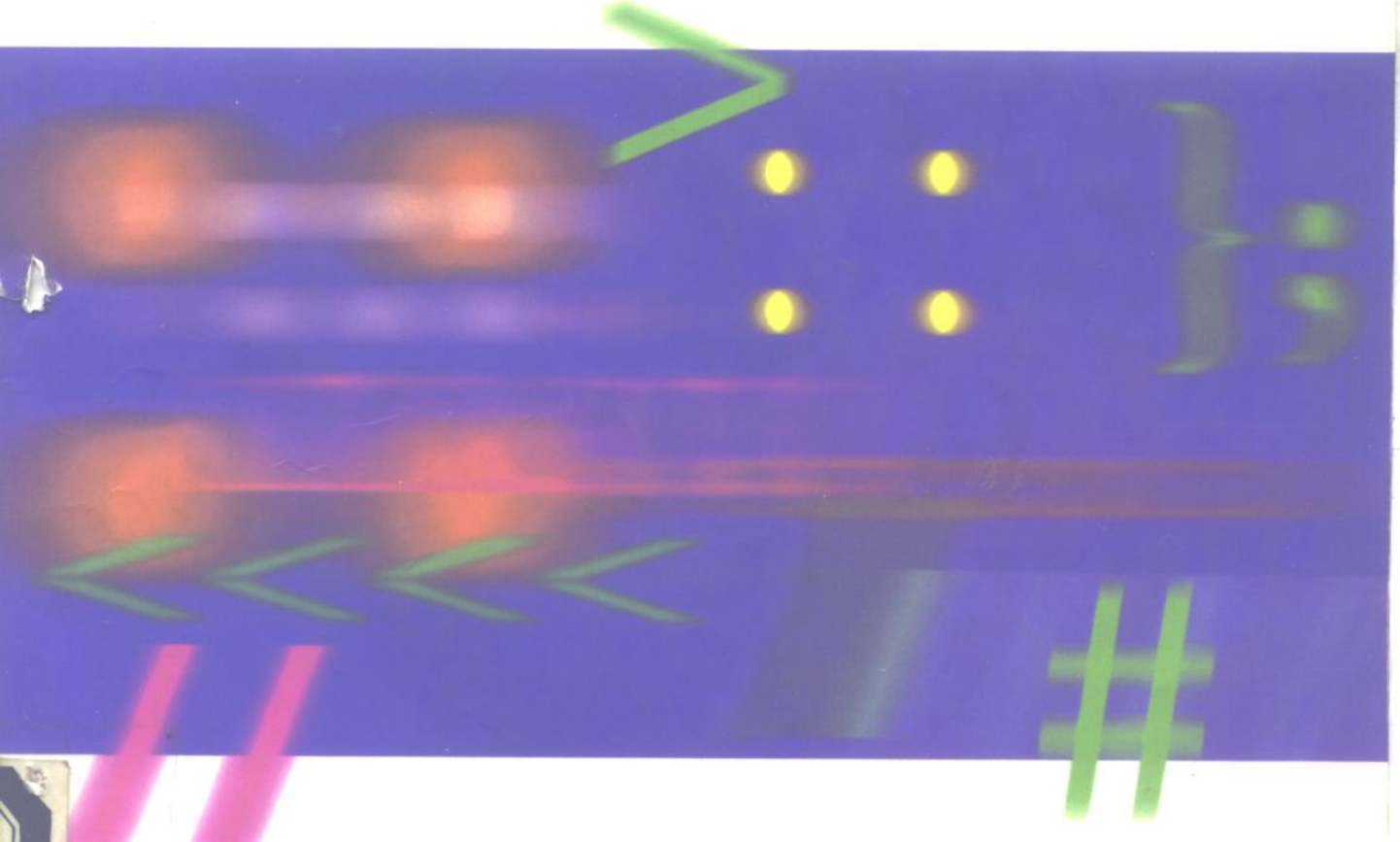
最新C++应用编程技术

[美] 斯蒂芬·布莱哈 著
孟庆昌 刘振英 牛欣源 黄中原 译
孟庆昌 校



国防工业出版社

INTERNATIONAL THOMSON COMPUTER PRESS

最新C++应用编程技术

〔美〕斯蒂芬·布莱哈 著

孟庆昌 刘振英 牛欣源 黄中原 译

孟庆昌 校

国防工业出版社

·北京·

著作权合同登记　图字：军-1996-007

图书在版编目(CIP)数据

最新C++应用编程技术/(美)布莱哈(Blaha,S.)著；
孟庆昌等译．—北京：国防工业出版社，1997.5
书名原文：C++for Professional Programmers with
PC and UNIX Applications
ISBN 7-118-01651-9

