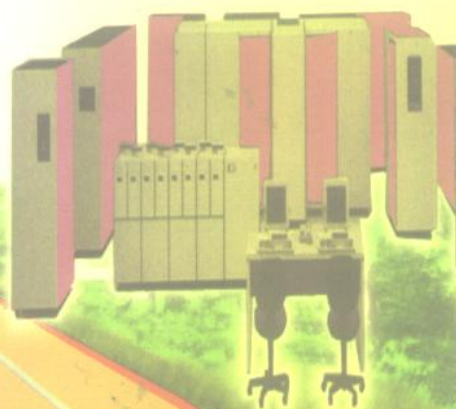


高等学校计算机系列丛书

C 语言程序设计

朱晋蜀 李向阳 主编



重庆大学出版社

7p312
ZJS/1

C 语言程序设计

朱晋蜀 李向阳 主编

重庆大学出版社

内容简介

全书共分十一章,主要包括C语言导引、基本数据类型、控制流语句、函数、数组、程序设计导论、指针、结构与联合、文件I/O、预编译、标准库函数。

本书注意了C语言的系统性和完整性,在内容安排上重点突出、难点分散、难易结合、深入浅出。对C语言的基本概念均配有英文术语,每章末附有英文习题和实验上机题。

本书可作为高等专科学校计算机专业、本科院校非计算机专业教材,亦可供从事计算机应用的技术人员及计算机爱好者使用。

C语言程序设计

朱晋蜀 李向阳 主编

责任编辑 梁 涛

重庆大学出版社出版发行

新华书店经销

重庆电力印刷厂印刷

开本:787×1092 1/16 印张:15.75 字数:393千

1997年7月第1版 1998年5月第1次印刷

印数:6001—12000

ISBN7-5624-1343-6/TP·121 定价:17.50元

序

面对知识爆炸,社会学家们几乎都开出了一个相同的药方:计算机。计算机也深孚众望,以其强大的功能,对人类作出了巨大的贡献,取得了叹观止矣的成就。自它1946年2月14日在美国费城诞生以来,至今已过“知天命”的年龄了。现在,计算机已是一个庞大的家族。如果说,它的成员占据了世界的每一个角落和每一个部门也并不过分,甚至找不到这样一个文明人,他的生活不直接或间接与计算机有关。目前,全世界计算机的总量已达数亿台,而且,现在正以每年几千万台的速度增长。

作为计算机在信息传递方面的应用,计算机加上网络,被认为是和能源、交通同等重要的基础设施。这种设施对信息的传递起着异常重要的作用。西方发达国家和我们国家对此都非常重视。例如,美国的信息高速公路计划,全球通讯的“铱”计划,我国也开始实行一系列“金”字头的国民经济管理信息化计划。这些计划中唱主角的设备便是计算机。计算机在各个方面的应用不胜枚举,我们每个人都自觉不自觉地处于计算机包围中。

计算机对社会生产来说是一个产业大户,对每个现代人来说是一种工具,对学生们来说,它是一个庞大的知识系统。面对计算机知识的膨胀,面对计算机及其应用产业的膨胀,计算机各个层次的从业人员的需要也在不断膨胀,计算机知识的教育也遍及从小学生到研究生的各个层次。

为了适应计算机教学的需要,重庆大学出版社近几年出版了大量的计算机教学用书,这一套教材就是一套适应专科层次的系列教材。我们将会看到,这一套教材以系列、配套、适用对路,便于教师和学生选用。如果再仔细研究一下,将会发现它的一系列编写特色:

1. 这些书的作者们是一些长期从事计算机教学和科研的教师,不少作者在以前都有大量计算机方面的著作出版。例如本系列书中的《Visual Fox Pro 中文版教程》的作者,十年前回国后最早将狐狸软件介绍到祖国大陆,这一本书已是他的第八本著作了。坚实的作者基础,是这套书成功的最根本的保证。

2. 计算机科学是发展速度惊人的科学,内容的先进性、新颖性、科学性是衡量计算机图书质量的重要标准,这一套书的作者们在这方面花了极大的功夫,力求让读者既掌握计算机的基础知识,又让读者了解最新的计算机信息。

