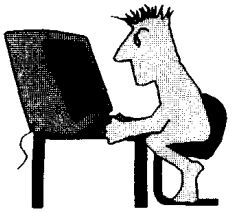
目　录

第1章 似曾相识燕归来

认识一个人，首先要从面孔开始。本章介绍C++的最基本的面孔组成——输入、输出、定义常量及注释等语句。因为这些都是C++语言对C语言的改进，感到似曾相识也就不足为怪了。

本章主要讨论如下问题：

- C++语言应运而生。
- C++语言的输入和输出语句。
- C++的换行语句。
- C++语言的注释语句。
- 为什么C++语言不再提倡宏定义。
- C++语言的const修饰符。



1.1 C++语言的来龙去脉

C++语言是C语言的扩充。在1980年，贝尔实验室的Bjarne Strotstrup博士及其同事开始对C语言进行改进和扩充，最初被称为“带类的C”，1983年才取名为C++。以后又经过不断完善和发展，成为目前的C++语言。一方面，它将C语言作为它的子集，使它能与C语言兼容，使许多C语言代码不经修改就可以为C++语言所用以及用C语言编写的众多的库函数和实用软件可以直接用于C++语言中；另一方面，C++语言支持面向对象的程序设计，这是它对C语言的最重要的改进。目前，C++语言已被应用于程序设计的众多领域，实践证明，它尤其适用于中等和大型的程序开发项目。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、可维护性等方面都显示出C++语言的优越性。

本章将讨论C++语言对C语言的一些改进，这些改进并不涉及面向对象的程序设计。尽管如此，我们仍将看到，与C语言相比，用C++语言编写的程序可读性更好，代码结构更为合理。通过本章的学习，相信具有C语言基础的读者，一定会很快转到C++语言上来。

1.2 输入和输出的新面孔

让我们通过比较一个简单的C程序的例子入手，介绍C++语言的风格。

【例1.1】 编写一个程序，输入一个人的姓名，然后输出"HELLO，***!"。

我们先用C语言来编这个程序，源程序如下：

```
#include <stdio.h>
main( )
  {
    char * name;
    printf("%s","Please input your name: ");          /* 输出信息 */
    scanf("%s",name);                                 /* 输入名字 */
    printf("Hello,%s! \n",name);
  }
```

C语言的输入/输出通过函数来实现，使得C语言更加精练。但它对于不同类型的数据又采用不同的控制字符，使人很难记住，也就难以轻松。

下面我们用C++语言来编写一个功能完全与此相同的程序。

```
#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
  }
```

通过比较上面两个程序，我们可以看出，C++语言在输入/输出方面对C语言作了较大改进。

C++ 语言的输入/输出是通过流来实现的。关于流，我们将在后面详细介绍。读者现在只需要知道，C++是自带输入和输出的，并且可以根据数据的类型自动使用合适的输出方式。

现在再来看我们的C++ 程序。在程序的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。这与C语言的道理一样，只是C语言的头文件是stdio.h。

第五行的"cout << "是告诉计算机，把后面的内容送到标准输出设备(这里是显示器)。同样，"cin >>"是告诉计算机，把标准输入设备(键盘)接收到的数据存入后面的变量。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"得名的原因。

当然，C风格的输入输出在C++ 中也是允许的。显然C++ 风格的输入/输出更方

便、更让人感到轻松！

1.3 灵活的注释方式

C++ 提供了一种新的注释方式：从“//”开始，直到行尾都将被计算机当做注释。例如：

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

另一方面，C 风格的多行注释在 C++ 中也仍然可以使用。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“//”。

1.4 告别宏定义

在C 语言中，宏定义是一个重要内容。无参数的宏作为常量，而带参数的宏则可以提供比函数调用更高的效率。在C++ 中，由于 const 修饰符和内联函数的引入，无论是带参数的宏还是不带参数的宏，都失去了存在的必要。也就是说，它们可以“光荣退休”了。