Ⅰ.最…　Ⅱ.①布…②孟…　Ⅲ.C语言-程序设计
Ⅳ.TP312C

中国版本图书馆CIP数据核字(96)第15472号

國防工業出版社出版发行
(北京市海淀区紫竹院南路23号)
(邮政编码　100044)
北京杯柔新华印刷厂印刷
新华书店经售
*
开本787×1092　1/16　印张25¾　591千字
1997年5月第1版　1997年5月北京第1次印刷
印数：1—5000册　定价：48.00元

译者的话

当今在计算机程序设计语言中最受大家欢迎的应是C和C++。C语言作为过程性语言，以其简洁、高效、灵活、可移植性好等诸多优点闻名于世，至今仍常用不衰。而C++语言作为面向对象程序设计语言，既具有面向对象语言的全部功能，又实现与C语言的全部兼容；既可用于面向对象类的设计、开发，又可用于应用程序的编制，因而C++语言在新一代程序设计语言中"后来居上"，迅速得到广泛的重视和应用，在UNIX、Windows 95、OS/2等平台上推广势头更为迅猛。

本书是Stephen Blaha.所著《C++ for Professional Programming with PC and UNIX Applications》一书的译本。统观全书，我们感到原著内容全面、系统，其中包括很多最新C++特性，如模板、异常、布尔量、RTTI等，首次详细介绍STL、IPC等新内容；注重实用性，对2000多名专业人员的反馈意见进行归纳，在书中予以解答，从而可满足广大程序设计人员的实际需要；在讲述安排上由浅入深，以通俗易懂的语言介绍C++的思想，以图表示例方式解释复杂的设计与实现。我们认为，原书是我们所见到的有关C++语言方面的书籍中较突出的一本。

全书共分17章和两个附录。其中孟庆昌译第1，4，10～13章，刘振英译第2、3章和附录A，牛欣源译第5～9章，黄中原译第14～17章，全书主要由孟庆昌译校。参加本书译校工作的同志还有：孟平、孙世忠、张卫丽、孟欣。

由于我们水平有限，对C++理解尚不够深透，加上时间紧张，书中难免有不少缺点或错误，恳请广大读者批评、指正，在此表示感谢。

译校者

于北京信息工程学院

1996.4

原　序

有关C++的论著一般分为两大类:C++语言的理论性论述和集中C++某些特点的介绍性书籍——这类书往往不提供语言的全部内容。

本书的独创之处在于:它集中了约2000名程序员的编程经验,这些程序员在近几年内均受到一周C++语言高级培训。这反映了来自商业、政府部门和教育领域的广大程序员对C++的关注和探讨。这些程序员具有各种各样的编程背景——多数人是用C语言,而其余人是用Pascal、Smalltalk、Ada、COBOL、FORTRAN以及其他语言。综合这近2000名学生的反馈信息,我认为非常有必要编写这本书,以满足职业程序员的工作需要。

本书目的在于带领读者快速进入C++面向对象程序设计。研讨班中数千名学生提出的众多问题在本书中都进行了讨论。所以,在老师上课期间你想提出的问题很可能会在本书中找到答案。而其他C++论著不是建立在这种信息反馈的基础上的。多数教科书在出版之前并未在教学中用过,其余的教科书至多由少数学生使用过。因此,对学生实际提出来的许多问题在那些书中并未做解答。

这2000名程序员对于现有的C++书籍感到不满意:理论性书籍过于抽象,难于理解;介绍性书籍又找不到想要的答案,涉及面太宽,常常遗漏一些重要的问题。

本书的目的是提供一个从头到尾完整的方法,介绍使用面向对象程序设计方法的C++语言的特点。更重要的是将您快速带入职业程序员的境界中。本书不是对语言繁琐的解释,而是给出学习中学生常提问题的解答。本书还提供了一些实际的编程和设计建议,对正在工作的程序员可能遇到的问题提供可选择的解答方案。