3. 在内容的深度和知识结构上,从专科学生的培养目标出发,在理论上,从实际出发,满足本课程及后续课程的需要,而不刻意追求理论的深度。在知识结构上,考虑到全书结构的整体优化,而不过分强调单本书的系统性。这样,在学过这一套系列教材后,学生们就可在浩瀚的计算机知识中,建立起清晰的轮廓,就会知道这些知识的前因后果,就会了解这些知识的前接后续。使学生们能在今后的工作实践中得心应手。

4. 计算机是实践性很强的课程,仅靠坐而论道是学习不了这些知识的。所以从课程整体设置来讲,包括有最基本的操作技能的教材。对单本书来说,在技术基础课和专业课中,都安排有一定的上机实习或实验,这样可使学生既具备一定的理论知识以利今后发展和深造,又掌握实际的工作技能胜任今后的实际工作。

编写一套系列教材,这是一个巨大的工程。这一套书的作者们,重庆大学出版社的领导和编辑们,都为此付出了辛勤的劳动。作为计算机工作者,以此序赞赏他们的耕耘,弘扬他们的成绩。

周明凡

1997年6月15日

前 言(Preface)

本书共分十一章,参考学时为 70 学时(其中 22 学时上机),主要介绍了 C 语言导引、基本数据类型、控制流语句、函数、数组、程序设计导论、指针、结构与联合、文件 I/O、预编译和标准库函数。

现在 C 语言的教科书不少,但适用于工程类大专层次的教材并不多。本书是基于:“基础理论教学以应用为目的,以必需够用为度,以掌握概念,强化应用为教学重点,教学应加强针对性和实用性,加强实践环节,注意教学内容的先进性”和“专科英语教学三年不断线”的思路编写的。每章附有习题和实验题。正文中的 C 语言基本概念配有相应的英文术语。本书中选用的英文习题,在保持原文风格的基础上,考虑大专二年级学生的英文水平做了适当的简化,这对锻炼大专生“用英语”的能力是大有帮助的。

由于 C 语言涉及的概念、规则相互牵连,所以很难按知识结构的先后顺序来编写。在编写时,不得不穿插引入概念,比如讨论运算符时还没有讲语句,但又需要语句的配合;在讨论语句时,又需要函数的配合等。教师在讲授时应该让学生明确这一点,引导学生采用“立体式”阅读方法,对于还没有详细讲到的概念要翻阅后面的章节,这样来把握 C 语言规则的内在联系。

本书由朱晋蜀、李向阳主编。朱晋蜀编写第一、二、三、六章,选编全书英文习题,并负责全书统稿,李桂清编写第八、九、十、十一章,李向阳编写第四、五、七章。该书由电子科技大学龚天富教授主审。

由于编者水平有限,书中难免还存在一些错误和缺点,希望专家和读者指正,以期在今后修编时更正。

编 者

1997 年 4 月

目 录(Contents)

第一章 C语言导引(Introduction to C)	1
1.1 C语言简述(An Overview of C)	1
1.2 C语言程序(C Language Programs)	2
1.3 编程基础(Programming Basis)	4
1.4 C系统(C System)	10
小 结(Summary)	11
思考题(Points to Ponder & Exercises)	12
实 验(Experiments)	13
第二章 基本数据类型(The Fundamental Data Types)	15
2.1 说明和表达式(Declarations and Expressions)	15
2.2 基本数据类型(The Fundamental Data Types)	16
2.3 char 类型(The Data Type char)	17
2.4 int 类型(The Data Type int)	19
2.5 整数类型(The Integral Types)	20
2.6 浮点类型(The Floating Types)	21
2.7 类型定义(The Use of typedef)	22
2.8 容量运算符(The sizeof Operator)	22
2.9 宏 getchar()和 putchar()	23
2.10 数学函数(Mathematical Functions)	25
2.11 类型转换(Conversions and Casts)	26
小 结(Summary)	28
思考题(Points to Ponder & Exercises)	29
实 验(Experiments)	30
第三章 控制流语句(Flow of Control)	32
3.1 关系和逻辑运算符(Relational and Logical Operators)	32
3.2 复合语句(Compound Statement)	37
3.3 空语句(Empty Statement)	38
3.4 if 和 if-else 语句	38
3.5 while 语句	40
3.6 for 语句	43
3.7 程序举例(An Example of Boolean Variables)	44
3.8 逗号运算符(The Comma Operator)	45
3.9 do 语句	46
3.10 程序举例(An Example of Fibonacci Numbers)	47
3.11 goto 语句	49
3.12 break 和 continue 语句	50
3.13 switch 语句	51
3.14 条件运算符(The Conditional Operator)	52

