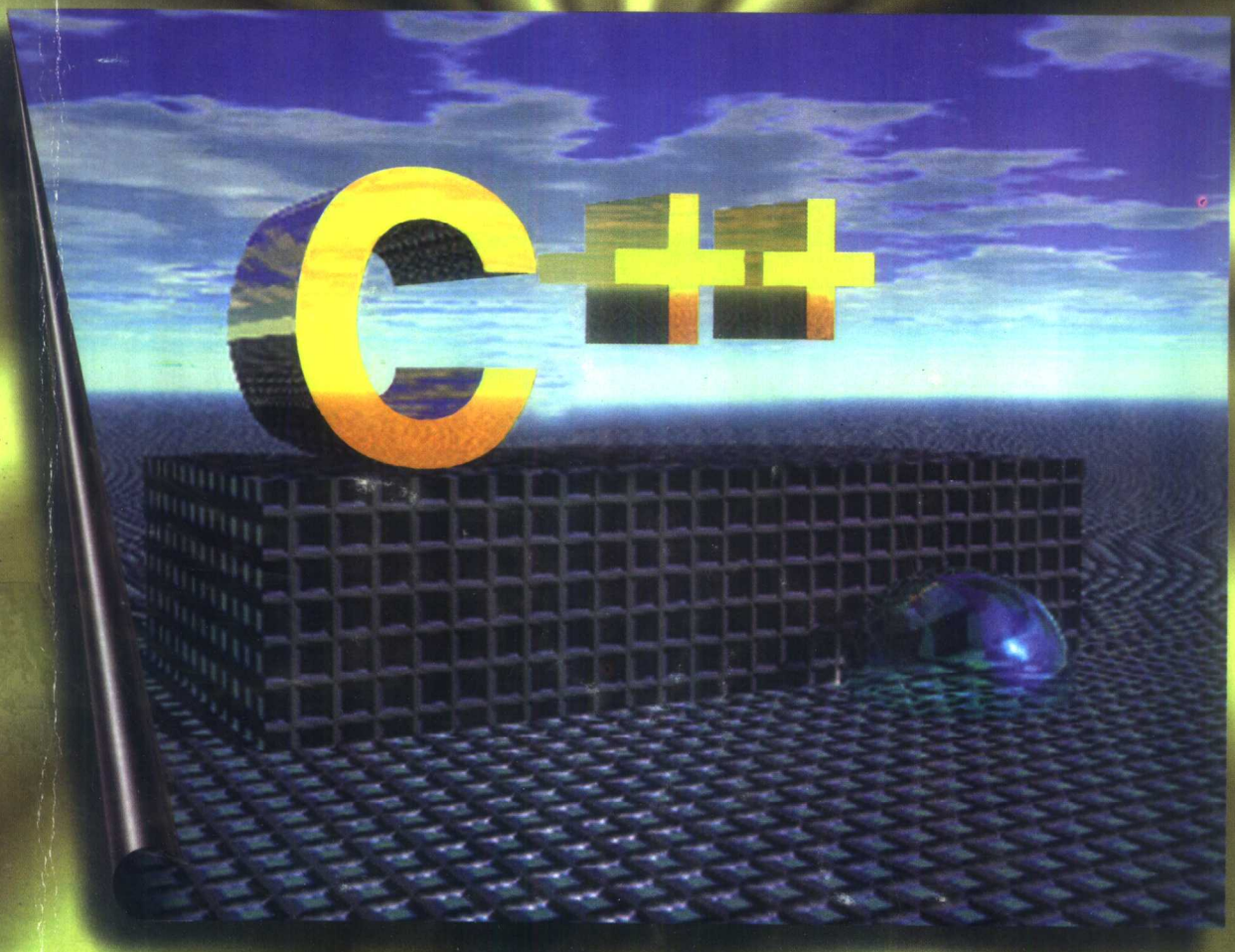
青松

# C++程序设计

刘振安　刘大路　编著

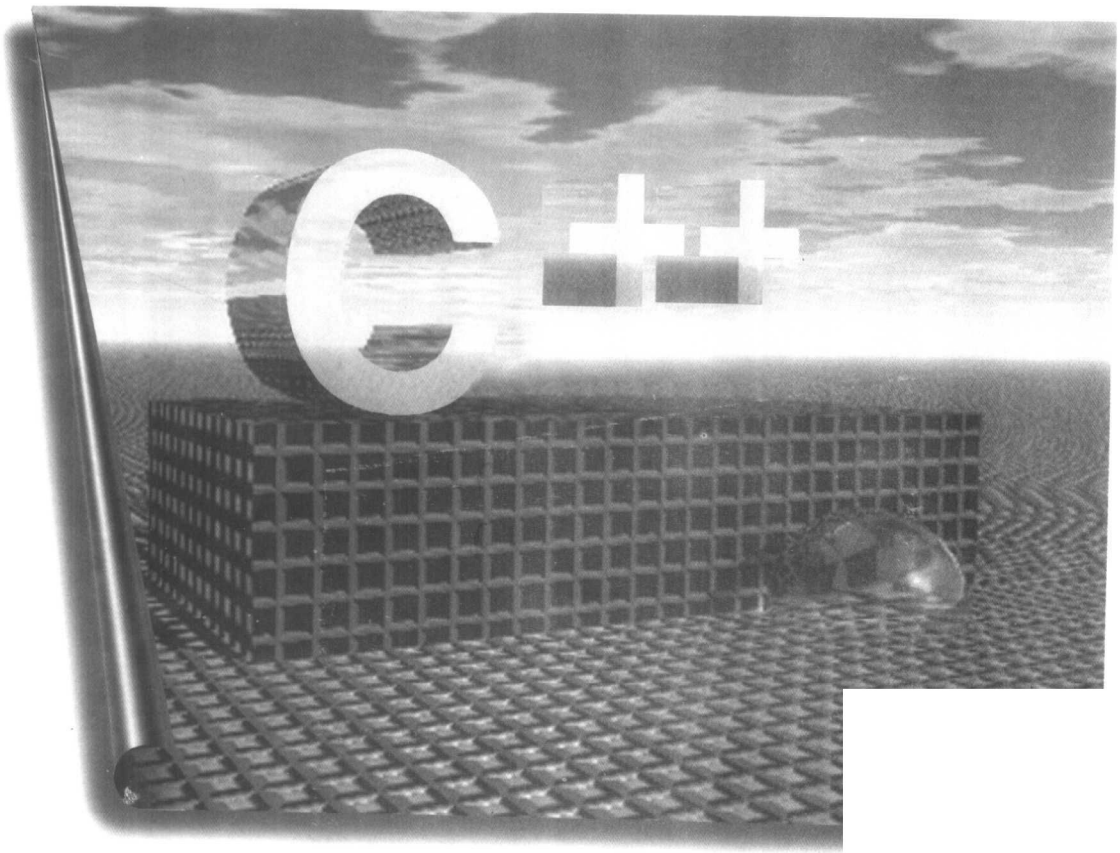



青岛出版社

青松

# C++程序设计

刘振安　刘大路　编著




青岛出版社

**鲁新登字 08 号**

**图书在版编目(CIP)数据**

C++程序设计/刘振安,刘大路编著. —青岛:青岛出版社,1998.9
ISBN 7-5436-1935-0

Ⅰ.C… Ⅱ.刘… Ⅲ.C 语言-程序设计 Ⅳ.TP312

中国版本图书馆 CIP 数据核字(98)第 22580 号

责任编辑 尹红侠 张化新
封面设计 辛 隆

*

青岛出版社出版
(青岛市徐州路 77 号)
邮政编码:266071
新华书店北京发行所发行
青岛双星集团华信印刷厂印刷

*

1999年1月第1版 1999年1月第1次印刷
16开(787×1092毫米) 11.75印张 270千字
印数1—3000
定价:19.00元

# 前言

本书以轻松的笔调介绍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，最重要的是如何尽快地达到入门水平，即理解面向对象的基本原理，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，中国科学技术大学黄山工作室编写了本书。

全书共九章。第一章和第二章介绍C++语言对C语言的一些改进；第三章和第四章介绍面向对象设计方法的一些基本概念以及类的设计；第五章讨论类的派生，即如何通过已有的类来产生符合需要的新类；第六章介绍运算符重载；第七章介绍输入输出及文件设计；第八章介绍模板；第九章讨论如何使用C++语言进行面向对象的程序设计，并结合设计实例介绍设计应用程序的基本方法；附录给出最常见的C++语言编译环境的操作方法及习题参考答案。

