微计算机丛书

C语言程序设计

C语言程序设计

〔美〕斯蒂芬·G·科钦 著

李 颖 齐文陆等 译 王家和 校

电子工业出版社

C语言程序设计

〔美〕斯蒂芬·G·科钦 著
李 颖 齐文陆 等译
王家和 校

电子工业出版社

内 容 简 介

本书译自〔美〕斯蒂芬·G·科钦著《Programming in C》，是目前各版本中最广泛采用的C语言教材。该书详细介绍了C语言及其程序设计，有丰富的可直接上机运行的程序，使初学者很易理解掌握、培养实际操作能力。该书重视程序设计理论的介绍，对C语言的理论问题及细节都作了论述，着重思路清晰、可读性好的程序的编写训练。书后附有C语言的语法规则一览表，对有经验的C语言的工作者也是极有价值的参考书。

本书适于大专院校有关专业师生、从事计算机工作的人员及C语言的初学者参考使用。

〔美〕STEPHEN G. KOCHAN 著
PROGRAMMING IN C
HAYDEN BOOK COMPANY, INC. 1983版
C 语言程序设计
李 颖 齐文陆 等译
王家和 校
责任编辑：路 石
电子工业出版社出版（北京海淀区万寿路）
新华书店北京发行所发行 各地新华书店经售
妙峰山印刷厂印刷
开本：787×1092毫米1/16 印张：19 字数：462千字
1987年12月第一版 1987年12月第一次印刷
印数：1—11000册 定价：4.85元
统一书号：15290·580
ISBN7-5053-0171-3/TP·18

译者的话

C语言作为一种新型的程序设计语言，随着UNIX操作系统的普及而被广泛应用。C语言以它的鲜明的特点和独特的设计而受到程序设计者的普遍重视。目前UNIX操作系统正在我国普及应用，迫切需要一本系统全面的C语言参考书，本书正是满足这一急需而翻译的。

本书译自〔美〕斯蒂芬·G·科钦（Stephen G.Kochan）所著的《Programming in C》一书，该书由美国海登图书公司（Hayden Book Company, Inc.）出版。

本书与其它C语言书籍相比，内容更广泛、系统，条理性强，注重实例。书中的例题程序算法清楚，可读性强，全书连贯、完整，通过阅读这些例题，有助于对概念的理解，这是本书的最大特点。书中内容由浅入深，通俗易懂，每当提出一个新概念时，就给出清楚的定义和充分的说明，并配有一小段完整、恰当的实例程序，不仅便于理解概念，而且对其应用也会有较深的认识。有了这些有趣的实例，使读者步步深入到C语言较复杂与深奥的领域。本书尤其对C语言中容易混淆的局部和全局变量、静态和自动变量的概念讲述清楚，对易产生模糊的结构说明和结构类型说明作了反复解释，概念明确，使读者一开始就建立正确的概念。

本书共有十七章，包括基本概念、循环、数组、结构、指针、位操作、预处理等内容，还有附录，归纳了C语言的语法，对于快速查找语法规则和发现错误是十分有力的工具。

本书非常适合自学，无论是初学者，还是已有一定程序设计基础的人；它也适合作为学生参考书；对于已掌握C语言的读者，本书也是一本极有价值的参考书。

请读者注意，本书程序中使用的双引号为“ 和 ”，在一些计算机关系统中皆为"。请您编程和键入时改为您的系统中的双引号。

本书由李颖、齐文陆、沈立人、郭文芝、宁乃永、耿俊峰等同志参加翻译，郭洪同志为技术顾问，王家和对全书作总校对，齐文陆、李颖作了技术校对。由于校、译者水平有限，书中难免有错误之处，希望广大读者批评指正。

译者　一九八七年三月

目　录

第一章 前 言

C 程序设计语言是贝尔实验室于70年代初期在已开发的程序设计语言的基础上发展起来的。然而直到70年代后期，这种程序设计语言才开始广泛普及。在此之前，C 编译程序还没有进入贝尔实验室之外的商业市场。 UNIX 操作系统的普及也促进了C语言的普及，该操作系统也是贝尔实验室开发的，它是用C语言作为“标准”程序设计语言，有90％以上用C语言编写的。