本书利用各种各样的程序和图表,对C++语言的规则作了通俗、简明的说明。书中的程序详细描述了处理复杂事件的步骤,揭示了程序运行的内部情况。C++程序使人看上去误以为简单且回避了复杂的事件序列。

本书包括了许多仔细设计的练习,通常每个练习解释清楚该章中介绍的一个特性,以方便读者能马上证实自己的理解是否正确。一般而言,每个练习是独立存在的,不依赖于以前章节的练习。

本书对C++的特性作了详细、深入、全面的介绍,使读者能准确快速地了解其要点。它是职业程序员辛勤劳作的结晶。

非常感谢Mark Brooks、Nick Filer、Dave Johnson、Don Kostuch、Garfield Lewis、Tom Moran、David A. Sykes和Richard Voss提出了许多建设性意见,感谢广大学生,他们在学习C++时提出的问题指引着本书的编写。最后,感谢Jim DeWolf先生和他的工作人员(尤其是Kathleen Raftery、Chris Grisonich和JoAnne Campbell)对整个出版过程的大力支持。

目　录

概　述

C++是基于C语言的高级程序设计语言，它成功地实现了面向对象程序设计思想，从1990年以来成为首选的最受欢迎的程序设计语言。C++提供了从C语言转换到更高级程序设计的理想途径。

大家争先恐后地利用C++进行开发，使它逐渐普及开来。C++引起越来越多人们的兴趣，在传播媒体和程序设计团体中，C++被广泛接受，吸引了人们的注意力。此外，多数C++编译程序也能编译C程序。单独购买C++编译程序，利用它们进行C语言程序设计，将来也可用它进行C++程序设计，这是一举两得的事情。总而言之，购买C++编译程序可以少花钱、多办事。

想学习C++的程序员往往会听到各式各样的议论："学习C++，你不必知道C。""C++比C容易。""C++比C难得多。"每种议论都是部分对、部分错。现实往往更复杂。事实上存在两种C++程序设计：一种C++程序设计是由应用程序员完成的，另一种C++程序设计是由C++类的设计开发人员实现的。对每种程序设计所需的背景差异相当大。本书是为这两种程序员编写的。

C++应用程序员编写C++应用程序代码时，利用的C++类库是由别人开发的，通常在C++应用程序设计环境下工作只需要很少的C和C++背景。这种形式的C++程序设计比C程序设计更容易，并且可能不需要很多C语言背景。因而会有人说："学习C++，不需要知道很多C。"以及"C++比C更容易。"

另一方面，C++类的设计开发人员编写高技术C++代码时通常要采用C和C++全部特性集合的优点。这类人员创建C++应用程序设计环境，供C++应用程序员使用。C++设计开发人员必须对C和C++知识有透彻的了解。由于C++的特性集合更大，在设计和编码时出现的选项很多，C++设计开发人员建立代码时必然更加困难。这样，设计开发人员往往会发现C++比C更难应付。

有些程序员既想当设计开发人员，又想当应用程序员，这就必须全面了解C++的特性。

如果用C++进行面向对象程序设计比C程序设计更困难、更复杂，那C++还有什么优点呢？C++的优点以两种方式出现：C++可以为应用程序设计建立简单、高效的程序设计环境，以及纯技术优点——众多新的方便性特性。

面向对象C++程序设计使我们能构造很好的程序设计环境（用几万行代码），在此环境中我们可以创建几百万行代码的应用程序，它们具有良好的可读性、可维护性以及易于修改性。对面向对象C++的报答是能够创建应用开发环境，它支持快速、高效的应用程序开发。

不同时期C++应用开发环境为其自身的简洁性和高效性做了多次反复改进。应用开发通常占程序设计工作的绝大部分，在C++开发环境中它有高得多的成本效益。所产

生的应用程序代码也更易于维护，便于该应用程序升级到下一个版本。

本书是为C++设计开发人员和应用程序员编写的。设计开发人员从中会找到对C++特性的全面介绍。本书还附有大量说明示例。本教程着眼于提供实用的且技术知识完整的C++特性介绍，在抽象语句之后总是有简单示例予以解释。

应用程序员会从中找到他们进行重要的应用开发所需的全部C++知识。他们可以选择阅读章节，忽略某些与他们当前需要并不直接相关的题目。至少他们要知道忽略的一些章节，以后如果需要可以再学习它们。

