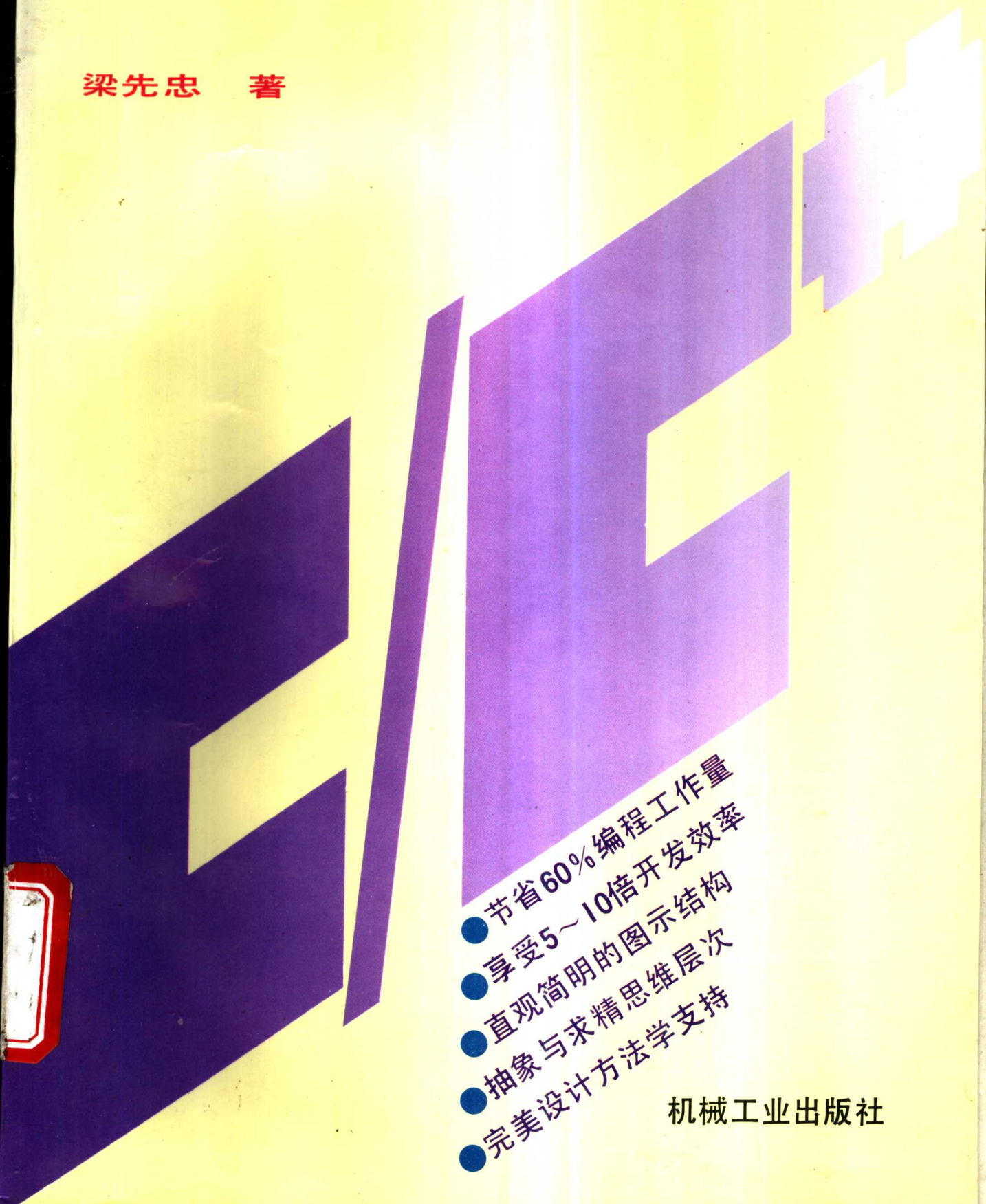
熟练编程一点通

梁先忠　著



机械工业出版社

熟练编程一点通

梁先忠 著

编程者的梦想——

超越繁琐编程细节，享受轻松工作环境

节省 60%编程工作量
享受 5～10 倍开发效率
直观简明的图示结构
抽象与求精思维层次
完美设计方法学支持
智能化的“傻瓜”软件

机械工业出版社

（京）新登字 054 号

C/C++语言由于其本身的成功在国内外得到了广泛使用。它以其高效灵活，特别是面向对象方法学的完美体现，将成为 90 年代最受欢迎的程序设计语言之一。武汉远望技术研究所的梁先忠，集近 10 年来对 Ada 语言和程序设计方法学潜心研究的经验，研制出了支持 C/C++程序设计的神图软件开发环境，强调程序设计的方法和软件工程原理，使人们不再陷入程序语言的繁琐文法细节中。本书结合神图描述方法的最新成就，自然地描述了 C/C++的语言特性；精炼地给出了良好设计风格的实例；深入地介绍了充分利用 C/C++语言特性的，按“积木”式构筑程序的面向对象设计的方法学，可以协助人们熟练掌握 C/C++这一强有力的语言工具。

