

计算机知识普及系列丛书

# C语言程序设计 参考技术

叶亚明 编写  
燕卫华 审校

学苑出版社

计算机知识普及系列丛书

# C 语言程序设计参考技术

叶亚明 编写

燕卫华 审校

学苑出版社

(京)新登字 151 号

### 内 容 提 要

本书详细讨论了 C 语言及其库的各个方面。既包括旧的实际的 C 标准,又包括 ANSI C 标准。可以说本书包括了 C 语言的全部细节、技巧和应用。该书对从事 C 语言开发和应用的广大程序员和应用人员具有很好的参考价值。

欲购本书的用户,请直接与北京海淀 8721 信箱书刊部联系,电话:2562329,邮  
码:100080。

计算机知识普及系列丛书

### C 语言程序设计参考技术

---

编 写:叶亚明

审 校:燕卫华

责任编辑:陆卫民

出版发行:学苑出版社 邮政编码:100036

社 址:北京市海淀区万寿路西街 11 号

印 刷:施园印刷厂

开 本:787×1092 1/16

印 张:31.375 字数:743 千字

印 数:1~1000 册

版 次:1993 年 12 月北京第 1 版第 1 次

ISBN7-5077-0821-7/TP·19

本册定价:17.00 元

---

学苑版图书印、装错误可随时退换

## 前 言

写这本书的大多数时间里总是充满了挑战、乏味、兴奋与挫折的混合心情。但创作《C语言大全》第二版有另外一种难以捉摸的心情。我的任务是编写一本C语言的完全的参考手册，它不只是一本书，而且应是一本大全。我在写一本书时，力求彻底；但在本书中做得更彻底。正是“大全”这个词常常萦绕在我的脑海里。我知道，全世界的程序员将会买我的书，因为这是一本大全，最后，我认识到，你也一定认识到，迄今为止还没有一本有关C语言的书籍能够包括这种语言的全部细节、技巧和应用。C语言实在是太丰富、太强有力了。但是，我要说：我喜爱撰写《C语言大全》第二版。我相信，广大读者对我的劳动成果会感到满意的。

### 这本书适合你吗？

本书为经历各异的所有C语言程序员编写。阅读本书的读者起码能编写一些简单的C语言程序。如果你正在学习C语言，本书将会成任何一本C语言教材的理想配套资料，并对你的特定问题作出回答。

### 第二版中有什么新内容

我保留了第一版书中的大部分基本结构。有几章已作了变动，但是每章中的材料反映了原书的内容。另外，添加了许多新内容，包括扩充的图形包。

第一版与第二版的主要差异来自于新的ANSI C标准。写第一版时，ANSI C标仍处萌芽状态，正在不停地修改。另外，在那时候也没有一个编译器同正在变化的标准保持密切的一致。因此，原版多数内容仅反映了Kernighan和Ritchie定义的、最公共的实际的C标准。但是，现在标准已处稳定，大多数C编译器同标准保持一致，因此现是转换到ANSI C标准的时机了。（老的实际的标准同ANSI C标准的主要差异在本书中作了说明，所以你不必一定用ANSI C标准编译器来使用本书。）

为同ANSI C标准一致引起的主要修改是函数原型。它用在本书的各个角落，包括所有头文件。在旧的C版本中，不需要大多数的头文件就能正确运行。然而，为了体现函数原型的长处，必须包括所有头文件，因为头文件包含了标准库函数的原型。

ANSI C标准的采用引起另一改变是，用函数说明的最新形式。（旧形式也称经典形式，也在本书中作了详细解释，但是程序例子都用最新形式。）

### 书中有什么

本书详细讨论了C语言及其库的各个方面。它既包括旧的实际的C标准，又包括ANSI C标准。然而，主要强调的是ANSI C标准。本书分为五个部分：

- C语言
- C库
- 常用算法和应用
- C程序设计环境
- C++：C语言的最新进程。

第一部分详细讨论了关键字、预处理器命令和C语言的特征。

第二部分讨论标准C库。K&R和ANSI标准库都作了讨论。还包括一些普遍使用的DOS编程环境下函数；DOS调用和图形。对特定的DOS函数，用Microsoft C作例子说明。