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

刘振安

1998年2月

# 目录

# 第一章 熟悉C++的面孔

认识一个人，首先要从面孔开始。本章也先介绍C++的最基本的面孔组成——输入、输出、定义常量及注释等语句。其实，这些都是C++语言对C语言的改进。

本章主要讨论以下问题：

(1) C++语言简介；

(2) C++语言的输入和输出语句；

(3) C++语言的注释语句；

(4) 为什么C++语言不再提倡宏定义；

(5) C++语言的const修饰符。

# 1.1 C++语言简介

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，这就使许多C语言代码不经修改就可以为C++语言所用，用C语言编写的众多的库函数和实用软件可以用于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，与C语言相比，用C++语言编写的程序可读性更强，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 C++输入和输出的新面孔

首先从一个简单的C语言程序例子入手，来介绍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，***!”。

用C语言编写的源程序如下：

```
#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语言的输入与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。

下面用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，可以看出，C++语言在输入与输出方面对C语言做了较大改进。

C++语言的输入与输出是通过流来实现的。关于流，本书将在后面详细介绍。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。

在例1.1的C++程序中，第一行包含了一个头文件iostream.h。有了它，就可以使用C++风格的输入与输出。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">>"看做表示方向的符号，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风格的输入与输出在C++中也是允许的。但是，难道读者不觉得C++风格的输入与输出更方便、更让人感到轻松吗？

## 1.3 C++灵活的注释方式与告别宏定义

### 1.3.1 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。一般地，多行注释仍旧使用"/* …… */"，而短的注释则较多使用行注释方式"//"。

### 1.3.2 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。

首先来介绍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被定义(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

下面举例说明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，对于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; 这表明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。

## 小 结

这里希望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,程序一样可以被正确编译并运行。

说得确实不错。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++ 的一个标志,而且读者可以很容易地记住这些面孔,所以还是应该快点给自己的程序做一个完美的 C++ 标记。

请记住以下五条:

(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。

## 习 题

1. 阅读下面的程序,并将此程序改写成 C++ 程序。

```
// 从键盘输入一个大写字母,改写成小写字母并输出
#include "stdio.h"
main()
{
  char c1,c2
  c1=getchar()
  printf("%c,%d\n",c1,c1)
```

```
    c2=c1+32
    printf("%c,%d\n",c2,c2)
}
```

2. 输入四个整数，按从大到小的次序输出。

3. 将下面的程序改写成 C++ 程序，注意不要使用宏。

```
#include "stdio.h"
#define PI 3.1416
#define R(d) d * PI/180   //将角度换算成弧度
main()
{
  float degree
  printf("%f\n",R(degree))
}
```

4. 使用 const 运算符，编程计算球体的表面积和体积。

# 第二章 展示C++的新特征

上一章讨论了C++语言对C语言修改后所呈现的新面貌。毫无疑问，C++决不会停留在对C语言的修改上。本章将展示C++的一些新特征，让读者领略C++独特的编程风格和设计思路。

本章主要讨论以下问题：

（1）为什么C++语言特别强调函数原型；

（2）C++语言的缺省参数；

（3）C++语言的new和delete语句；

（4）内联函数；

（5）C++语言的引用方式；

（6）面向对象的概念；

（7）抽象、封装和多态性。

## 2.1 C++函数原型显身手

在C++中，函数原型是一个很重要的概念。任何函数缺少了函数原型，C++都将无法对其编译。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)；

其中参数类型说明列表是用逗号隔开的一系列类型说明，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 C++缺省参数真方便

C++语言的设计者为用户考虑得非常周到。一方面，他们总是在设法降低编码的复杂程度，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 C++新的李兄弟——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。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。如果为一个包含5个整数的数组分配内存，就只需要写成：

int *ip;

ip=new int[5];

甚至可以直接写成以下形式：

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。它的一般使用形式为：

delete p;

其中p必须是一个指针，指向new分配的内存的首址，下面来看一个例子。

【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：

```
* pi = 5;
```

来为该变量置初始值。也可以在用 new 分配内存的同时，为其指定初始值，例如：

```
int * pi = new int(5);
```

这和使用 malloc()一样，当不能成功地分配所需要内存时，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 C++ 内联函数解疑难

下面讨论内联函数。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(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，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“--”运算符的时候。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，于是就达到了带参数的宏的全部功能。同时由于是用函数体进行替换，又避免了带参数的宏的副作用。