没有通往C++面向对象程序设计的捷径。在C++方面取得成功的人显然取决于他们的C语言背景。

C语言程序设计的经验非常有益，这有两个原因：其一，C程序设计开发了程序员进行抽象程序设计的能力，并为他(或她)奠定了C++的更为抽象的概念和技术基础。归根到底，C++是C语言的扩展，并且分享它的很多技术风格。其二，C程序设计特性在C++中得到频繁使用。一个人使用C的经验越丰富，编写C++代码也就越容易。

Pascal、Ada和其他现代语言也为学习C++提供了良好的背景，因为它们有助于开发程序员的抽象编码能力，并且它们在很多地方与C语言类似。但另一方面，FORTRAN或COBOL程序设计经验一般并没有太大帮助，因为这些语言缺少对程序设计的抽象化和对不同风格程序设计的实用性。

附录A给出了简短、但相当完整的C语言主要特性介绍，可供没有C语言背景的程序员参考。

本书的一个指导原则是C++的可移植性：编写的代码应便于移植到其他硬件和软件环境上。可喜的是，C++较易于移植。然而，在C++编译程序之间存在明显的差别。例如，一些编译程序在各处保留的"temp"对象比其他编译程序更长久；一些编译程序在细节上有差别，例如在"throw"语句之后需要有返回语句，尽管这个return从来也不会得到执行；还有些编译程序在与ANSI C++标准一致性方面有差异，在各种环境中现有程序库支持方面也有差别。

我们的目标是完全可移植C++代码(对类库的函数调用中模数有差别)。这种可移植性是我们努力争取的。几年以前我开办了一个小型软件公司。我们希望把我们的MS—DOS的C代码移到相对说来价格更贵的UNIX机器上，但是我们无力去购买。我们发现，附近有一个计算机商店在演示厅中有这种机器的演示版，决定试一试某些"street(街道)"程序设计。于是我来到这个商店，询问我是否可以在他们的UNIX机器上试试程序设计。在得到同意之后，我用了大约半个小时(需要做一点儿修改)移植了上述代码(相当大的数量)。我带着几张软盘走出大门，软盘上有我们的新产品(作为"答谢"，我在这个商店买了一盒软盘)。这个故事说明可移植代码到底是什么，可移植代码应该是少量修改并且快速编译。这正是我们探讨C++程序设计的目的。

本书介绍了可移植C++编码技术。第14章、第16章和第17章介绍的性能，在当前某些C++程序设计环境中是不可用的。第14章介绍新采用的C++特性，第16章介绍在面向对象的外衣下包装InterProcess Communication(进程相互通信，IPC)的性能。在大多数当代PC和UNIX环境中都支持IPC，如Win32，Windows NT，Windows 95，OS/2，AIX，Solaris，SUNOS，HP-UX以及其他UNIX操作系统环境。第17章介绍Standard Template li-

brary(标准模板库,STL),它是标准C++库的一部分。本章内容在C++环境之间是可移植的,这些环境中包括标准C++库或标准C++库的STL部分。

本书的C++代码可利用当前主要的C++编译程序毫不费力地进行编译(可能除第14章和第17章介绍的C++新特性之外)。

本书的另一个目的是我所称的"原子编码(atomic coding)"。原子编码采用最简单、最直截了当的方式,用逐步逼近的方法编写代码,从而避免使用一次做很多事情(称作副作用)的复杂语句。这种代码易于编写、阅读和维护。采用原子编码风格可使代码编写工作快速和高效。例如,我的小软件公司在80年代中期就采用这种方法为MS—DOS编写C代码。

我们快速而高效地生产了Desktop Publishing(桌面出版系统)产品,可能包括PC机上的第一个桌面出版程序——它使用低成本激光打印机、第一个字形编辑器、第一个PC机上廉价的扫描程序。我们的竞争对手和某些资本投机厂商猜想我们有一个程序员小组,他们夜以继日地工作。事实上,我一个人既是晚上的程序设计小组,又是白天的客户支持小组和市场小组。这是原子编码产生的巨大成果!