第三部分包括一些重要的通用算法和应用；所有的C语言程序员应备的算法和应用。第二十二章，读者将会发现有关处理人工智能求解技术问题。

第四部分包括C语言编程环境，如同汇编语言接口、效率、移植和调试等问题。

第五部分介绍了C语言最新进展C++。C++是C语言的超集，其为了处理大规模项目的严格要求和帮助程序正确性验证。

# 目 录

## 第一部分 C 语 言

<b>第一章 C语言概述</b> .....	( 1 )
1.1 C语言的起源.....	( 1 )
1.2 C是一种中级程序设计语言.....	( 1 )
1.3 C是一种结构化程序语言.....	( 2 )
1.4 C是面向程序员的语言.....	( 3 )
1.5 编译器与解释器.....	( 4 )
1.6 C语言的形式.....	( 5 )
1.7 库和链接.....	( 6 )
1.8 分别编译.....	( 7 )
1.9 编译C程序.....	( 7 )
1.10 C语言存贮映象.....	( 7 )
1.11 术语.....	( 8 )
<b>第二章 C语言表达式</b> .....	( 8 )
2.1 五种基本数据类型.....	( 8 )
2.2 修饰基本类型.....	( 9 )
2.3 标识符名字.....	( 10 )
2.4 变量.....	( 10 )
2.5 访问类型修饰符.....	( 15 )
2.6 存贮类别说明符.....	( 16 )
2.7 变量初始化.....	( 20 )
2.8 常量.....	( 21 )
2.9 运算符.....	( 22 )
2.10 表达式.....	( 33 )
<b>第三章 程序控制语句</b> .....	( 35 )
3.1 真值和假值.....	( 36 )
3.2 选择语句.....	( 36 )
3.3 迭代语句.....	( 45 )
3.4 转移语句.....	( 52 )
3.5 表达式语句.....	( 55 )
3.6 块语句.....	( 56 )
<b>第四章 数组和字符串</b> .....	( 56 )
4.1 一维数组.....	( 56 )
4.2 产生指向数组的指针.....	( 57 )
4.3 一维数组传入函数.....	( 57 )

4.4	字符串 .....	( 58 )
4.5	二维数组 .....	( 60 )
4.6	多维数组 .....	( 63 )
4.7	下标指针 .....	( 64 )
4.8	数组初始化 .....	( 65 )
4.9	实例: 棋盘游戏 .....	( 67 )
<b>第五章</b>	<b>指针 .....</b>	<b>( 69 )</b>
5.1	指针是什么 .....	( 69 )
5.2	指针变量 .....	( 70 )
5.3	指针操作符 .....	( 70 )
5.4	指针表达式 .....	( 71 )
5.5	指针和数组 .....	( 74 )
5.6	多次间接 .....	( 75 )
5.7	指针初始化 .....	( 76 )
5.8	指向函数的指针 .....	( 77 )
√ 5.9	C语言动态分配函数 .....	( 79 )
5.10	指针带来的问题 .....	( 82 )
<b>第六章</b>	<b>函数 .....</b>	<b>( 84 )</b>
6.1	函数的一般形式 .....	( 84 )
6.2	函数的作用域规则 .....	( 85 )
6.3	函数自变量 .....	( 85 )
6.4	argc和argv——main( )的自变量 .....	( 90 )
6.5	返回语句 .....	( 92 )
6.6	返回非整型值的函数 .....	( 94 )
6.7	函数原型 .....	( 95 )
6.8	返回指针 .....	( 96 )
6.9	类型void的函数 .....	( 97 )
6.10	main( )返回什么 .....	( 98 )
6.11	递归 .....	( 98 )
6.21	说明变量长度和类型参数表 .....	( 99 )
6.13	经典与现代函数的参数说明 .....	( 99 )
6.14	实现问题 .....	( 100 )
6.15	库和文件 .....	( 101 )
<b>第七章</b>	<b>结构、联合、枚举和用户定义类型 .....</b>	<b>( 102 )</b>
7.1	结构 .....	( 102 )
7.2	结构数组 .....	( 104 )
7.3	把结构传给函数 .....	( 110 )
7.4	结构指针 .....	( 112 )
7.5	结构内部的数组和结构 .....	( 114 )