C语言是一种“比较高级的语言”，然而它也向用户提供了接近硬件和在较低的水平上使用计算机的能力。C语言是一种通用结构程序设计语言，考虑到了系统程序设计的需要，为用户提供了很大的能力和灵活性。实际上，在编程应用中存在着用C语言比较容易，而用其它语言（如Pascal，FORTRAN或BASIC）则很困难的情况。

本书将告诉你如何使用C语言设计程序。它能使有经验的程序设计师和没经验的新手都对此书发生浓厚的兴趣。如果你有设计程序的经验，你会感到，C语言的工作方法独特，与其它语言有很大差别。如果你使用过表面上看与C语言相似的Pascal语言，你将发现C语言具有很多特点，如指针、字符串和位操作等。

C语言的每个特点都将在本教科书中作介绍。每介绍一个新特点时，都要举一个小型的完整程序的实例来说明。这正好反映了该书的写作特点：即注重实例。一张图画比一千字的描述更清楚，选择合适的程序实例也有同样效果。如果你使用的计算机设备支持C 程序设计语言，我们极力鼓励你输入并运行该书中的每个程序，并把系统的输出结果与书中给的输出相比较。这样做，不但可以学会该语言及其语法，还可以熟悉C程序的键入、编译和运行过程。

C语言的教学方法是：先在计算机上提出一个有待解决的问题，然后用C语言开发程序来解决这个问题。这样，新的语言结构便在解决特定问题时被引出。

本书自始至终都着重强调程序的可读性。在写程序时，一定要使作者本人或其它人容易阅读。根据经验和一般认识你将发现，该书中的程序几乎都很容易编写、调试和修改。在程序设计中，如果真的坚持了结构程序设计的原则，那么所开发出来的程序就一定可读。

该书为教学用书，每章的内容都以前面一章节的内容为基础。因此最好是按照章节的顺序逐章阅读，不要打乱章节顺序或跳读。在学习下一章之前一定要做前一章后面的练习。

第二章介绍了高级程序设计语言和程序编译过程的一些基本术语。从第三章开始介绍C语言，在十六章以前把该语言的基本特点介绍完。第十六章对输入输出操作进行比较深入的介绍。该书的最后一章介绍该语言的一些比较先进和深奥的特点。

附录A用作参考资料，它归纳了C语言的语法。另外还附有综合检索，以便快速地找出C语言的特定特征的说明和程序实例。

本书没有指定特定的可执行C语言的计算机系统或操作系统。然而，因为C语言大都

运行于UNIX操作系统，所以对用C语言编写的UNIX操作系统进行了重点介绍。书中介绍了如何在UNIX下编译和执行程序。附录中还列出了UNIX运行时库程序的一览表和cc命令及Lint程序的说明。

如果把C编译程序用于UNIX以外的操作系统，那么计算机设备上的C语言与本教科书中讲的语言之间会有一些微小的不同，因为到写这本书为止，C语言还没有标准的定义。

第二章 基本概念

本章将介绍在学习用C语言设计程序前必须要理解的一些基本术语，并将概述高级语言程序设计的特点，即介绍用这种语言开发的程序的编译过程。

一、 程序设计

计算机是无声的机器，只能按照指令来完成特定的工作。大部分计算机系统都在非常原始的水平上进行操作。例如，大部分计算机可在某个数上加1，也可检测某数是否等于零（这些基本操作的复杂性也就如此而已）。计算机系统的基本操作构成计算机的指令集。有些计算机的指令集非常有限，例如，DEC PDP-8计算机的指令集只有少量基本操作，而DEC VAX 计算机的指令集则由几百个基本操作构成。

为了用计算机解决问题，我们必须用特定计算机的指令来表示问题的解。计算机程序实际上是解决某特定问题所需指令的集合。解决问题的方法叫做算法。例如，当我们打算开发一个检测某数是奇数或偶数的程序时，用以解决该问题的那组语句便是程序。用于检测某数是奇数或偶数的方法即是算法。在一般情况下，当开发一个解决某特定问题的程序时，首先要用算法来表示该问题的解，然后开发实现该算法的程序。解决奇/偶问题的算法可这样说明："首先用2除该数，如果余数是零，该数为偶数，否则该数为奇数。"有了上述算法后便可写入必要的指令来实现该算法。这些指令将由某特定计算机语言的语句表达，如BASIC，Pascal或C。