小 结(Summary)	53
思考题(Points to Ponder & Exercises)	54
实 验(Experiments)	56
第四章 函数(Functions)	58
4.1 函数的定义(Function Definition)	58
4.2 return 语句	59
4.3 函数的原型(Function Prototypes)	61
4.4 函数的调用(Function Invocation)	62
4.5 递归调用(Recursion)	64
4.6 存储类型(Storage Classes)	65
小 结(Summary)	72
思考题(Points to Ponder & Exercises)	73
实 验(Experiments)	76
第五章 数组(Arrays)	78
5.1 一维数组(One-dimensional Arrays)	78
5.2 二维数组(Two-dimensional Arrays)	80
5.3 字符数组和字符串(Character Arrays and Strings)	82
5.4 数组作函数的参数(Arrays as Function Arguments)	86
小 结(Summary)	89
思考题(Points to Ponder & Exercises)	89
实 验(Experiments)	91
第六章 程序设计导论(Introduction to Programming)	94
6.1 算法与数据结构(Algorithms and Data Structures)	94
6.2 “八皇后”问题(The Eight Queens Problem)	95
6.3 结构化程序设计(Constructed Programming)	99
小 结(Summary)	104
思考题(Points to Ponder & Exercises)	104
实 验(Experiments)	105
第七章 指针(Pointers)	106
7.1 指针变量(Pointer Variables)	106
7.2 指针作函数参数(Pointers as Function Arguments)	108
7.3 指针与数组(The Relationship between Pointers and Arrays)	112
7.4 字符指针与字符串(Character Pointers versus Strings)	121
7.5 指针数组(Using Pointers with Arrays)	124
7.6 命令行参数(main with Arguments)	127
7.7 函数指针(Pointers to Functions)	129
小 结(Summary)	134
思考题(Points to Ponder & Exercises)	134
实 验(Experiments)	138
第八章 结构与联合(Structures and Unions)	140
8.1 结构(Structures)	140
8.2 运算符优先级(Operator Precedence and Associativity)	145

8.3	类型定义 (typedef)	146
8.4	结构与数组 (Using Structures with Arrays)	147
8.5	结构与指针 (Using Structures with Pointers)	151
8.6	结构与函数 (Using Structures with Functions)	154
8.7	位域 (Bit Fields)	160
8.8	联合 (Unions)	163
8.9	枚举 (Enumeration Types)	166
8.10	自引用结构 (Self-referential Structures)	168
	小结 (Summary)	175
	思考题 (Points to Ponder & Exercises)	176
	实验 (Experiments)	181
	第九章 文件 I/O	183
9.1	文件的基本概念 (Fundament Concepts of Files)	183
9.2	标准级文件的打开与关闭函数 (fopen() and fclose())	185
9.3	标准级文件 I/O 函数 (Standard I/O Functions)	188
9.4	文件定位函数 (File Positioning Functions)	194
9.5	文件的块存取 (Direct input/output)	196
9.6	低级文件 I/O (System I/O)	199
	小结 (Summary)	206
	思考题 (Points to Ponder & Exercises)	207
	实验 (Experiments)	208
	第十章 预编译 (The Preprocessor)	210
10.1	宏 (Macros)	210
10.2	文件包含 (include)	214
10.3	条件编译 (Conditional Compilation)	215
10.4	其它预处理命令 (Other Preprocessing Operators)	217
	小结 (Summary)	219
	思考题 (Points to Ponder & Exercises)	219
	实验 (Experiments)	223
	第十一章 标准库函数 (The Standard Library)	224
11.1	字符类函数 (Character Functions)	224
11.2	字符串函数 (String Functions)	225
11.3	数学函数 (Mathematical Functions)	229
11.4	内存管理 (Memory Handling)	230
11.5	图形处理 (Graphics Handling)	233
	小结 (Summary)	239
	思考题 (Points to Ponder & Exercises)	239
	实验 (Experiments)	239
	参考文献	241