7.6	位域 .....	(114)
7.7	联合 .....	(116)
7.8	枚举 .....	(118)
7.9	用sizeof保证可移植性 .....	(119)
7.10	typedef 语句 .....	(120)
<b>第八章</b>	<b>控制台I/O</b> .....	(121)
8.1	读写字符 .....	(121)
8.2	读和写字符串 .....	(123)
8.3	格式化的控制台I/O .....	(124)
8.4	scanf ( ) .....	(129)
<b>第九章</b>	<b>文件I/O</b> .....	(133)
9.1	ANSI I/O与UNIX I/O .....	(133)
9.2	流与文件 .....	(134)
9.3	流 .....	(134)
9.4	文件 .....	(134)
9.5	文件系统基础 .....	(135)
9.6	fread ( )和fwrite ( ) .....	(143)
9.7	fseek ( )和随机访问I/O .....	(148)
9.8	fprintf ( )和fscanf ( ) .....	(149)
9.9	标准流 .....	(150)
9.10	类UNIX文件系统 .....	(151)
<b>第十章</b>	<b>C语言预处理器和注释</b> .....	(156)
10.1	C语言预处理器 .....	(156)
10.2	#define .....	(156)
10.3	#error .....	(158)
10.4	#include .....	(158)
10.5	条件编译指令 .....	(158)
10.6	#undef .....	(161)
10.7	#line .....	(161)
10.8	#pragma .....	(162)
10.9	#和##预处理器运算符 .....	(162)
10.10	预定义宏名字 .....	(162)
10.11	注释 .....	(163)

## 第二部分 C语言标准库

<b>第十一章</b>	<b>链接、库和头文件</b> .....	(164)
11.1	连接器 .....	(164)
11.2	C语言标准库 .....	(167)

11.3	头文件 .....	( 167 )
11.4	重新定义库函数 .....	( 169 )
<b>第十二章</b>	<b>I/O函数 .....</b>	<b>( 169 )</b>
<b>第十三章</b>	<b>字符串函数和字符函数 .....</b>	<b>( 205 )</b>
<b>第十四章</b>	<b>数学函数 .....</b>	<b>( 222 )</b>
<b>第十五章</b>	<b>时间、日期和其它有系统函数 .....</b>	<b>( 231 )</b>
<b>第十六章</b>	<b>动态分配 .....</b>	<b>( 262 )</b>
<b>第十七章</b>	<b>屏幕和图形函数 .....</b>	<b>( 274 )</b>
<b>第十八章</b>	<b>其它函数 .....</b>	<b>( 294 )</b>

### 第三部分 算法 及其 应用

<b>第十九章</b>	<b>排序和搜索 .....</b>	<b>( 311 )</b>
19.1	排序 .....	( 311 )
19.2	选择一个排序方法 .....	( 320 )
19.3	其它数据结构的排序 .....	( 320 )
19.4	随机访问磁盘文件的排序 .....	( 322 )
19.5	搜索 .....	( 324 )
<b>第二十章</b>	<b>队列、栈、链表和树 .....</b>	<b>( 326 )</b>
20.1	队列 .....	( 326 )
20.2	循环队列 .....	( 330 )
20.3	栈 .....	( 332 )
20.4	链表 .....	( 336 )
20.5	单链表 .....	( 336 )
20.6	双链表 .....	( 340 )
20.7	通讯录例子 .....	( 343 )
20.8	二叉树 .....	( 347 )
<b>第二十一章</b>	<b>稀疏数组 .....</b>	<b>( 353 )</b>
21.1	链表稀疏数组 .....	( 354 )
21.2	二叉树稀疏数组 .....	( 356 )
21.3	指针数组的稀疏数组 .....	( 358 )
21.4	散列 .....	( 360 )
21.5	选择一种方法 .....	( 364 )
<b>第二十二章</b>	<b>表达式的分析与求值 .....</b>	<b>( 365 )</b>
22.1	表达式 .....	( 365 )
22.2	分解一个表达式 .....	( 366 )
22.3	表达式分析 .....	( 368 )
22.4	一个简单的表达式分析器 .....	( 369 )
22.5	分析器加上变量处理 .....	( 373 )