二、 高级语言

在计算机发展的初期，程序设计的唯一方法是利用二进制数。二进制数与特定机器指令和计算机存储器中的单元直接对应。软件技术的进一步发展便是汇编语言。汇编语言可使程序设计人员在比较高的水平上使用计算机，可使程序设计人员使用符号名称进行各种运算及引用特定的存储单元，而不需要规定二进制数序列来执行特定任务。汇编程序是一种特殊程序，可把汇编语言程序从符号格式翻译成计算机系统的特定机器指令。

因为在每个汇编语言语句和特定的机器指令之间仍存在一一对应的关系，所以汇编语言被看作是低级语言。程序设计人员必须要学会特定计算机系统的指令系统，才能用汇编语言写程序，所产生的程序是不能移植的，即是说，如果不重写该程序，即不能用于其它型号的计算机。不同的计算机系统具有不同的指令系统，汇编语言程序是用这些不同的指令系统写的，这些指令系统都与机器有关。

FORTRAN 是最早的高级语言之一。用 FORTRAN 开发程序时，不需要考虑计算机的体系结构，用 FORTRAN进行的运算比较先进成熟，并且比较高级，与机器的指令系统无关。一条FORTRAN指令(或语句)将产生很多不同的机器指令，这与汇编语言语句

和机器指令之间的一一对应不同。

高级语言语法的标准化，使得用该语言写的程序与机器无关，也就是说，程序可用于任何能支持这种语言的机器，程序有时有一些微小的变化，有时则毫无变化。

为了支持一种高级语言，必须要开发一种特殊的计算机程序，该程序能够把用高级语言开发的程序的语句翻译成计算机能理解的形式，即是说，翻译成计算机的特定指令。这种程序叫做编译程序。

三、 操作系统

我们先介绍称为操作系统的计算机程序的作用。操作系统是一种程序，它控制计算机系统的整个操作。所有的输入/输出通道操作都要通过操作系统来实现。操作系统还要控制计算机系统的资源，并要控制程序的执行。

UNIX操作系统是目前最常用的操作系统之一，是由贝尔实验室研制的。该操作系统是一种特殊的操作系统，可用于很多不同类型的计算机系统。从前，操作系统只用于一种类型的计算机系统。但是，UNIX主要用C语言写，很少考虑计算机的体系结构，所以已能很容易地移置到很多不同的计算机系统中。

四、 编译程序

编译程序是一种程序，其原理与本书中所提到的那些程序完全相同，但要复杂得多。编译程序用于分析由特定计算机语言开发的程序，然后把该程序翻译成在特定计算机系统中执行的适当形式。

图2-1表示用C程序设计语言开发的计算机程序的输入、编译和执行过程。

要编译的程序先被键入到计算机系统的文件中。计算机设备有各种不同命名文件的约定，但在一般情况下，名称的选择取决于操作者。在用UNIX时，如果最后两个字符是.C，C程序可任意命名。在使用UNIX时，C程序的有效文件名称是Program .C。

把C程序输入到文件时，一般需要使用文本编辑程序。在使用UNIX时，通常用ed编辑程序输入程序。为了能够试验一下该书中的程序，你首先必须学会如何使用这种编辑程序。

输入到文件中的程序叫作源程序，它表示用C语言表达的程序的原始形式。当源程序输入到文件后，便可进行编译。

键入特殊的命令便可开始编译过程。输入该命令时，必须要指定含有源程序的文件名。例如，在使用UNIX时，程序编译开始的命令是cc。键入下列命令可开始把含在Program.c中的源程序进行编译。

```
cc  program.c
```

在编译过程的第一步，编译程序先检查源程序中的每个程序语句，以确保程序语句与该语言的语法和语义学一致。这时如果编译程序发现源程序有错误，便通知操作人员，并在此处停止编译过程。源程序中的错误改正后（用文本编辑程序），编译过程才重新开始。在编译过程的这一步，所发现的典型错误可能是语法错误（表达式的圆括号不对称），也