原子编码是基于简单、逐步编码的风格,它尽量避免不必要的装饰。每条代码语句带领我们朝目标前进一步,每一步应尽可能地小——原子的,而目标是实现最大的可读性和可维护性。本书告诉程序员如何创建原子C++代码——它易于编写,可读性好,便于维护。

把原子编码与很多公司采用的一些现代编码比较一下。很多C和C++程序员从编写密集的难以阅读的代码中得到某些精神上的满足。像这种代码编写起来费很多时间,并且比简单的、直截了当的代码需要更多的维护。

除了移植性和原子编码之外,我们选择方法的另一个指导原则是最小化。"不打入无益的代码。"例如,我们不打入代码

```
struct employee e1;
```

而应打入:

```
employee e1;
```

因为在C++的这个上下文中关键字struct是可选的,而且使用它,我们从中得不到任何好处。通过省略可选表达式,C++代码往往可以写得更简单。我们的介绍将持续不断地努力帮助你编写出最简洁易读的代码。

当你阅读本书时会注意到:我们使用了带有类的简单示例,如Animal(动物),Giraffe(长颈鹿),Boat(小船),Car(小轿车)或者Ghost(鬼怪)。一些读者觉得,所用示例应当取自现实世界的类和类库,但可惜的是,现实世界示例通常是不可移植的,对特定的环境它是专有的。现实世界示例还要花费大量的背景开发,也没有必要因为它们而加长本书的篇幅,使本书的重点弄得模糊不清。所以我们把现实世界示例限定在第15章和第16章。

第15章包含具体应用程序(如链表)的讨论。第16章讨论InterProcess Communications(IPCs),如接插件(socket)、文件加锁、信号灯、消息队列等等。这些操作系统资源可用于各种PC和UNIX环境。支持IPCs的PC环境包括Win32,Windows NT,Windows 95和OS/2。PC机的IPC包括有名管道和WinSock接插件(从UNIX的接插件衍变来的)。根据SVID标准(UNIX系统V接口定义),在UNIX操作系统环境中的IPCs也被标准化了。

本书主要目的之一是在有关设计开发人员和应用程序员的程序设计实践方面开发一种完美的观点。设计开发人员应创建应用开发环境，它有利于实现高效、无错的应用程序开发。

Edward Deming 是行业质量控制部门的负责人，几年之前他提出一个见解，他认为在产品制造中出现质量问题的根本原因是生产环境助长了错误。如果我们把这个见解用于应用软件开发，我们便会看到，C++设计开发人员应试图建立应用开发环境，它帮助应用程序员避免错误，并且促进高效率代码的生产。

设计开发人员应为应用程序开发者建立失效安全编码环境，它有简单的程序设计界面，可帮助程序员避免错误。这种方法的一个示例是第 16 章开发的接插件界面。传统的 C 代码应打开一个接插件，其代码如下：

```
FD_ZERO(&idset);
FD_SET(soc_id,&idset);
FD_SET(accept_id,&idset);
if((phost=gethostbyname(hostname)) == -1)
        perror("Can't get host information.");

if((soc_id=socket(AF_INET,SOCK_STREAM,0)) == -1)
        perror("Can't open socket.");

bzero((char *) &sockname,sizeof(sockname));
sockname.sin_family = AF_INET;
sockname.sin_port = htons(portnum);
bcopy(phost->h_addr,(char *) &sockname.sin_addr,phost->h_length);

if(bind(soc_id,(struct sockaddr *) &sockname,sizeof(sockname)))
                                perror("Can't bind.");
listen(soc_id,5);
size_acceptname = sizeof(acceptname);
accept_id=accept(soc_id,(struct sockaddr *)&acceptname,&size_acceptname);
...
```

上述代码很复杂，并且有很多出错以及排印有误的机会。比较一下这个代码与取自第 16 章的下述应用程序开发代码，它与上述代码片断实现的任务相同。下述代码片断使用一个类设计，提供对接插件的简单应用程序设计接口。

```
Soc soc1("mymachine",PORT_NUM);
char buf[128];

if(soc1.soc_bind() == -1)
        perror("Can't bind.");
```

```
soc1.soc_listen();

if(soc1.soc_accept() == -1)
        perror("Accept error.");
```

可清楚地看出，C++应用程序代码片断更易于编写、阅读和维护。其简洁性减少了程序设计出错的机会。第1个代码片断需要很多细节情况，带有很多出错的机会——正是Deming理论起作用的示例。在第2个代码片断中有更高的纠正代码可能性，因为它很简单。