22.6	递归下降分析器中的语法检查 .....	( 379 )
<b>第二十三章</b>	<b>人工智能问题求解 .....</b>	<b>( 380 )</b>
23.1	表示和术语 .....	( 380 )
23.2	组合爆炸 .....	( 381 )
23.3	搜索技术 .....	( 383 )
23.4	评价一个搜索方法 .....	( 383 )
23.5	图形表示 .....	( 384 )
23.6	深度优先搜索 .....	( 384 )
23.7	宽度优先搜索 .....	( 392 )
23.8	加启发性信息 .....	( 394 )
23.9	登山搜索 .....	( 394 )
23.10	最小代价搜索 .....	( 399 )
23.11	选择搜索技术 .....	( 400 )
23.12	寻找多个解 .....	( 400 )
23.13	寻找最优解 .....	( 406 )
23.14	回到寻找钥匙的问题 .....	( 410 )
<b>第二十四章</b>	<b>利用系统资源 .....</b>	<b>( 412 )</b>
24.1	8086类处理器 .....	( 413 )
24.2	8086中断和DOS .....	( 413 )
24.3	访问BIOS中的系统资源 .....	( 415 )
24.4	使用DOS访问系统功能 .....	( 423 )
24.5	利用系统资源的最后设想 .....	( 427 )
<b>第二十五章</b>	<b>图形 .....</b>	<b>( 427 )</b>
25.1	方式和调色板 .....	( 427 )
25.2	写点 .....	( 428 )
25.3	画线 .....	( 431 )
25.4	画矩形和填充矩形 .....	( 434 )
25.5	综合应用 .....	( 436 )

## 第四部分 C语言软件开发

<b>第二十六章</b>	<b>汇编语言子程序接口 .....</b>	<b>( 444 )</b>
26.1	汇编语言接口 .....	( 444 )
26.2	C编译器的调用约定 .....	( 445 )
26.3	Microsoft C语言的调用约定 .....	( 445 )
26.4	建立汇编语言函数 .....	( 446 )
26.5	建立汇编语言框架 .....	( 453 )
26.6	使用asm .....	( 453 )
26.7	何时用汇编编程 .....	( 453 )

<b>第二十七章 G语言软件工程</b> .....	( 456 )
27.1 自顶向下设计 .....	( 456 )
27.2 保护函数法 .....	( 458 )
27.3 函数原型法 .....	( 459 )
27.4 lint和make .....	( 460 )
<b>第二十八章 效率、移植和调试</b> .....	( 463 )
28.1 效率 .....	( 463 )
28.2 程序移植 .....	( 468 )
28.3 调试 .....	( 470 )
28.4 程序维护的艺术 .....	( 476 )

## 第五部分 新的进展

<b>第二十九章 C++</b> .....	( 478 )
29.1 数据抽象 .....	( 478 )
29.2 目标 .....	( 479 )
29.3 注释 .....	( 480 )
29.4 类别 .....	( 480 )
29.5 函数复用 .....	( 486 )
29.6 操作符复用 .....	( 487 )
29.7 C++的其它特性 .....	( 490 )

## 附录A K&R C与ANSIC的差异

A.1 关键字删除 .....	( 491 )
A.2 关键字扩充 .....	( 491 )
A.3 传递结构 .....	( 492 )
A.4 函数原型 .....	( 492 )
A.5 现代与经典函数说明 .....	( 493 )
A.6 标准函数库 .....	( 493 )
A.7 附加预处理器命令 .....	( 493 )

## 第一部分 C语言

本书的第一部分对C程序设计语言给出了一个全面的论述。第一章是对C语言的总论，有更高要求的程序员可以直接跳到第二章。第二章分析了C语言的固有数据类型，变量，操作符和表达式。第三章给出了程序的控制性语句。在第四章中研究了数组和串。第五章讨论了指针。第六章讨论了函数，及第七章包括结构、联合和用户自定义类型。第八章分析了控制台I/O。第九章包括文件I/O，第十章讨论C语言预处理器和注释。

本书中材料（本书大部分材料）反映ANSI标准。但是，原来的C标准，如UNIX版本5.0的C标准也作了讨论，重要的差别特别指出。本书既包含ANSI又包含原C语言标准，保证你能找到同你的C语言编程有关的信息。

### 第一章 C语言概述

本章的目的是对C程序设计语言给出一个轮廓性的描述，同时讨论它的发展过程，应用范围以及基本原则。本章主要为C语言的新手而写。

#### 1.1 C语言的起源

C语言的最初设计和实现是由Dennis Ritchie在DEC PDP-11上的UNIX操作系统环境下完成的。C语言的前身是BCPL语言，后者在欧洲目前还有应用。BCPL语言是由Martin Richards设计的，它深刻地影响了Ken Thompson设计的B语言的风格。B语言在1970年发展成C语言。