本书另有神图 C/C++开发环境软件，含系统用户手册，经典程序。读者可与本社联系。

图书在版编目（CIP）数据

熟练编程一点通/梁先忠著. —北京：机械工业出版社，1995

ISBN 7-111-04542-4

熟… Ⅱ. 梁… Ⅲ. 程序设计-基本知识 Ⅳ. TP311

中国版本图书馆 CIP 数据核字（95）第 03425 号

出版人：马九荣（北京市百万庄南街 1 号　邮政编码 100037）
责任编辑：王福俭　版式设计：王　颖　责任校对：张晓蓉
封面设计：姚　毅　责任印制：卢子祥

北京交通印务实业公司・新华书店北京发行所发行

1995 年 8 月第一版第 1 次印刷
787mm×1092mm 1/16・11.25 印张・262 千字
0 001— 3 000 册
定价：18.00 元

致　谢

感谢在本书手稿准备过程中帮助过我的人。特别是马晓岩和程国政，他们两位在成书过程中，提出了许多有益的建议。Diagsoft 实验室的同仁们付出了许多时间和我讨论 C/C++的技术问题，胡奇寒和徐琼芳小姐不仅录入了全稿，还在文字修辞方面做了大量工作。

神图 C/C++开发环境的研究使我受益非浅，本书中的许多技术关键与之紧密相关。

感谢中国科学院院士北京大学杨芙清教授的耳提面命，我的恩师，中国船舶工业总公司七〇九所王振宇教授曾给予的悉心指导，清华大学郑人杰教授的直接指点，电子部的领导和专家杨天行、陈冲和朱鹏举给予了大力的支持，还有许多专家教授也给予了热情的支持和指导，他们是应文烨、叶仲翘、陈曾涤和王广荣等。

还要感谢机械工业出版社，给予的支持和帮助。尤其要感谢我的妻子多年来在神图的研究过程中对我充分的理解、悉心的照顾和鼓励。

作者的话

C/C++是一种表达能力较强的通用程序设计语言，提供了类似汇编语言那样的系统资源操纵能力和程序执行效率，使低级的程序设计活动得以融入高级语言；更重要的是它提供了对90年代程序设计方法学的最新成就——面向对象方法的支持！

然而，C/C++是入门容易得"道"难的程序设计语言。一般程序员在一周内读完一本C/C++语言手册并能上机编程并非难事，但要真正掌握其精髓，充分发挥其潜能却相当困难。起源于60年代Bell实验室的C语言，囿于历史的局限，由于过分强调灵活性和简洁性，该语言在严格完整定义方面存在许多缺陷；C++基本上是C的一个超集，它主要增强了支持面向对象编程。客观上C/C++在描述方面存在某些先天不足，所以C/C++的程序设计，经验是很重要的。

神图C/C++环境是以Ada设计思想为基础的、支持C/C++程序设计的一种新型CASE工具。它以其独特的设计规范，使程序设计活动既依附于现行的C/C++语言，又不拘泥于其繁琐的文法格式。神图将程序设计方法学及软件工程原理与C/C++完美结合，在充分利用C/C++语言特性的同时，也弥补了它的缺陷，可以协助读者正确领会C/C++的精华，避免其自身的弱点。

- 本书不仅仅是又一本介绍C/C++的新书，它还能满足以下目的：
- 协助读者对C/C++性能的深入了解，掌握其精髓；
- 促进良好的C/C++程序设计风格，并给出实例；
- 类比程序设计语言的最新发展，分析利弊，使读者扬长避短；
- 结合神图的设计规范，强调程序设计方法学并弱化C/C++具体文法格式；
- 凸现面向对象设计方法，充分利用C/C++的功能，进而帮助人们组织和管理大型软件系统的开发。

若个人的编程经验在程序设计中占较大比重，只能算是一种效率低下的人工智力劳动。即便对于比较精通C/C++语言的人在进行大规模系统开发时，也可能会被一些突如其来的、难以解释的错误搞得束手无策，使他们如堕五里雾中。只有按照规范化、科学化的体系进行程序设计，才能适应大规模软件开发的要求。