第一章 C 语言导引(Introduction to C)

人们通常从 1,而不是从 0 开始计数,C 语言中计数则是从 0 开始。例如,数组的下标从 0 开始。0 还用来表示逻辑状态“假”,非 0 表示“真”。0 在 C 语言中还有其它一些特殊含义,如 '\0' 表示字符串结束。对于初次接触 C 语言的人来说,一定要改变对 0 传统的认识。记住 0 在 C 语言中是一个既重要又特殊的数。在这一章里,主要讨论 C 语言中的“字符”、“关键字”、“标识符”和“运算符”等基本概念。这些概念在后续章节中还要进行深入的讨论。总之,要学好 C 语言,一定要“从零开始”。

1.1 C 语言简述(An Overview of C)

C 语言是一种高级程序设计语言。所谓“高级”是相对于机器语言或汇编语言这些“低级”语言而言的。C 语言由贝尔实验室的 Dennis Ritchie 设计,1972 年在小型机 PDP-11 上实现。著名的 UNIX 操作系统就是用 C 语言写成的。80 年代后期,美国国家标准委员会对 C 语言的语法进行了规范,定义了标准 C 语言,也叫 ANSI C。现在 C 语言已经成为一种用途十分广泛的通用程序设计语言,在不同的计算机或不同的操作系统下都能够运行 C 语言。在 C 语言之后,又出现了面向对象的 C++ 语言。在本书中主要研究 C 语言,因为它是 C++ 的基础。C 语言是一种精练的程序设计语言。它与其它高级语言相比(例如 PASCAL 语言),有较少的关键字(Key words),也叫保留字(Reserved words)。C 语言有功能强大的运算符集,并且有的运算符允许程序员访问到位(bit)级。C 语言的增量运算符“++”在许多机器上就是直接模拟机器语言的“运算器加 1”操作,而不像其它高级语言是用“变量加 1”的二元运算来实现此功能的。C 语言的表达式融运算和寻址为一体,有时一个表达式能实现其它高级语言需多条语句才能实现的功能。所有这些就是 C 语言与其它高级语言的重大区别。即 C 语言既是高级语言,同时又兼有某些汇编语言的特点。软件工程的研究表明,一个程序员平均每天只能写出 20 条左右的有效代码。C 语言的精练无疑正是程序员们所期望的。

C 语言是可移植语言。在某种计算机上写的 C 语言程序可以方便地在另一种计算机上编译通过。C 语言为程序员提供了在各种计算机上都相同的标准库函数,C 语言还有预处理器,使得程序员在任何时候都可用标准语法编程。

C 语言是模块化语言。所有 C 模块均是以函数形式出现的。C 语言函数的参数是以“传值”方式传递。C 语言的(变量)“静态存储”类型提供了某种屏蔽作用,亦即在不同的文件和函数模块中可以使用同一变量名。所有这些,再加上操作系统提供的工具,完全具备了开发用户自定义库函数和模块化程序设计的条件。

C 语言是 C++ 的基础。C 程序员所用的许多结构和程序设计方法同样也被 C++ 程序员所采用。可以认为学习 C 语言是学习 C++ 的第一步。

C 语言在大多数计算机上的运行效率高,因为 C 语言中某些结构与机器无关。C 语言程序以最适应某种机器体系结构的方式来实现。

当然 C 语言不是一种十全十美的程序设计语言。C 语言的不方便之处是语法太复杂。C 语言编译也不检查数组的上下界,对数据类型的检查也不严格。C 语言运算符的一符多用是 C 语言的另一大特点,如运算符“*”和“=”等都有多种用途。最容易使初学者出错的就是误把赋值等“=”当做相等运算符“==”。然而不管怎么讲,C 语言是一种计算机专业人员首选的编程语言,它的简洁、优美和高效受到愈来愈多的程序员喜爱。