很多年以来，C语言的实际标准一直由UNIX操作系统第五版所支持。它的描述在Brian Kernighan和Dennis Ritchie于1978年出版的《C程序设计语言》这本书中。随着微型机的大众化，各种C语言的实用版本纷纷出现。在不同版本上，C语言的源程序可以奇迹般地保持高度兼容。另一方面，由于没有一个标准，它们彼此之间也有一些差异。为了改变这种情况，美国国家标准局ANSI成立了一个委员会，他们在1983年夏初建立了一个C语言的标准。在本书的写作过程中，这个标准已经很完整了，它已在1987年被采用。所有主要C编译器已经实现了ANSI标准。本书完全包括了ANSI C标准，并强调这一点。同时，还包含了旧的C语言UNIX版本的内容。换一句话说，不管你用什么C语言编译器，在这里会找到适用的材料。

#### 1.2 C是一种中级程序设计语言

一般都把C语言叫做中级计算机语言。所谓中级语言并不是指它的能力弱或者难于使用，也不是指它不如BASIC或PASCAL之类语言高级。C语言并不象汇编语言那样经常使用者造成麻烦，它只不过是把高级语言的一些特征同汇编语言的能力结合起来形成的中级语言。表1-1列出有关语言的分类以及C在这些语言中的位置。

高级语言	中级语言	低级语言
Ada	C	Assembler
Modula-2	FORTH	
Pascal	Macro-assembler	
COBOL		
FORTRAN		
BASIC		

表 1-1 C语言在计算机语言中的位置

C语言作为一种中级程序语言，它允许对字节，字位和地址直接操作，而地址是计算机的基本工作单位。C语言也很便于移植，所谓可移植性是指一个软件从一种计算机类型转移到另一种计算机类型仍能保持其功能。例如，如果一个在Apple上写的程序可以在IBM PC机上运行，我们就说这个程序是可移植的。

所有的高级程序设计语言都支持数据类型这个概念。数据类型定义了一组值，可以通过一些操作来存取具有这种类型的变量。普通的数据类型包括整型，字符型和实型。尽管C有五种基本类型，但它并不象Pascal或Ada语言那样对类型要求那么严格。C语言几乎允许所有类型互相转换。例如，在很多表达式中，C语言都允许整型和字符型自由混用。C语言也不进行象数组越界或者变量类型不匹配这一类的运行错误检查。这些类型的检查是程序员的事情。

同样，原C语言版本不对函数参数同调用函数的自变量作一致性检查。例如，在C语言原版本中，虽然函数实际上定义为要用浮点自变量类型，但是你用指针调用函数，是不会发生错误信息的。然而，ANSI标准介绍了函数原型的概念，也就是说，如果程序员指定函数原型，则报告那些潜在的错误。（函数原型在第六章中讨论）。

C语言的特殊性表现在它允许对位域、字节，字和指针的直接操作。这就使它很适合频繁使用这些操作的系统级程序设计。C语言的另一个特征是仅有32个关键字（Kernighan和Ritchie定义了27个，ANSI标准化委员会增加了5个）。这些关键字构成了C语言的骨架。与此对应是IBM PC机上的BASIC语言有159个关键字。

### 1.3 C是一种结构化程序语言

所谓C语言的块状结构并不是一种严格适用于C语言的说法，我们一般把C语言称为结构化程序语言，这是因为它们结构很象ALGOL，PASCAL和Modula-2。严格的块结构定义是指允许在过程或函数内部说明另外的过程或函数。在这种情况下，全局变量和局部变量的概念被作用域规则所代替，该规则控制变量或过程的可见性。因此，既然C语言不允许在函数内部建立函数，它也就不能叫做块状结构的语言。

结构化语言的一个显著性质就是代码和数据的分离。这种语言可以为完成特定任务所用的信息和指令同程序的其它部分分隔开。所用的方法就是使用局部变量和子程序，利用局部变量，程序员可以写出对程序其它部分没有作用的子程序。这样，他就可以很容易地写出需要写的代码段。换句话说，如果你要开发一个独立的函数，你只需要知道这个函数要做什么，而知道它怎样做。需要注意的是，过份地使用全局变量（在整个程序内都起作用的变量）的副作用，从而使程序出错（使用过BASIC的程序员都会遇到过这个麻