一、问题的提出

在写作这本书的过程中，我毫不怀疑C/C++将会成为90年代最受欢迎的程序设计语言之一。但是，很多关于C/C++语言的教程仅从语法和语义的角度讲述程序设计语言的细节和特点，却善意地遗漏了C/C++存在的不足和忽略了程序设计与方法学相结合的要求。下面的阐述将有益于帮助读者正确认识C/C++的语言特性和熟练掌握这一强有力的语言工具。

1. 现行C/C++程序设计的不足

在程序设计过程中，程序开发往往被分解成多模块来进行。C/C++以头文件的形式"链"接多模块成为系统。实际上，在C/C++的程序设计中，即便各个模块分调时都无懈可击，但是对拼联起来的系统进行联调时，可能存在运行出错、死机等问题。因为现行C/C++

此为试读，需要完整PDF请访问：www.ertongbook.com

编译系统的“诊断”能力较弱，无法检查的错误主要有两类：其一，程序语言描述方面的隐患，它可能给程序造成大量的错误；其二，程序员在程序逻辑与算法方面的问题。这在各种程序设计语言中都客观存在，C/C++也不例外。现行C/C++程序设计的缺陷具体表现为：

(1) 无法保证程序的一致与合法性　程序是众多人员分工合作设计的结果，一致与合法性是软件系统有机构成的基本要求。一致性是指各模块使用一致的符号（没有任何不必要的差异）；合法性是指调用函数时，其调用的实参必须在个数和类型上与函数定义的形参严格匹配。

(2) 头文件的隐患　头文件是“链”接诸多模块构成完整软件系统的辅助描述。按现行观念，C/C++的程序设计没有它的参与是不可想象的，所以称之为构筑程序的“神链”。而组织不当的头文件可能造成许多隐患潜伏于软件系统中，使系统的有机构成在一致与合法性方面遭到破坏，它又是导致程序故障的“魔鬼”。

(3)“孤立”编译的缺陷　C/C++的编译系统在编译某个模块时，不直接参照其他与之相关联的模块，而是依赖头文件描述的函数原型对函数调用进行合法性检查。人为编写的头文件可能存在多层次的不一致，导致的错误被滞留到程序的运行和调试阶段。

(4) 程序调试无可奈何　若设计阶段头文件的隐患被滞留到程序的运行和调试阶段，它们与程序的逻辑算法错误混为一体，依靠人工“诊断”的被动方式会极大地增加调试难度。

> 电脑软件的奇才 Edsger W. Dijkstra 先生曾就程序调试说过：
> “调试只能发现程序有错，而不能证明程序无错”

2. Ada 对我们的启示

Ada 是美国国防部（DoD）通过国际招标制定的军用标准语言，旨在控制其软件成本的不断增长和克服软件开发出现的危机。除了美国，其他西方发达国家也采用 Ada 作为军用标准语言。我国从“七五”期间就开展了对 Ada 的研究，并拟采用 Ada 作为我国的军标。

(1) Ada 集程序设计语言之大成　Ada 是专为大规模、实时、复杂和长期生成目标而设计的，它汇集了软件工程及现代程序设计方法学中几乎全部有用的思想；同时使语言表达能力、应用范围、可读性、可理解性和严格完整的定义等，都以较高的水平求得解决。

(2) Ada 语言及其编译系统的特点

1) 程序包是基本的设计单位（模块）　程序包被划分成规格和体。规格着重描述该模块的抽象性质，为外部引用提供了完美而严谨的接口，而程序包体，受规格的约束和指导，具体地实现程序包的功能特性；

2) 规格说明是保证系统有机拼联的机制　外界模块对程序包定义的数据和子程序引用，受程序包规格的严格制约，自然符合程序的一致性与合法性要求；

3) 对模块实施关联的分别编译　当一个模块被编译时，将自动“参照”与之相关联的其他模块，始终对程序一致与合法性实施动态检查；

4) 程序调试集中表现在逻辑与算法错误上　Ada 语言及其编译系统的根本宗旨是及时发现错误——使错误尽早在编译过程中暴露出来（自动诊断），减轻程序调试的强度。

3. 神图的灵魂——无头文件的 C/C++ 程序设计

神图 C/C++ 环境是以 Ada 设计思想为基础的、支持 C/C++ 程序设计的一种新型 CASE 工具。在神图环境中进行 C/C++ 程序设计时，编程人员几乎不用考虑头文件的编写、组织，