ANSI 是美国国家标准研究所(American National Standard Institute)的简称。该研究所制定了包括程序设计语言在内的许多系统标准。80 年代后期,该委员会推出了 ANSI C 或叫标准 C 语言。1990 年,国际标准组织(ISO)通过了 ANSI C。所以现在的 ANSI C 或 ANSI/ISO C 是国际上承认的标准。C 语言标准是以规定 C 语言程序的写法来表现的,对 C 语言程序有唯一的解释。制定标准的目的是使 C 语言具有更好的可移植性、可靠性和可维护性,不同机器上的 C 编译器编译出的 C 语言程序可以在不同机器上正确运行。

1.2 C 语言程序(C Language Programs)

C 语言与其它程序设计语言一样,定义了字符集和语法(Syntax)。按语法写出的程序称为 C 语言源程序(Source Program),或简称代码。检查源程序是否正确并可将其翻译成机器代码的程序叫编译器(Compiler)。编译器会检查出 C 代码中的语法错误。如果没有错误,源代码就是合法的 C 语言程序,然后编译器将 C 程序翻译成目标代码(object code),最后通过链接(link)成为可执行文件。

C 编译器将组成 C 程序的一连串字符翻译成目标代码,再将目标代码转换成特定机型的机器语言。编译器首先是把程序中出现的符号收集在一起,以建立 C 语言的词汇集。在标准 C 语言中有 5 类符号(token):关键字(keywords)、标识符(identifiers)、常量(constants)、运算符(operators)和分隔符(punctuators)。编译器根据标准 C 语言的语法检查符号的正确性。C 编译器是一种机械式的翻译器,这与人阅读不同,人阅读时遇到一个错误标点或一两个错字仍能正确理解文章的意思。编译器则不同,哪怕是错一个字符它都无法翻译。

程序员应该追求写出可读性好的程序。要做到这一点的关键是多加注释和使用有含义的标识符。

C 语言程序是由一串字符组成的。可用于 C 程序的字符集如下:

小写字母 a b c ... z

大写字母 A B C ... Z

数字 0 1 2 3 4 5 6 7 8 9

符号 + - * / = () [] < > ' "

! @ # \$ % & _ | . , ; : ?

空白键 空格键 回车键和跳格(tab)键

这些字符(symbol)是构成符号的基础。下面看一个简单程序中出现的符号。

[例 1.1] 读入两个整数并打印其和。

```

/* 读入两个整数并要印其和 */
#include <stdio.h>
int main(void)
{
    int a, b, sum;
    printf("Input two integers: ");
    scanf("%d%d", &a, &b);
    sum = a + b;
    printf("%d + %d = %d\n", a, b, sum);
    return 0;
}

```

[例 1.1]程序的分析:

1. /* 读入两个整数并要印其和 */

注释以 /* 和 */ 为界。编译器用一个空格替代注释,并不做更多的处理。注释的用途是帮助人们阅读理解程序。

2. #include <stdio.h>

这行由编译器的预处理器处理。它的作用是将<stdio.h>这个标准头文件包括进来,因为程序中用到的函数 printf() 和 scanf() 的原型是在该文件中定义的。函数的原型其实是一种说明。编译器在函数生成目标代码时需要它们的原型。

3. int main(void)

```

{
    int a, b, sum;

```

编译器将这些字符分为四类符号。main 是一个函数名标识符,紧跟其后的括号()是个运算符。这在一开始时会引起困惑,因为跟在 main 后的是(void),但是仅有()才是运算符。这个运算符告诉编译器 main 是个函数。其它字符“{”、“,”、“;”是标点符。int 是关键字。a、b、和 sum 是标识符。

4. int a, b, sum;

int 和 a 之间的空格用来区分两种不同类型的符号,不能写成 inta,b,sum。然而逗号后的空格可要可不要。可以写成

```
int a,b,sum;
```