烦)。

一种结构化程序语言允许你在程序设计中有很大的选择余地，它直接支持各种循环结构如：while, do-while和for语句。在结构化语言中，goto语句的使用是被禁止或者不提倡的，它不象在BASIC或FORTRAN语言中那样作为一种常见的程序控制结构。另外，结构化语言允许你采用缩进语句书写方法，同时也没有严格的语句位置概念（FORTRAN则严格要求语句的位置）。

下面我们列出一些结构化语言和非结构化语言的例子：

非结构化语言	结构化语言
FORTRAN	pascal
BASIC	Ada
COBOL	C
	Modula-2

结构化语言是现代的发展趋势，而非结构化语言正在过时。实际上，老牌计算机语言的标志就是它是非结构化的。目前广泛认为，结构化语言要比非结构化语言更便于编写程序，同时，结构化语言程序的清晰性使得维护它们也更加容易。

C语言的主要结构成份就是函数（也就是C中的独立子程序）。在C语言中，函数被描写成一个块状结构，所有的程序动作在其内部进行。该语言允许把程序的每个任务的定义和编程独立实现，因而使你的程序易于模块化。一旦函数建立，你可以把它放在不同环境下工作，而不必担心会对程序的其他部份产生副作用。能够建立独立性强的函数是大型项目中非常严格的要求。在那里，一个程序员写的代码不会影响另一个人的代码是非常必要的。在二十六章我们将讨论如何在大型项目的开发中消除副作用。

C语言中另一个结构化和独立化是使用代码块。一个代码块是指一组程序语句在逻辑上可以当作一个单位来使用。C语言中的代码块是靠在一组语句的前后加上{ }来构成，例如：

```

if ( x < 10 ) {
    printf ( "too low, try again/n" );
    scanf ( "%d", &x );
}

```

if语句后面的两条语句作为一个整体在x<10的情况下执行。这两条语句构成了一个代码块。它们是一个逻辑单位，必须同时执行。需要注意的是，每一个语句既可以是单个语句，也可以是语句块。使用代码块既可以使算法更加清晰，优美和高效，也可以帮助程序员看清一个过程的本来面目。

### 1.4 C是面向程序员的语言吗？

有人会对这句话提出疑问，难道不是所有计算机语言都是面向程序员的吗？答案是：“不”。有些语言如COBOL和BASIC就是面向非程序员的经典例子。COBOL语言的设计目标不是为了使程序员更自由，而是为了改善代码的可读性。它不关心代码的可靠性，也不关心代码执行的速度，只是想非程序员能读懂这个程序。BASIC语言最初也是为非程序员编写简单的计算机程序而设计的。

与此相反，C语言完全是实际程序员建立，影响和使用的语言。结果是C语言提供了程序员想要的一切：很少的限制，没有什么修饰，块结构，独立函数和很少的关键字。使用C语言可以使程序员达到使用汇编语言的执行效率，同时还保留ALGOL和Modula-2的结构化特性。毫无疑问，在非常优秀的程序员中，C语言是一种很通用的程序设计语言。

C语言可以在程序员之间流行的一个重要因素是它可以代替汇编语言。汇编语言用符号来表示计算机能够直接执行的二进制代码。每一条汇编语言的语句都对应着计算机实际执行的一个动作。尽管汇编语言可以给程序员完成其任务的最大灵活性和执行效率，但却很难用汇编语言来开发和调试程序。而且由于汇编语言是非结构化的，最后写出的程序象一团乱麻，充满了转移语句，调用语句和变址语句。这种缺乏结构性的汇编语言程序很难阅读，改进和维护。更重要的是，汇编语言程序在不同的机器（CPU）之间不可移植。

最初，C语言是用来进行系统程序设计的。系统程序是在大量计算机程序中构成操作系统及其支持程序的部分。下列程序一般被叫做系统程序：

- 操作系统
- 解释系统
- 编辑程序
- 汇编程序
- 编译程序
- 数据库管理系统。

随着C语言的流行，很多程序员利用它的可移植性和高效性开始编制各类程序。因为很多机器上都有C语言编译程序，所以可以把在一种机器上编写的程序很少或不加以改变就能运行在另外的机器上。这种可移植性节省了时间和金钱。C编译程序还可以产生非常紧凑和高效的目标代码，这要比BASIC编译程序好得多。