以及系统头文件的包含，从而杜绝了由头文件导致的系统隐患多、软件开发效率低下等问题。神图的模块设计 IC 化，“即插即用”的模块重用性和有关联的模块分别编译等特性是大规模程序设计的关键。

神图 C/C++程序设计的优势表现为：

(1) 模块设计 IC（集成芯片）化　IC 化模块是根据硬件提出的新概念，主要体现了软件模块重用性和系统拼联的简便性。在设计中引用其他模块时，人们只需关注作为抽象性质的模块规格说明，而不必理会作为整体封装模块的细节，因此 IC 化模块具有“即插即用”的特性。

(2) 自动生成辅助头文件　当进行模块链接时，神图环境将自动生成辅助头文件供有关模块隐式包含，不仅避免了繁琐头文件的人为编写，同时自动生成的头文件自然符合程序的一致性要求。

(3) 智能包含系统头文件　神图环境能根据编程人员使用系统库函数的情况对相应的系统头文件自动包含。

(4) 对模块进行关联的分别编译　现行的 C/C++的独立编译方式在神图环境中变成了有关联的分别编译。譬如，一个模块的接口（规格说明）被修改，将导致引用该模块的其他模块自动重新编译，始终保证对程序的一致与合法性进行动态检查。

(5) 调试集中表现在逻辑算法错误上　由于不存在一致与合法性的隐患，在运行和调试阶段排除了由 C/C++语言描述造成的错误，程序的错误只集中在逻辑算法上。

(6) 大大提高开发效率　用神图开发 C/C++程序至少可以节省 60%的设计和编程工作量；可使设计、编程直至调试过程的开发效率提高 5～10 倍。

二、结构

程序设计不仅是语言文法格式的描述，也是方法的体现。软件工程的宗旨是提高软件生产率和保证软件质量的正确可靠，其三大支柱是语言、方法和软件工具。我们可以用代数式表达如下：

软件工程＝语言＋方法＋工具

本书分为 3 篇、12 章，其组织结构按软件工程原理，分别从第 1 篇的语言基础、第 2 篇的设计方法学和第 3 篇的软件工具支撑，逐步展开对 C/C++语言和程序设计的介绍：第 1 篇 C/C++导论　包括对 C/C++历史的考察，纵观 C/C++语言的特征，从程序设计方法学的角度分析 C/C++的基本概念并深入浅出地通过详细的实例加以论述；第 2 篇 用神图 C/C++进行程序设计　在本篇中，作者着眼于大型软件系统的组织管理方法，着重介绍使用神图的设计规范进行程序设计和 IC 化模块的重用性和系统拼联的简便性，结合神图的设计规范并通过实例，分别阐述：以程序抽象为基础的功能设计 (Functional Decomposition) 和以数据抽象为核心的面向对象设计 (Object-Oriented Decomposition)；第 3 篇 神图 C/C++开发环境概要地介绍了神图的系统功能和用户操作指南。

作者

1994 年 10 月

目　录

第2篇 用神图C/C++进行程序设计

第 3 篇 神图 C/C++开发环境

第1篇　C/C++语言导论

本书第1篇共分5章，主要介绍C/C++语言的基础知识。众所周知，作为一种程序设计语言，C/C++拥有自己的特性、文法形式和语义。C++除了主要增强支持面向对象设计的方法外，基本是符合ANSI标准C语言的超集。从语言文法的角度来看，C++在C语言的基础上增加的描述成分较少，更大的增强应该说是在程序设计方法和编程人员的思想观念上。

构成C语言的大多数基本概念，普遍适合于C++。然而与C++支持面向对象设计的方法相悖的某些C语言概念不是C++推荐的良好设计风格，基本上不作为本书讨论的重点。程序设计方法学与C/C++结合，包括以程序抽象为基础的功能设计和以数据抽象为核心的面向对象设计方法，将在本书的第2篇中详细讨论。

第1章　C/C++语言概述

C/C++语言由于其自身的成功和提供了类似汇编语言那样的系统资源操纵能力和程序执行效率，特别适合于系统软件和应用软件的开发，正受到人们广泛的重视与应用。更重要的是它提供了对面向对象设计方法的支持，将成为90年代最受欢迎的语言之一。

1.1　C/C++语言起源

C语言的最初设计和实现是由Dennis Ritchie于70年代在Bell实验室完成的。Smalltalk语言的出现带来了程序设计方法的飞跃，它提出的面向对象设计方法，其实质就是采用结构化编程的最佳思想，使程序员充分理解并管理更大型、更复杂的程序，这种方法改变了人们传统的设计观念。面向对象设计方法的最大特点是将数据和处理该数据的代码封装为一体，去映射问题求解空间的逻辑实体。