但不能写成 int absum, 否则编译器会把 absum 看成是一个标识符。

5. printf("Input two integers: ");

```
scanf("%d%d", &a, &b);
```

printf 和 scanf 都是函数名标识符。它们后面紧跟的一对括号说明它们是函数。在编译器完成了 C 代码的翻译后,要通过链接生成可执行程序。如果程序员没有提供 printf() 和 scanf() 的代码,链接程序会到 C 语言的标准库中去找这两个函数的代码。通常不允许程序员定义 C 语言已使用函数的名字,这种名字叫关键字或保留字。

6. "Input two integers: "

用引号括起来的一串字符称为字符常量。编译器把它当做一个符号来对待,也在内存中提

供存储字符常量的空间。

7. `&a, &b`

逗号是个分隔符。字符 `&` 是地址运算符。编译器将其当做符号(token)。尽管字符 `&` 和 `a` 紧靠在一起,但编译器仍然将它们看成是两个符号。我们可以写成

`&a, &b` 或 `&a, &b`

但不能写成

`&a &b` /* 掉了逗号 */

`a&, &b` /* `&` 在 `a` 的左边 */

8. `sum = a + b;`

字符“=”和“+”是运算符。C语言称“=”为赋值运算符。在这个式子中,空格符不起作用,所以可以写成

`sum=a+b;` 或 `sum = a + b ;`

但决不能写成

`s u m = a + b ;`

如果哪样写,编译器会将每一个字母误认为是一个标识符。因为 `s`、`u` 和 `m` 并没有在前面说明过,编译器会指出错误。即使所有的标识符在前面说明过,即

`int s, u, m, a, b;`

但是表达式 `s u` 仍然是一个非法式子。

编译器要么将空格符当做分隔符,要么忽略它。程序员使用大量的空格是为了提高程序的可读性。程序对于编译器而言,仅仅是一串(一维)字符流,但对于人们而言,程序却是一个二维的字符集合。

1.3 编程基础(Programming Basis)

1.3.1 关键字(Keywords)

关键字是已被C语言预定义了的标识符,在程序中起特定含义,也称为保留字。程序员在编程时,不能将它们重新定义为标识符。C语言规定的关键字如下:

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

在不同的机器上或不同的操作系统中,有的编译器可能定义其它一些关键字。比如,下面这些关键字在Turbo C中是有定义的:

`asm` `cdecl` `far` `huge` `interrupt` `near` `pascal`

与大多数高级语言相比,C语言的关键字算是较少的。Ada语言有62个关键字。关键字少是C语言的特点之一。C语言的有效字符集相对较小,但功能却相当强。

1.3.2 标识符(Identifiers)

标识符是由字母或下划线“_”开头的字母、数字和下划线“_”组成的字符串。在大多数编译器中将大小写字母视为不同,C语言也是如此。良好的程序设计风格提倡将有含义的单词定义为标识符,这可以大大增加程序的可读性。下面是一些合法的标识符:

```
k      _id      iaamanidentifier      so _am _i
```

而后面这3个标识却是非法的:

```
not #me      /* 标识符中不能使用符号# */  
101 _south   /* 不能用数字开头 */  
_plus        /* 不要把减号“-”与下划线“_”混淆 */
```

程序中的标识符用来给程序控制的目标或对象命名。关键字可以认为是具有特殊含义的标识符。前面提到的scanf和printf那样的标识符是C语言标准库中的输入输出函数名。这些函数名通常不能重新定义。标识符main是C语言中的特殊标识符,每一个完整的程序必须有且仅有一个main函数。各种操作系统支持下的C编译器之间的区别之一,是规定的标识符长度不同。在一些较早的系统中,可以接受8个以上的字符,但仅有前8个字符有效。编译器会跳过后面的字符。在这种系统中,下面的两个标识符被认为是相同的标识符:

```
i _am _an _identifier      和      i _am _an _elephant
```

在标准C语言中,至少前31个字符是有效的。还有的C编译器识别更长的标识符。

提倡的编程风格中有一条就是尽量使用有含义的标识符。如果编写一个处理纳税问题的程序时,最好选tax_rate(税率)、price(价格)和tax(税款)等做为标识符。所以下列的表达式可读性好:

```
tax = price * tax_rate;
```