使C语言能用于各种场合的一个重要原因就是程序员喜欢用它。C语言有汇编语言的速度和FORTH语言的能力，又没有Pascal和Modula-2的限制。每个C语言程序员都可以建立和维护特有的函数库，以适应他自己的风格并能应用于各种不同的程序中。因为C语言允许并鼓励单独编译，所以这使程序员可以方便地管理大型项目，并减少重复性工作。

## 1.5 编译器和解释器

编译和解释是指程序的执行方式。从理论上说，任何程序语言都可以既编译执行，又解释执行，但是，有一些语言通常采用其中的一种形式。例如，BASIC语言通常是解释执行的，而C语言则是编译的（最近有一些C语言解释器出现，用来帮助调试程序）。一个程序的执行方式并不取决于它用哪种语言编写。解释和编译的概念只是说明怎样对你的源程序进行处理。

一个解释器读入你的程序的源代码，一次一行，并执行这一行给出的特定指令。一个编译器则一次读入整个程序，并把它转换成目标代码，这种目标代码可以直接在机器上执行。目标代码也称为二进制代码或机器代码。一旦程序编译后，源代码中的行就不会在程序执行过程中体现出来了。

当使用解释器时，它必须在每次你要运行程序时存在。例如，你使用BASIC语言时，就必须先执行BASIC解释程序，然后调入你的程序，并用RUN命令启动它。BASIC解释器每

序中的一行，先排除错误再执行它。每次程序执行时，都要重复这个缓慢的过程。编译器把你的程序转换成可以直接在计算机上运行的目标代码。由于编译器并行一次转换，以后你就可以直接执行你的程序。换句话说，仅仅需要输入程序了。总之，编译只是一次开销，而解释则是程序每次运行时都要开销。

## C语言的形式

表1-2列出C语言的32个关键字，它们同正式C语言语法一起构成了C程序设计语言。其27个由原C版本定义。这五个关键字enum, const, signed, void,和volatile是ANSI委员会补充的。

auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while

表 1-2 ANSI C关键字表

另外，C编译器已加了几个关键字，便于更好地利用8088/8086类处理器的存储结构，和支持混合语言的程序设计以及支持中断处理。这里列出最常用的、已扩充的关键字：

asm	_cs	_ds	_es
_ss	cdecl	far	huge
interrupt	near	pascal	

你的编译器也可能支持其它的扩充，有助于更好地利用特定的环境。

所有的关键字都是小写字母。在C语言中大写字母和小写字母是不同的：小写的else是关键字，而大写的ELSE则不是。关键字不能在C语言程序中用于其他目的，也就是说，不能用来作为变量或函数的名字。

所有的C语言程序都包含一个或多个函数。唯一不可缺少的函数叫做main( )。它在程序开始执行时第一个调用。在优秀的C语言程序中，main( )实质上包含程序的轮廓。这种轮廓由一系列函数调用组成。尽管main( )这个词技术上并不是C语言的一部分，但我们经常要把它看成是这样。不要用main( )作为变量的名字，因为这会给编译器带来极大混乱。

C语言程序的一般形式如图1-1，在这里f1( )到fn( )代表用户定义的函数。

```

global declarations
return-type main( parameter list )
{
    |
    | statement sequence
    |
}
return-type f1( parameter list )
{

```

```

    statement sequence
}
return-type f2 ( parameter list )
{
    statement sequence
}
.
.
.
return-type fN ( parameter list )
{
    statement sequence
}

```

图 1-1 C程序的一般形式

## 1.7 库和链接

从技术上说,完全用程序员自己写的语句就可以构成一个有用的和功能性的C语言程序。不过这种情况很少见,因为C语言(作为一种语言定义)并没有提供任何执行输入输出操作的手册。结果,很多程序中都调用了C语言标准程序库中的函数。

所有的C语言编译器都附带一个标准函数库,用来提供常用的操作。ANSI C标准规定了包含在库中的最小函数集。然而,你的编译器可能包含许多其它函数。例如,ANSI标准不定义任何图形函数,但是你的编译器很可能包括一些图形函数。