如果设计开发人员遵循Deming理论，利用C++面向对象程序设计可生产出更简单、质量更高的应用程序代码。本书将告诉你如何创建C++应用程序环境以及如何编码可获得高效率应用程序环境，用以减少人为出错机会。

最后，我们用有关C++作为过程性语言的一句话来结束。多数C++设计时考虑到面向对象程序设计。C++程序设计界的领导者继续强调C++程序设计的面向对象的方面，并且严厉批评那些试用或者甚至想用C++作过程程序设计的人们。

从实践观点出发，我们为什么不能把C++看作有一些漂亮特性的C的增强版而用它创建过程性程序呢？有很多无足轻重的程序设计项目，在那里用全套的面向对象程序设计工作并不合算。在那些情况下C++提供了很多令人感兴趣的方便使用的特性，如在函数调用中参数默认，运算符重载以及带引用参数的函数，这使它引起过程性程序设计的兴趣。C++很多新特性，像模板、异常、布尔量和类型强制改进等，确实可主要看作是技术上的过程性改进。

本书强调C++面向对象的程序设计，但读者应牢记，在较小规模的程序设计项目中，把C++用作过程性程序设计的C的增强版是有益的。

本书阅读指南

根据你自己的知识背景，有多种阅读本书的方式，以便获取最大收益，并且使你的时间得到最有效的利用。本书预计给四种读者使用，建议他们采用下述方式阅读本书：

第 1 种读者：对 C 语言和面向对象概念了解得很少。

阅读顺序：第 1 章，附录 A，第 2 章，依次是其余各章。

第 2 种读者：对 C 语言有很好的了解，但对面向对象概念知之颇少。

阅读顺序：第 1 章，第 2 章，依次是其余各章。

第 3 种读者：对 C 语言了解得很少，但对面向对象概念有良好了解。

阅读顺序：附录 A，第 2 章，依次是其余各章。

第 4 种读者：对 C 语言和面向对象概念都有良好的了解。

阅读顺序：第 2 章，依次是其余各章。

特别应注意：新术语、关键字、代码表达式以及特别重要的规则（通常是一些违背常规的或者易造成误会的），往往用黑体字表示。当查找一项时，黑体字也有助于更快地找到一个术语或者关键字。

第1章　面向对象设计和编程

本章介绍面向对象设计和面向对象程序设计的思想。

1.1　C程序设计中的问题

在过去20多年间C程序设计发生了显著的变化。在1970年用C进行程序设计的项目通常是小的、面向字符的(短小或非图形的),多数项目的工作量从几个人月到几个人年,典型的大型程序才有2万行代码。

如今用C进行程序设计的项目规模变得越来越大,它们要花费很多人年的工作量,往往有几百万行代码。这样,C程序设计的不足在小型程序设计中尚能被容忍,但现在却成为程序员的主要问题。

对程序员说来,大型C程序设计的部分问题来自高层需求的精确度和准确度。为了书写正确的代码,程序员必须确切地记住各数据变量的细节。通常程序的数据表示并没有良好的结构化,往往按随意方式加以构造,可以直接地或者通过指针间接地由程序的若干部分对数据进行访问。

对大型程序进行维护和性能改进也是困难的,因为进行系统维护和性能改进需要详细了解数据的表示。数据表示的任何改变都影响到程序的其他部分、导致一连串的改动,这就可能产生新的故障。由于大型C程序在程序逻辑上有很多分支,对它进行过程分析也很困难。

现代程序的规模变得非常大,这就需要大型的程序结构。像以往那种一次处理一个单独变量、编写无数行代码来操纵众多单独变量的方式,不再是一种可取的方法。与此相反,我们希望所编写的代码能处理变量的集合,这个目标就引导我们走向面向对象的程序设计。

正如概述中提到的Deming理论所指出的那样,当环境助长出错时,就会导致质量问题。大量过程性C程序设计提供的环境增加了出错机会,并且助长产生错误,这样就会出现劣质软件。而面向对象程序设计(OOP)所要创建的程序设计环境是减少以至消除产生错误的机会。OOP促进大型优质软件的开发,其程序设计的规模比使用底层单个变量时的规模更大。

1.2　解决方案:面向对象程序设计

解决过程性C程序设计中存在问题的一种方案是强化C语言,使它支持采用变量(对象)集合进行程序设计。代码应完成"大图(The Big Picture)",不要处理单个变量细节这