C++是C语言的一个扩展版，由Bjarne Stroustrup于80年代初也在Bell实验室发明出来。如果说C语言主要体现的是以程序抽象为基础的功能设计（Functional Decomposition）的话，那么作为C语言的扩展，C++则增加了以数据抽象为核心的面向对象设计（Object-Oriented Decomposition）方法。因此C/C++代表目前这两种主流程序设计方法的折衷。

当Bjarne Stroustrup在Bell实验室酝酿C++的过程中，他深知在增强对面向对象设计方法支持的同时，C++离不开C—这一为人们普遍接受与广泛应用的强有力的通用程序设计语言。最为重要的是应保留C语言的原始精华，其中包括高效率和灵活性、编程方便和适应性强等基本特性。Bjarne Stroustrup不愧为天才的“魔术师”，他的目标都实现了。如果说Bjarne Stroustrup在对C的扩展成为C++所作的增强是极小的，那么C++对面向对象设计方法的支持所取得的成就和效益是巨大的。用这位天才发明者的话说，“C++中的面向对象特性，使

此为试读，需要完整PDF请访问：www.ertongbook.com

程序构造得清晰，可扩充易维护，且不失有效性”。

虽然C++的初衷是为管理大型软件系统而设计的，但是它不仅只适应于大型软件的开发。C/C++的功能设计方法和面向对象特性，广泛适应各种系统软件和应用软件的开发。譬如，编辑程序、数据库管理系统、个人文件系统和通信程序等软件项目。此外，由于C++植根于C语言的基本特性，更高性能的软件都是用C++开发的。

1.2 C/C++语言是程序员的语言

C/C++语言是程序员的语言，不像专为非计算机专业的程序员而设计的BASIC语言——被用来编制解决比较简单问题的程序，也不像汇编语言那样使用起来完全无结构、杂乱无章，给程序员带来灾难，它的最大特点是将低级的程序设计活动巧妙地融入高级语言。用C/C++编程，程序员可以获得类似于汇编语言那样的系统资源操纵能力和程序执行效率，同时又具有像PASCAL和FORTRAN那样的高级程序结构。处理能力强与程序结构的清晰可读性，正是专业程序员所喜爱的独特功能要求。

C/C++语言之所以深受程序员喜爱，它既是结构化的，又是高低级功能都具备的程序设计语言。所谓结构化语言提供了大量的程序设计能力，直接支持若干简明程序结构。譬如循环结构：while，do-while和for，以及分支结构：if，switch等。在程序设计中能够避免使用GO-TO语句；而不像BASIC和FORTRAN那样将GOTO作为常用的、难以避免的程序控制语句。C/C++广泛为程序员所接受的另一个原因是，它不仅可以代替汇编语言，而且便于开发大型软件系统。表1-1比较了C/C++同其他语言的关系：

表1-1 C/C++与其它语言

C/C++	良好结构与 方法学支持	非结构化的 高级语言	低级的 汇编语言
C++ ↑ C ↓	Ada Modula-2 Pascal ALGOL	COBOL FORTRAN BASIC Macro- Assembler	Assembler

C语言最初被用于系统程序设计，像操作系统、编译程序、编辑程序、汇编程序、翻译程序以及数据库管理程序等方面。随着C语言的普及和功能的增强（演变成C++），它更广泛地被用于其它的应用程序的开发。

1.3 C/C++程序风格

从前面的讨论可以看出，C/C++是广泛为人们所接受的程序设计语言。虽然用软件工程

的观点来看，C/C++存在一些不足，但它是面向专业程序员的。以下几个方面进一步阐述了语言决定程序风格的特性。

1. C/C++语言表达能力强　C/C++是结构化程序设计语言并支持面向对象方法，通用性强，不局限于某种机器。它不仅可以直接处理字符、数字和地址，完成通常由硬件实现的算术、逻辑运算，有效到足以取代汇编语言来设计各种系统软件和应用软件；而且还提供了抽象与信息隐藏和设计方法机制，使大型程序开发能力增强，按“积木”式构造程序系统。其中最有特色的是，运行多态性和继承功能，它们能重用已有设计和最大限度地减少重复劳动。

2. 表达简洁、使用灵活　在表达方法上力求简明，使程序紧凑，如一对｛，｝花括号代替一般的 begin、end；又如累计运算 a ＋＝ 1，代替一般的 a ＝ a ＋ 1，对运算赋值的形式进行了精简。这些也可以说是不足，因为它以牺牲程序的可读性作为简洁的代价，所以一般认为，读 C 的程序比较难。下面比较一段程序可以体会其简洁性：

```
// -- C ------          :      ( * ---- PASCAL --- * )
if (a>0) {              :      IF (a > 0) THEN
    do _ sth1;          :        BEGIN
    do _ sth2;          :          do _ sth1; do _ sth2
}                       :        END
else                    :      ELSE
    do _ nothing;       :        do _ nothing
```