首先我们来看 const 修饰符。用类型修饰符 const 声明的变量只能被读取。该变量必须在声明时定义(即初始化)，并且它的值在程序中不能被改变。如果在程序中企图改变这个变量的值，则是非法的，会出现编译错误。

读者一定还记得这样的宏定义：

```
#define PI 3.1416
```

下面我们就来看看，const 是怎样代替它的。

【例 1.2】 输入圆的半径，输出圆的面积和周长。

```
#include <iostream.h>
const float pi=3.1416;          // 定义 pi
void main( )
  {
    float r,s,c;
    cout << "r=";
    cin >> r;
    s = pi * r * r;
    c = 2 * pi * r;
    cout << "s=" << s << endl;
    cout << "c=" << c << endl;
  }
```

const 修饰符的使用就是这么简单。事实上，对基本数据类型的变量一旦加上 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。使用 const 可以使编译器对处理内容有更多的了解，

从而允许对其进行类型检查，同时还能避免对常量的不必要的内存分配，并可改善程序的可读性。

const 还可以修饰指针变量。以 char 类型为例，有以下三种情况：

(1) const char * p；这表明 p 是一个指针，它只能指向一个被 const 修饰的 int 类型的变量，即只能指向一个整型常量，例如：

```
const char *p;
const char var="5555";
p=&var;
```

(2) char * const p；这表明指针 p 本身是常量，即 p 指向一个固定的 char 类型的地址，而 p 的内容却是可以修改的，例如：

```
char const * p="abcd";
p[2]=a;
```

(3) const char * const p；这里指针和它所指的内容都是常量，对任何一个进行修改都是错误的。

1.5 完美的标志

我们要求读者千万别小看这些新面孔，而且应时刻记住这些新面孔。读者可能会认为：C++语言兼容 C 语言，不使用这些新语句，程序一样可以正确编译并运行。

说的确实不错。人们同样也会说："这是一个新手编的程序，到处流露出 C 的不良痕迹。"

这些面孔是代表程序员是否进入 C++的一个标记，而且也容易熟悉并记住，所以还是应该快点轻松愉快地给自己的程序作个完美的 C++标志。

记住如下五条：

(1) 使用 cout 代替 printf；

(2) 使用 cin 代替 scanf；

(3) 短的注释使用"//"；

(4) 使用 const 代替 #define；

(5) 使用 endl 代替\n。

第2章 小荷才露尖尖角

上一章讨论了C++语言对C语言的修改所呈现的新面貌。毫无疑问，C++决不会停留在对C语言的修改上。C++的一些新特征就像刚出水的小荷，露出可爱的尖尖角，引人入胜。

在讲述对象或其它面向对象的构造之前，本章先进一步讨论C++语言突出的一些新特点，以便为学习C++语言的类打下基础。

本章主要讨论如下问题：

- 为什么C++语言特别强调函数原型。
- 使用C++语言的缺省参数。
- C++语言的new和delete语句。
- 内联函数。
- C++语言的引用方式。
- 面向对象的概念。
- 抽象、封装和多态性。

2.1 函数原型显身手

在C++中，函数原型是一个很重要的概念。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。

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类型说明，其个数和指定的类型必须和函数定义中的一致。例如：

```
int sum (int, int);
```

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。

2.2 缺省参数暗中助

C++ 语言的实现者为我们考虑得非常周到。一方面，他们总是在设法减少编码的复杂程度，另一方面，他们又使得编译器能检查出更多的错误。在函数调用时，他们引进了一种新类型的参数：缺省参数。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。当程序员需要传递一个与预先设定不同的值时，必须显式地指明。缺省参数是在函数原型中说明的。例如：

```
int SaveName(char * name, char *last_name=" ");
```

缺省参数无论有几个，都必须放在参数序列的最后，例如：

```
int SaveName( char * first, char * second = " ",
              char * third = " ",
              char * fourth = " ");
```

这种方式表明，在实际调用函数 SaveName( )时，调用者可以忽略参数 second、third 和 fourth。如果一个缺省的参数需要指明一个特定值，则在其之前的所有参数都必须赋值。在上面的例子中，如果需给出参数 third 的值，则必须同时也对 second 赋值，例如：

```
status=SaveName("Alpha", "Bravo", "Charlie");
```