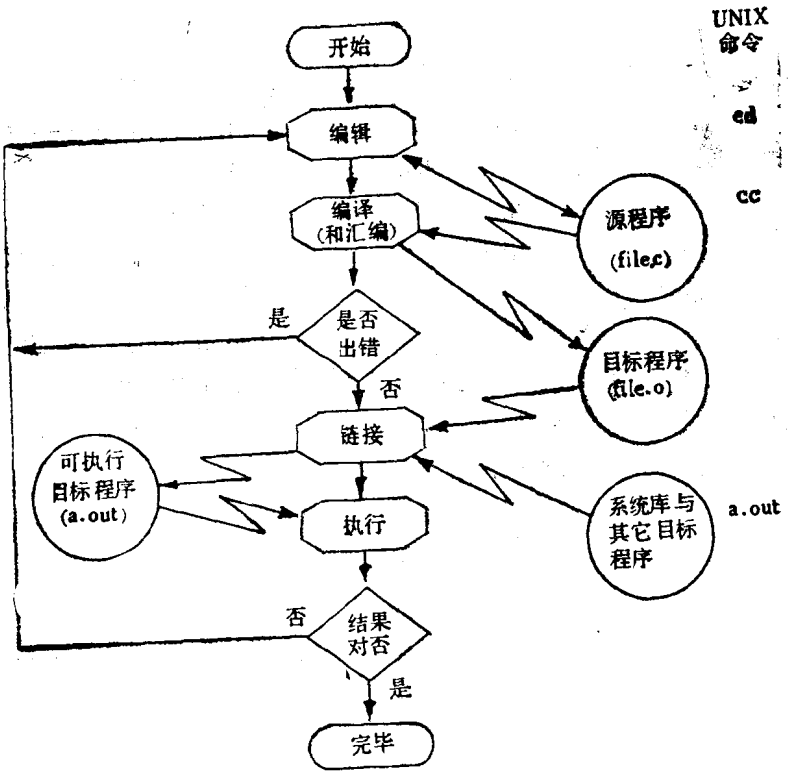

图 2-1 C程序的输入、编译和执行过程
(典型的UNIX命令)

可能是语义错误（变量未定义）。

当消除了程序的语法和语义错误之后，编译程序将把程序的每个语句翻译成“低级”形式。在大部分机器中，这意味着编译程序把每个语句翻译成等效于汇编语言的语句以执行该语句的任务。

把源程序翻译成等效的汇编语言程序之后，再把汇编语言语句翻译成实际的机器指令，这是编译过程的第二步。在第二步中，可能执行汇编程序，也可能不执行汇编程序。在使用UNIX时，当使用cc命令编译程序时，汇编程序便自动执行。如果用其它的操作系统，可能需要你在这时发另一个命令来翻译汇编语言程序。

汇编程序把汇编语言语句转换成目标码（二进制格式），然后写入系统中的另一个文件。该文件的名称与源文件的名称相同，但最后一个字母是“o”而不是“c”。

程序翻译成目标码之后，便可进行连接。在使用UNIX时，发出cc命令后，该过程便自动执行。如果用其它操作系统，可能需要发另一个命令来执行这项任务。“连接”的目的是使程序变成在计算机上被执行的最终形式。如果程序使用由编译程序处理过的另一些程序，那么在这一阶段这些程序将被连接。在这一阶段，从系统程序库来的程序也与目标程序连接。

最终连接后的文件格式是可执行的目标码格式，该文件存储在系统中的另一个文件中，以待执行。在使用UNIX时，该文件以a.out被缺省调用。在执行该程序时，在UNIX操作下，

键入可执行的目标文件的名称即可。下列命令可把调用a.out的程序装入计算机存储器，并开始执行。

```
a.out
```

执行程序时，程序的每个语句都依顺序执行。如果程序向操作人员请求数据（即输入），这时程序将暂停执行，以便输入数据。程序的结果（即输出）将在终端上显示。

如果一切正常，程序将执行其预期的功能。如果程序显示的结果有误，那么必须要从头开始重新分析程序的逻辑，这就是所谓的调试阶段。在这一阶段，要想办法排除所有的程序问题。为了排除程序的所有问题，很可能要对源程序做一些修改。这时，程序的编译、连接和执行的整个过程必须要重复一次，直到获得理想的结果为止。

另外还有一种方法用于分析和执行用高级语言编写开发的程序。用这种方法时，程序不被编译，而是被解释。解释程序同时分析和执行程序的语句。用 BASIC 程序设计语言开发的程序就是被解释的，而不是被编译的。因为用C语言开发的程序是被编译，所以我们不详细介绍解释过程。