对于 PASCAL 程序，可以看成是一段比较清晰的英语文字，而 C 的程序则不那么明显，倒像是一堆符号。C 语言的这种牺牲可读性换取简洁性，对非英语作为母语的中国人影响不大。或许可以有这样的结论：中国软件发展的后劲较强正是由于有像 C 语言这样简洁的语言，对英文的要求不高，C 语言的可读性差是能被国人所容忍的。

3. 具有很强的类型构造和流程控制能力　C/C++除了提供的预定义基本类型，如 int（整型）、float（浮点型）等，还能按抽象层次产生各种构造类型，如数组、指针、结构（struct）和类（class）。这些丰富的构造类型使程序设计更好地吻合问题的求解空间，增强了程序的可理解性和可维护性。下面的构造类型直观地反映了人事档案的基本信息结构：

```
typedef enum {   // 性别分 MALE-男性和 FEMALE-女性
  MALE, FEMALE
} SEX;
typedef struct {
  char p _ name [10];    // 姓名
  SEX p _ sex;           // 性别
  int age;               // 年龄
} PERSON;                // 构造的新类型，刻划人事档案数据
```

其中 //... 是C++的注释标记，类似于 / * ... * /

另外，它所提供的各种控制语句，如描述分支流程的 if，switch 和描述循环流程的 while，do-while，for，以及描述定向转移的 break，continue，exit 等功能很强，足以描述结构良好的程序；进而 C++中的类以及 public，private，protected 等访问限定符，则有助于信息隐藏的模块化和结构化程序设计。

4. 语言生成的代码质量高　高级语言能否用来描述系统软件和实时的应用软件，除要求语言的表达能力之外，更重要的因素就是语言生成的代码质量。实验表明，针对同一问题，用C/C++编程其生成代码的效率比用汇编语言编写的代码仅低10%～20%。由于用高级语言进行程序设计工作量小，易于调试、修改和移植，所以C/C++不失为理想的软件开发工具。

5. 可移植性强　可移植性是指程序从一个环境不加或稍加修改就可搬到另一个完全不同的环境上运行。汇编语言因依赖计算机硬件不具备移植性，其他高级语言及其编译也是不易移植的。C语言的编译程序便于移植，也就使得在某个环境用C/C++开发的程序可以很方便地移到另一个环境上去。可移植性良好是C/C++最为人们称道的特点，正如1983年度颁发计算机最高奖——图灵奖时所评论的那样，“对UNIX操作系统可移植性的关键贡献是Ritchie开发的C程序设计语言”。

6. C++预编译简单有效　严格说来，C++没有自己的编译程序，由于C++仅在支持面向对象设计方法上面作了一些增强，所以基本上是C语言的一个超集。C++通过预处理程序将新增强的成分“编译”成为C语言的代码，然后通过调用C编译程序生成可执行代码，这种方式全盘保留了C语言的简洁灵活性，不过也继承了C语言的某些缺陷。图1-1说明了C/C++的编译方式。

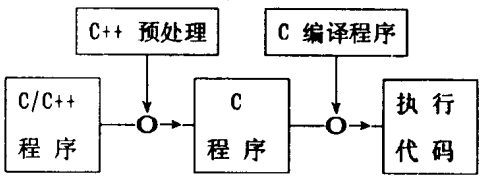


图1-1　C/C++ 程序的编译

1.4　程序库、链接与程序重用

从技术上讲，纯粹由程序员自己编写的语句构成C/C++的程序是可能的。然而C/C++之所以便于移植是它没有提供执行输入/输出操作的任何方法。因此大多数C/C++程序都含有对C语言种种标准库的调用。

所有C/C++编译系统都是与其标准库一起提供的。C语言的函数库和C++的类库十分丰富，它们含有完成各种常用任务的函数和刻划常用数据抽象的类(class)。C/C++编译程序的实现者已经编写了常用的通用函数库和类库，统称为C/C++的标准库。在程序设计中，当我们调用或使用别人编写的函数和类时，编译系统程序“记忆”它的名字，随后，“链接程序”把我们编写的程序同标准库中找到的目标代码结合起来，这个过程称为“链接”。