有些C语言实现中,函数库是一个很大的文件;在另一些版本中,则提供多个小文件,并用一个装配程序保证其有效性和实用性。为简便起见,本书只提供“库函数”这个词来指明这两种情况。

在你的C语言编译器中会有很多你所需要的通用函数。当你调用不属于你自身程序的函数时,C语言编译器会记住这个名字,然后连接程序会把你写的程序与标准函数库中的目标代码进行连接。这个过程叫做程序的连接。有些C编译器有自己的连接程序,另一些则用操作系统提供的标准连接程序。

函数库中的函数采用的是可重新定位的格式。这就是说,机器代码的内存地址并不是绝对的,仅仅保留相对偏移信息。当你的程序同标准库中的函数进行连接时,这些内存偏移才变为实际使用的地址。有一些技术手册和书专门来解释这个过程。不过你不必在C语言程序中关心实际重定位过程。

你在编写程序所需要的大部分函数都会在标准库中找到。你所做的仅仅是把这些砖块搭在一起。如果你编写了一个经常要使用的函数,你最好把它放在函数库中。有些编译器允许你把你自己的函数加入标准库;另一些则要求你建立一个附加数库。不论哪种方法,你都可以不断使用你所编写的这个函数。

记住,ANSI标准只规定了最小标准库。大多数编译器提供远比ANSI定义的函数要多的库。此外,一些在原UNIX C版本中的函数不由ANSI C标准定义,因为它们冗余的。本书不仅讨论了最重要的和最广泛使用的、旧UNIX C标准定义的函数,而且讨论了ANSI定义的所有函数。另外,也涉及了几个既不由ANSI定义又不由旧UNIX标准定义的广泛使用的



函数。（非ANSI函数将作出说明，以便避免混淆。）

### 1.8 分别编译

很多短小的C语言程序完全可以装在一个源文件中。但是，随着程序长度的增加，编译的时间会越来越长。C语言允许把程序分成几块，分别放在不同的文件中，并且分别编译这些文件。一旦所有的文件都编译好，就可以把它们同其他的标准库函数连接起来，构成完整的目标代码。分别编译的优点是当你只改变一个文件中的代码时，你不必重新编译整个程序。除了非常简单的程序以外，别的情形就节省了大量时间。（在第四部分中我们将详细讨论分别编译）。

### 1.9 编译C程序

编译一个C语言程序包括下面三个步骤：

1. 建立你的程序
2. 编译你的程序
3. 把程序同必要的库函数连接起来

一些C编译器提供完整的程序设计环境，包括编辑程序。有一些，你必须用一个单独的编辑器建立你的程序。编译器只接收标准的文本文件。一般编译器不识别某些字处理软件生成的文件，因为它们含有控制字符和不可打印字符。

编译程序的适当方法依赖于你使用什么样的C编译器。另外，如何实现链接随编译和环境而改变。有关这些细节，查阅你的用户手册。

### 1.10 C语言存储映象

一个编译好的C程序建立并使用四个独立的逻辑区域，这些内存区域用于不同目的。第一个内存区域用来存放程序的代码部分。下一个区域用来存贮全局变量。余下的两个部分分别称为栈和堆。栈在程序运行时起着很大作用，它保留函数调用的返回地址，函数的自变量以及局部变量。它也用来保存当前CPU的状态。所谓堆是一块空闲存贮区，当你的程序要使用C语言的动态存贮分配函数，为链表和树留出空间时，就要用到。

程序的确切分布可能随不同的编译器和不同的环境而变化。例如，8086类处理器的大多数C编译器有六种不同方法组织内存的结构，这是因为8086的段存储结构的缘故。8086类处理器的存储模型在本书的后面讨论之。

由于CPU类型和C语言版本的不同，造成这四个区域实际分布的位置不同。图1-2给出一般性的C语言存储映象图。

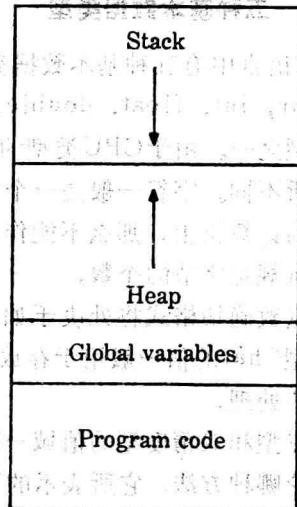


图 1-2 C程序存储映象