下划线“_”通常用来连接本应用空格隔开的一串词。应注意的是,用下划线开头的标识一般是系统(如标准库函数)内部使用。例如,标识符_iob通常是stdio.h函数中定义的结构名。如果程序员一定要用_iob这个标识符,则要么编译器不通过,或程序执行时出现莫名其妙的操作。开发应用程序的程序员,最好不要使用以下划线开头的标识符。

1.3.3 常量(Constants)

在前面程序例子中,我们看到C程序处理各种值。像0和17这样的数字叫整型常量,而像1.0和3.14159叫浮点常量。像其它大多数高级程序设计语言一样,C语言将整数和浮点数视为不同类型的数。在C语言中,还有字符类常量。例如'a'、'b'、和'c'等。字符常量用单引号括起来。在后面可以看到字符常量与整数有些关系。某些字符常量有特殊含义,如像回车符'\n'。反斜杠“\”叫转义字符。把\n看成“不是原来n的意思”。即使\n是两个字符\和n,但C语言编译器只把它看成一个字符,表示回车符。

除了已经讨论过的常量外,C语言还有“枚举”常量,使将在第7章中讨论。

十进制整数是有限的十进制数字串。因为C语言除十进制数外,还有八进制和十六进制数,所以必须小心区别不同类型的数。例如,17是十进制整数,017是八进制数,0x17是十六

进制数。当然像-33这样的负数被认为是常数表达式。

下面是一些十进制整数常量：

0

77

1234567890 /* 这个数可能对某些机器而言太大 */

以下不是十进制整数常数

0123 /* 8 进制整数 */

-49 /* 常数表达式 */

123.0 /* 浮点数 */

虽然已经使用了 144(整数)和 39.7(浮点)这样的常量,但涉及到类型、所需的内存空间和机器精度等问题,将在第二章中详细介绍。

1.3.4 字符串常量(String Constants)

用双引号括起来的一串字符叫做字符串,例如"abc"就是一个字符串常量。编译器将其视为一个符号。字符常量和字符串常量是两种不同的概念,'a'和"a"是不相同的。

注意双引号""仅仅是一个字符。如果双引号本身要在字符串中出现,在其前面必须冠以一个反斜杠"\"。如果字符"\"本身要出现在字符串中,在其前面也要冠以一个反斜杠。下面是一些字符串的例子:

"a string of text"

""

/* 空串 */

" "

/* 由 4 个空格组成的字符串 */

"a = b + c; "

/* 仅是一个字符串,不是一个表达式 */

"/* this is not a comment */"

/* 这不是一句注释 */

"a string with double quotes \" within "

/* 一个字符串 */

"a single backslash \" is in this string"

/* 一个字符串 */

以下不是字符串:

/* "this is not a string" */

"and

neither is this"

如果两个相邻的字符串中间只有一个空格,如"abc" "def",标准 C 将视为一个字符串"abcdef"。在传统的 C 中没有这个功能。编译器把字符串常量看成是符号。与其它常量一样,编译器在内存中提供空间存放字符常量。在第七章讨论字符指针与字符串时,将会重点研究编译器在内存中如何存放字符常量。

1.3.5 运算符和分隔符(Operators and Punctuators)

在 C 语言中,许多字符有其特殊含义。算术运算符加、减、乘、除、求余符分别由

+ - * / %

表示。5%3 得 2,7%2 得 1。

在程序中,运算符可兼做分隔符。虽然有时在运算符后再加一空格来增加可读性,但对编

译器而言都是一样的。有些符号根据上下文而意思不同,比如在下面两个语句中的 % 符号。

```
printf("%d", a);    和    a = b % 7;
```

前者表示打印格式说明,后者表示求余数。

分隔符包括圆括号、花括号、逗号和分号。在代码

```
int main(void)
{
    int a, b = 2, c = 3;
    a = 17 * (b + c);
```

中,尾随 main 后的圆括号被视为运算符,它告诉编译器 main 是个函数的名字。在其后出现的符号“{”、“,”、“;”、“(”和“)”都是分隔符。

有些特殊字符在程序的不同地方出现,其含义是不同的。以圆括号为例,有时用来表示函数名,有时用来做分隔符。下面例子可说明这个问题。

```
a + b    ++a    a += b
```