编写程序使用标准库的方式给我们提供了一种程序构件化管理和简便组合的途径。然而对于不同的系统软件和应用软件的开发，C/C++语言所提供的标准库毕竟是常用的，它太基础了。编程人员往往构筑自己的专用库（包括函数库和类库），针对自己专门应用的程序开发，在使用基础的标准库的同时，还使用新增加的附加库。这种使用已有程序构件组合程序的方式称为程序拼联，而可被多次使用的程序段称为可重用的程序（reusable）。

为便于按程序构件进行程序的组合，须采用良好设计风格来编写程序段（称为程序模块），使程序具有重用性，编写的每一个程序模块都作为程序构件添加进附加库中。按神图的观点，C/C++的程序开发是不断扩充附加库，并用附加库和标准库进行程序拼装的过程。图

1-2 说明了 C/C++的开发支撑环境。

在神图 C/C++软件开发环境中，程序设计是以模块化设计为基础的。这种模块强调的是重用性并提供了附加库的管理功能，也就是说，神图设计的每一个模块总是自然地进入专用的附加库，因而，程序拼联不像一般的 C/C++程序那样通过头文件的组织方式加以管理，而是按照模块的拓朴关系加以组合，这可以避免由于头文件组织不当给程序系统带来的隐患。随着本书的进程我们将逐步对此加以介绍。

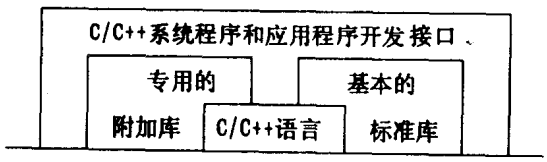


图 1-2 C/C++的开发环境

1.5 程序的编译

C/C++是高级程序设计语言，用它编写的程序，计算机是不能直接运行的，需通过“编译程序”进行代码生成方可执行。编译程序首先将整个源程序转换为可直接在计算机执行的目标码，然后通过链接程序组合该程序对其他构件库的引用，最后成为可直接运行的机器代码。被组合的构件库既可是 C/C++提供的标准库，也可以是设计人员自己的附加库。

1.5.1 PC 机的编译模式

众所周知，PC 机上的编译程序与内存模式相关，因此，编译时要求进行编译模式匹配。通常 C/C++的编译程序提供如下模式：

Tiny ——微模式　　Small ——小模式

Medium——中模式　　Compact——紧凑模式

Large ——大模式　　Huge ——巨模式

匹配选择主要依赖所开发程序的大小来定。

内存模式是一个预定义的规则集，编译器按照这些规则，将程序代码和数据转换成内存中的 64KB 段（内存块）。内存模式定义了编译器将代码和数据组织进段的方法，同时还规定了访问每段的代码和数据寻址方式（近程、远程和巨程）。近程地址是一个 16 位的偏移量（相对于段的头），它最多可以访问 64KB 的内存；远程地址和巨程地址是一个 32 位地址，它可以访问所有内存。

选择一种内存模式，即告诉编译器，它可以对程序的特性作某些假定，并产生出相应的机器代码。例如，当选择大内存模式时，编译器将允许程序带有 64KB 以上的数据；当选择中模式时，编译器只能产生 64KB 以下的数据，并产生数据项的近程地址。

1.5.2 模块划分和独立编译

用 C/C++编写的短小程序完全可以放在一个源文件中提交给编译程序，然而，随着程序长度的增加，编译时间也增加。因此，C/C++允许将程序分解成多模块，按分工合作的方式加以组织管理并提交给编译程序进行独立编译。一旦所有的模块编译完毕，就可以将它们连同“库”一块进行链接，形成一个完整的目标代码程序。单独编译各个模块的优点是，当程序的部分模块发生修改，却不用重新编译程序的所有模块。

单独编译存在一个致命的缺陷，就是编译程序孤立地看待每一个模块并对其进行单独处理。但是，程序模块的划分并非是完全孤立的，尽管某个模块的修改绝大部分是局部化的，对外界不产生任何影响，而模块的少数关键代码修改将影响引用该模块的其他模块。单独编译方式则只对修改负责，它不管使用的程序是否一致与合法，这种情况可能会给程序带来灾难。下面用两段简单程序来说明这一问题：

引用模块：	定义模块： ——→	修改形式：
main ( )	sum (int a, int b, int c)	sum (int a, int b, float *c)
{	{	{
int a, b, c, s;	int s1 = a+b+c;	int s1 = a+b + *c;
s=sum (a, b, c);	return s1;	return s1;
}	}	}

引用模块的 main 函数调用了由定义模块设计的 sum 函数，完成三个值的求和。由于 C/C++的编译程序对模块采用的是单独编译方式，针对前述例子将有两种情况：