2.3 同时上阵双包胎——new 和 delete

用 C 语言编程，少不了和指针打交道，而使用指针，又常常遇到动态内存分配。在 C 语言中，动态内存分配是通过系统函数 malloc( )、free( )和运算符 sizeof 来实现的；而在 C++中，上述用法已经很少使用了，取代它们的是 C++ 的两个运算符：new 和 delete。

运算符 new 用于进行内存分配，它的使用形式为：

```
p = new type;
```

其中 type 是一个数据类型名，p 是指向该类型的指针。new 的最大优点是不必进行类型转换。例如为一个包含 5 个整数的数组分配内存，只需要写成：

```
int *ip;
ip=new int[ 5 ];
```

甚至可以直接写成如下形式：

```
int *ip = new int[ 5 ];
```

上述用法比使用 malloc( )要简单得多。

运算符 delete 用于释放 new 分配的内存，它的使用形式为：

```
deltete p;
```

其中 p 必须是一个指针，指向 new 分配的内存的首址。下面我们来看一个例子。

【例 2.1】 演示 new 和 delete 的用法。

```
#include <iostream.h>
void main ( )
  {
    int * pi;
    pi = new int;
    * pi = 5 ;
    cout << * pi;
    delete pi;
  }
```

上面这个程序是用 new 建立的变量来初始化指针 pi，在该变量不再使用之后，又使用 delete 将其撤销。new 所建立的变量的初始值是任意的，所以程序中使用语句：

```
* p = 5;
```

为该变量置初始值。也可以在用 new 分配内存的同时为其指定初始值，例如：

```
pint = new int(5);
```

这和使用 malloc( )一样，当不能成功地分配所需的内存时，new 返回空指针 0。例如：

```
int * p = new[1000];
if (p == 0)
    cout << "run out of memory";
```

这在实际中也常常用到。

2.4　内联函数攻难关

下面我们再来看内联函数。先看一个例子：

【例 2.2】 编写函数 Myabs，求绝对值。

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

这个函数很简单，但很有用，在许多程序中都要用到，但对于这样一个简单的函数，使用函数调用的开销(尤其是程序中多次调用这个函数时)所带来的好处，有时却不足以补偿使用这个函数所带来的坏处。在这种情况下，我们过去总是通过带参数的宏进行替换。但宏有时又会有副作用，特别是遇到++和--的时候。为此，C++ 引入了内联函数。

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。于是我们达到了参数宏的全部功能，同时由于是用函数体进行替换，我们又避免了参数宏的副作用。

2.5 引用打开新局面

引用的引入是 C++ 对 C 语言的一项重大改进，同时也是本章的重点和难点。我们在以后的章节里将多次用到引用，因而有必要在这里花较多的篇幅进行讨论。

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。因而原变量和新变量——规范地说是变量和它的引用——总是具有相同的值。

C++ 是通过引用运算符 & 来定义一个引用的。如果这个引用不是用作函数的参数或返回值，则在说明时必须进行初始化。例如：

```
int num=50 ;          // 这个变量必须赋值
int & ref = num ;
```

ref 被说明是一个 int 类型变量的引用，所以要用 int 类型的变量 num 给其置初始值，我们称 int 为 ref 的引用类型。下面的程序说明引用的使用。

【例 2.3】 引用的使用。

```
#include <iostream.h>
void main( )
  {
    int num=500;
    int & ref = num;
    ref = ref+100;
    cout << "num=" << num << endl;
    num=num+50;
    cout << "ref=" << ref;
  }
```

程序的输出是：

```
num = 600
ref = 650
```

在这个例子里，我们首先定义一个 int 类型的变量 num，并给它赋初值 500。然后我们又定义了一个 int 类型的引用 ref，并将它和 num 相联系。这样，无论我们是对 num 还是对 ref 进行操作，实际上都是对那个一开始放着 500 的物理单元的内容进行操作，于是我们就得到了最后的输出。

需要指出的是，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。例如：