在上例三式中都用到了运算符+,但++和+=均为单个运算符。在不同的上下文中,相同字符有不同的解释,这正是精练的 C 语言功能强大的原因。

1.3.6 运算符的优先级(Precendence and Associativity of Operators)

运算符的优先级确定了一个运算式计算顺序。与算术规则一样,先做括号中的子式,括号可以用来改变原有计算顺序。

```
1 + 2 * 3
```

在 C 语言中,运算符 * 比 + 的优先级高,所以先做乘后做加,运算结果等于 7。得同样结果的表达式如下:

```
1 + (2 * 3)
```

但 (1 + 2) * 3 的运算结果与其不同,结果为 9。观察下式:

```
1 + 2 - 3 + 4 - 5
```

因为二元运算符+和-的优先级相同,计算顺序是遵从“从左至右”的规则。下面同样的表达式用括号表示了“从左至右”的规则:

```
((1 + 2) - 3) + 4 - 5
```

表 1.1 中给出了 C 语言中部分运算符的优先级,除了刚刚讨论过的运算符外,还有一些是将要讨论的运算符。

表 1.1 运算符优先级表(I)

运算符	结合顺序
() ++ (后缀) -- (后缀)	从左至右
+ (一元运算符) - (一元运算符) ++ (前缀) -- (前缀)	从右至左
* / %	从左至右
+ -	从左至右
= += -= *= /= 等	从右至左

上表中,同一行的运算符属于同一优先级的运算符。优先级按行递减,即前行比后行的运算优先级高。在每一行的右端标明了结合(associativity)顺序,结合顺序亦即运算顺序。以后每介绍一个新运算符,都将标明其运算优先级和结合顺序。符号“+”用来表示二元加和一元加(正号),负号“-”同样也有二元减和一元减(负)。要注意,只有在标准 C 语言中才有一元加(正),在传统 C 语言中没有一元加(正),仅有一元减(负)。

从上表中可以看出,一元运算比二元运算的优先级高。在

$-a * b - c$

中,第一个“-”是(负号),第二个“-”是(减号)。使用优先级规则,则下式

$((-a) * b) - c$

与前式为同一运算式。

1.3.7 加 1 减 1 运算符(Increment and Decrement Operators)

加 1 运算符“++”和减 1 运算符“--”具有与“+”(正)和“-”(负)相同的运算优先级,它们之间的结合顺序为从右到左。“++”和“--”只能用于变量,不能用于常量和普通表达式。这两个运算符既可用于前缀又可用于后缀,其运算结果根据上下文可能不相同。

$++i$ 和 $i++$

是合法的,但下列两式却是非法的

$777++$ /* 不能用于常量 */

$++(a * b - 1)$ /* 不能用于普通表达式 */

表达式 $++i$ 和 $i++$ 得到同一个运算结果,都使变量 i 加 1。但前者是 i 先加 1,然后 i 以新的值参加其它表达式的运算。而后者是 i 以当前值参加其它表达式运算后,再加 1。请看下列代码:

```
int a, b, c = 0;
```

```
a = ++c;
```

```
b = c++;
```

```
printf("%d %d %d\n", a, b, ++c); /* 打印结果分别是 1 1 3 */
```

同理, $--i$ 使变量 i 先减 1,然后以新的值参加其它运算。 $i--$ 是以当前 i 的值参加运算后, i 再加 1。注意“++”和“--”这两个运算符使内存中变量的值发生改变,而其它所有的运算符均不能改变内存变量的值。例如,表达式 $a + b$ 并不修改变量 a 和 b 的值。

有时,既可以将“++”用做前缀,也可以将其用做后缀,所得的结果是相同的,比如:

$++i$ 和 $i++$

都等效于表达式

$i = i + 1;$

可以把 $++$ 和 $--$ 看成是变量加 1 或减 1 的简写形式。但在某些情况下,一定要注意将其用于前缀和后缀的效果是不同的。