第三章　用C编写程序

本章将向读者介绍C语言，使对用C语言编程是怎么回事有一个感性认识。但是要正确评价这种语言，最好的方法是实际看一下用C编写的程序。

首先，让我们挑选一个相当简单的例子——在终端上显示一个短句“Programming is fun.”的一个程序。无需费什么力，下面的一个C程序就能完成这个任务。

例题程序3-1

```
main ( )
  {
    printf ("Programming is fun. \n");
  }
```

在C编程语言中，小写与大写字母是不相同的。在FORTRAN，COBOL，PL/1或BASIC等大多数的编程语言中，是专门使用大写字母的。假如你习惯上述某种语言编程，那么你要有一段时间才能习惯使用小写字母。

C与其它许多编程语言还有一个不同之处是：不特别在乎你键入的那行的起始位置。有一些语言，如FORTRAN和COBOL，在这一点上是十分注意的，而对于C，你可以在该行的任何位置上开始键入你的语句。这一点有助于使开发出的程序更便于阅读。

让我们回到第一个C程序上来，假如我们把这个程序键入了计算机，并发出正确的命令到系统中进行汇编与执行，那么，在终端上会显示下面的结果或输出。

例题程序3-1的结果输出

```
Programming is fun.
```

现在我们仔细地看一下例题程序3-1。该程序的第一行是告诉系统，该程序名为main。main是C系统使用的专用名，它表示该程序从这里开始执行。

紧跟在main后面的一对圆括号规定了这个程序没有期望的“变元”或参数。（关于这个概念，在第八章讨论函数时将要详细解释。）

现在，我们已经告诉系统我们的程序名是maim，而且我们已准备并指定了这个程序要执行的功能，这就是用一对花括号括起来的全部程序语句。（对于熟悉Pascal的读者来讲，C中的花括号有点类似于Paseal语言中的BEGIN和END程序块说明符。）该系统把花括号中的全部程序语句作为main程序的一部分。在例题程序3-1中，我们只有一句这样的语句。这个语句表明要调用printf程序。要传递到printf程序的参数或变元就是字符串

“Programming is fun.\n”

printf程序是C系统的一个函数，它只是在终端上或在CRT屏幕上打印出或显示出一个变元（或多个变元，你马上就会看到）。把字符串中最后的两个字符，即反斜线\和字母n合起来作为newline（换行）字符。newline字符告诉C系统所要做的事就是“进入新的一行”。在newline字符后的任何字符，在终端上或显示屏幕上都出现在下一行中。实际上，newline字符类似于打字机上的回车键。

C的所有程序语句必须用分号(；)结尾。所以分号紧跟在printf调用的闭圆括号后面出现。

我们已把例题程序3-1讨论完毕.现在我们要让它增显出"And programming in C is even more fun.", 要做到这点，只要简单地再加上一个如下面所示的printf程序调用，记住，每一个C程序语句必须用分号结尾。

例题程序3-2

```
main ( )
  {
   printf ("Programming is fun.\n");
   printf ("And programming in C is even more fun.\n");
  }
```

假如我们键入程序3-2，并对它进行编译和执行，我们预期在终端上得到下列输出。

例题程序3-2结果输出

```
Programming is fun.
And programming in C is even more fun.
```

正如你从下面的程序例子中可看到的那样，对于每行输出不必都单独调用printf程序。请研究下面列出的程序，并在验证输出结果之前先试着预料一下它的结果如何。

例题程序3-3

```
main ( )
  {
   printf("Testing...\n..1\n...2\n.... 3\n");
  }
```

例题程序3-3结果输出

```
Testing...
..1
...2
.... 3
```

在本书中，printf程序是最通用的程序。这是因为它显示程序的结果容易方便。它不仅能显示简单的短句，还能显示变量值和计算结果。下一个程序就是使用printf程序显示两个数50和25的相加结果。

例题程序3-4

```
main ( )
  {
   int  sum;

   sum = 50 + 25;
   printf ("The sum of 50 and 25 is %d\n", sum);
  }
```