1. 局部化修改，对调用不产生影响　若将定义模块中的 int s1 = a+b+c 改成 int s1 = b+c+a；这种修改因为没有涉及到函数的接口，所以是局部化的，它根本不会对调用模块产生影响，所以单独编译没有问题。

2. 接口修改，对调用产生影响　若将定义模块改成前述的修改形式，函数的接口就发生了变化。这时 sum 函数定义接口要求调用的参数必须是：前两个参数是整型 int，最后的参数是实型的指针（变量的地址）。这时若仍按前述使用模块的调用形式，就不满足函数定义的要求，调用是不合法的。因此，结果也不会正确。在复杂的程序中这种错误可能是致命的，而单独编译方式不能“诊断”这类错误。

1.5.3 神图关联的分别编译

神图环境的最大特点是，将现行 C/C++编译程序的独立编译方式变成有关联的分别编译。通常程序设计分为定义模块和引用模块，前者提供某些公共的设施供外界模块（如后者）引用。所谓关联分别编译是指：编译程序除了对各自模块实施分别编译外，若定义模块修改了其对外界的接口，如函数原型，环境将“引发”对引用模块的重新编译，并“参照”函数新修改的接口，检查调用的实参在个数和类型上是否与函数的定义一致。有错时，编译程序将报告函数的不合法调用。详细内容将在本书的第 2 篇中讨论。

1.6 从语言发展看 C/C++

为了建立计算机的解只有方法学是不充分的，还必须有合适的软件开发工具。程序设计语言是最基本的开发工具，计算机对问题的解以及在程序设计中采用的方法，无疑都得用语言这个开发工具来表达。然而从语言的发展轨迹来看，许多语言都存在某些方面的局限性。虽然 C++增强了对面向对象设计方法的支持，但其固有缺陷是客观存在的。

1.6.1 语言的发展轨迹

程序设计语言从 50 年代发展至今，已引进许多好的设计方法和语言机制。表 1-2 中，P.

Wegner 按照语言引入的性质，把最流行的若干语言划分为“代”，见表 1-2。

表 1-2 程序设计语言的发展轨迹

第一代语言（1954-1958）	
FORTRAN I	
ALGOL 58	
Flowmatic	
IPL V	
第二代语言（1959-1961）	
FORTRAN II	子例程，分别编译
ALGOL 60	块结构，数据类型
C	增强低级处理能力的 ALGOL 60
COBOL	数据描述，文件处理
LISP	表处理．指针
第三代语言（1962-1970）	
PL/1	FORTRAN + ALGOL + COBOL
ALGOL 68	ALGOL 60 的扩展了的后继
Pascal	ALGOL 60 的简单化的后继
SIMULA	分类，数据抽象
间断代（1970－1980）	
C++	改进的 C 语言，增强 SIMULA 分类功能（实属第三代语言）
间断代许许多多不同的语言，但都不确定	

1.6.2 程序的拓扑结构

最广泛使用的高级语言（包括 C 语言），如表 1-2 所示，都是在计算机科学史的早期，在人们了解大型软件系统的开发问题以前设计的。结果它们不能反映现代设计方法学的要求，从而不得不以增加预处理器（Pre-processor）和扩展的方法来修改它们，强迫它们满足现代设计方法，如 S-FORTRAN 和 C++。通过分析较早几代语言的拓扑结构（如图 1-3a 所示），可以看出其程序组织的不足。虽然 C++被划作为第三代语言，但其组织结构仍然没有脱离第一和第二代语言的拓扑结构的特点。它们都是将程序的数据和子程序分离管理的，由全局变量和一层子程序组成。由于使用了全局的数据结构，程序中某段的一个错误可能对系统的其它部分具有毁灭性的波及效应。

C/C++增加了数据的访问限定符（public，private 和 protected），虽避免了一些全局数据所带来的副效应，但 C/C++的组织结构是将数据作为头文件的形式，让一层函数通过（include）来共享，如图 1-

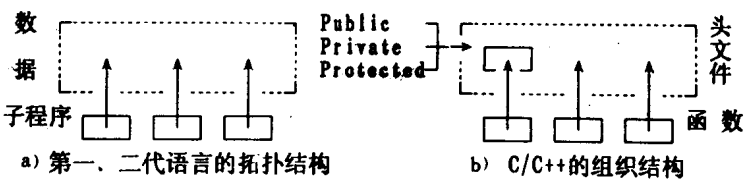


图 1-3 C/C++的拓扑和组织结构

此为试读，需要完整PDF请访问：www.ertongbook.com