例题程序3-4结果输出

The sum of 50 and 25 is 75

对上述程序作一点注释是有好处的。第一个C程序语句说明变量sum是属于整型的。C与Pascal类似，而与FORTRAN或BASIC不同，它要求一个程序在使用变量之前，对所有程序中的变量都要加以说明。一个变量的说明给C编译程序规定了该程序将要使用的是怎样的特定变量。编译程序需要这种信息，以便产生一个正确的指令去存取相应的变量。被说明为int(整)型的变量只能容纳整数值，即数值不带小数位，例如3，5，-20和0。有小数位的数，象3.14，2.455和27.0称为浮点(floating point)数。

整型变量sum用于存储50和25两个整数的相加结果。我们在变量说明行下面有意地留了一行空白行。这么做是为了看程序时把程序变量说明与程序语句分开，这是一种严谨的作风。有时在程序中加上一行空白行，更便于阅读程序。

程序语句

```
sum = 50 + 25;
```

与大多数其它编程语言相同，把它读作：数值50加上（用正号表示）数值25，其结果存储（由赋值运算符，即用等号表示）到变量sum中。

在程序3-4中，printf程序调用中的圆括号内有两个参数项，或称为两个变元。这两个变元用逗号分开。对于printf程序，第一个变元总是所要显示的字符串。在显示字符串的同时，我们还可能常常想要显示某些程序变量的值。在我们的实例中，我们希望在终端上，在下面的字符串

The sum of 50 and 25 is

之后显示出变量sum的值。在第一个变元中的百分数字符是一个能被printf函数识别的专用字符。紧跟在百分数符号后面的那个字符，说明在该点上将显示的数值是什么类型的。在上面的程序中，系统识别出字母"d"，这表明要显示的值是一个整数值。

只要printf程序在字符串中发现了%d字符，即自动显示printf程序的下一个变元的值。因为这里printf下一个变元是sum，在显示出字符"The sum of 50 and 25 is"以后，自动显示sum的值。

现在我们再试猜一下以下程序的结果输出。

例题程序3-5

```
main ( )
  {
    int  value1, value2, sum;

    value1 = 50;
    value2 = 25;
    sum = value1 + value2;
    printf ("The sum of %d and  %d is %d\n", value1, value2,  sum);
  }
```

例题程序3-5结果输出

The sum of 50 and 25 is 75

第一条程序语句说明所用的三个变量value1，value2和sum都是int型。也可以用三个

单独的说明语句表示这个语句：

```
int  value1;
int  value2;
int  sum;
```

说明了三个变量后，程序把数值50赋值给变量value1，又把25赋值给value2。然后计算这两个变量的和，并把结果赋值给变量sum。

现在所调用的printf程序包含四个变元。再重复一遍，第一个变元通常称为格式串(format string)，它告诉系统其余的变元将如何显示。显示字符“The sum of”后，接着要显示value1值。同样，格式串中后两个%d字符出现的位置，表示是要显示value 2和sum的值。

在本章的最后一个程序中，我们要引入注释（comment）的概念。程序中使用注释语句的目的是把程序做成一个文件，并提高程序的可读性。如同你在下面实例中所看到的，注释用来告诉程序的读者——这些读者可能是编程员本人或是作维护程序工作的人——编程员写一个程序或一系列特定语句时，他（或她）的脑中想的是什么。

例题程序3-6

```
/* This progran adds two integer values and displays
   the results                                        */

main ( )
  {
    /* Declare variables */
    int  value1, value2, sum;

    /* Assign values and compute the result */
    value1 = 50;
    value2 = 25;
    sum = value1 + value2;

    /* Display the results */
    printf ("The sum of %d and %d is %d\n", value1, value2, sum);
  }
```

例题程序3-6结果输出

```
The sum of 50 and 25 is 75
```

注释语句用/和*两个字符开始，在这两个字符之间一定不能留空格。用*和/字符作为注释语句的结尾，同样中间不能插入空格。在/*和*/之间的所有字符都作为注释语句对待，并为C系统所忽略。在例题程序3-6中，使用了四个独立的注释语句。而这个程序实际上与程序3-5是相同的。对于即使这么简单的实例，把两个程序作对比后，还是能确认：加上注释语句后极大提高了程序的可读性，有助于弄清程序的功能与操作。

强调在程序中使用大量的注释语句是不过分的。许多时候编程员回头看看也许是6个月