C语言及其程序设计

孙玉方　孟庆昌

《C语言》编辑部

目　录

前　言

随着UNIX分时系统在国际上的广泛流行，近年来C程序设计语言在软件工程领域里颇为引人注目。

C语言最早是用来描述UNIX操作系统及其上层软件的，但目前它的应用已不限于UNIX系统，而在多个操作系统和多种计算机硬件上实现了。实现它的机种从8位微型机到最高档的超高速机Cray－1，遍及微、小、中、大型机。尽管它在精确性和严密性上尚不如某些语言（如Pascal、Algol 68），但由于它具有简洁、实用、代码质量高，特别是可移植性好等特点，所以受到计算机用户特别是软件工程技术人员的赞赏，并得到越来越广泛的应用。

C语言的发展与UNIX一样，经历了产生、完善和广泛应用的过程。它原是一个实验室的科研产品，只是在八十年代才随UNIX一起进入商业市场。外界对该语言的研究也是近几年才开始，再加上C语言不是按BNF严格定义的，所以有关C语言方面的专著甚少。国内到目前为止（除了翻译过来的《C程序设计语言》一书外），还没有看到一本比较适合于学校教学用的C语言程序设计的教材。而国内在UNIX系统上的研究、应用及开发工作正在蓬勃开展，对C语言的学习掌握是迫在眉睫的事情。

这些年我们在学习、研究UNIX系统的同时，在学校和有关讲习班里多次讲述了UNIX核心结构、UNIX操作系统原理及UNIX使用等。深感熟悉C语言对研究和开发UNIX的重要性。为此我们曾在讲习班上或多或少地多次介绍过C语言，但是手上仍缺乏一本较系统完整的教材。为了更好地提高教学质量，帮助广大用户更好地使用和开发UNIX系统，我们参照有关资料，结合自己讲授和使用C语言的体会，把手稿加以整理，形成了本初稿。初稿由北京信息工程学院和有关讲习班采用。根据教学实践和学员们提出的意见，我们对初稿进行了全面修改，特别是后几章，加进了许多更为实用的例子，加强了说理，试图对指针、结构等C语言中的难点作更为深入、系统的剖析和归纳，以期使本书更为完善。

本书共分七章，后面有若干个附录。

第一章简单叙述了C语言的发展历史、特点，并且结合简单例子对C语言的编写、编译和运行全过程作了概括地介绍，目的是让读者熟悉C语言运行环境，以便在后面的学习中通过上机学习来加深对C语言的了解。

第二章介绍了C语言的基本概念和成分：标识符、变量、常量、数据、及变量的简单类型。表达式是语言的基本组成部分，在本章中对各种主要表达式都作了介绍。在表达式基础上介绍了一种最简单和基本的语句，即赋值语句。最后介绍了C语言众多的运算符，给出其优先顺序。

第三章介绍C语言主要的语句和控制流：条件语句、开关语句、各种循环语句以及各种转移语句。

第四章介绍构成C程序的主要成份——函数。围绕函数介绍了函数结构、函数的返回类型、参数和递归调用。联系到函数，我们介绍了C语言特有的变量存储类。最后介绍了C的程序结构和预处理程序。

除了第二章介绍的几种基本数据类型外，第五章和第六章专门介绍几种复杂的数据类型。

第五章介绍指针和数组，主要包括它们的定义、使用，它们之间的关系及它们与别的语言成份（函数、其它类型的数据）之间的关系。

第六章重点介绍结构和联合。这包括结构的组成、其成份的访问，结构数组与指针，引用自身的结构及其所构成的多种数据结构、联合的定义、联合与结构之间的不同点等。

最后一章介绍的内容虽然不是C语言的成份，但却是使用C语言完成工作所必需的，主要包括输入输出以及C语言使用与UNIX系统（主要是文件系统）的关系。

后面的附录列出了C语言作者Ritchie所给出《C语言参考手册》。我们还给出了C语言语法的BNF表示和完整的语法图表示，一部分编写C语言程序所用到的标准函数和系统调用，C编译程序在若干微机上的实现，以及对C语言的若干评价意见。最后列出了本书编写时所参考的主要资料。

在编写本书时，我们用语法图方式对C语言的语法成份进行描述。这种描述方式比较直观、形象，也比较严格。但是由于C语言至今还没有给出一个形式化的描述，所以我们这种描述方法还是一种尝试。附录B中我们结合有关资料和个人的研究工作对C语言语法用BNF进行了描述。这是一种更为形式化和更严格的方法。但是我们还没有用这种BNF法来构造一个C编译程序的经验，即这种描述方法还没有经过考验，因此，还是不成熟的。我们是想借此向大家作一介绍，抛砖引玉，以期引起大家的讨论。

本书编写过程中，我们注意说理与实例并重，以实例作为说理的基础，促使读者对一些概念加深直观理解，而说理又把实例中的现象加以归纳和提高。但本文总的是从实用角度来讲述C语言的。由于各种条件限制，有些说理部分可能不很严格。我们认为学习语言的最好方法是实践，所以本书中的实例许多都是完整的，可以直接上机运行。我们还给出了许多习题，希望广大读者尽量多做习题、多上机实习，通过练习来巩固自己所学的知识，也为以后用C语言开展工作打下基础。

由于我们的经验和水平有限，疏漏之处难免。另外，语法图的描述方法对于C语言是否合适，我们还只是作了一次尝试，很可能存在许多问题。我们恳切地希望广大读者和有关专家提出批评建议，以便我们以后再行修改。

第一章 C语言概述

1.1 C语言历史和特点

1.1.1 C语言的演变历史：C是UNIX系统的主力语言，它与UNIX系统有着互相依存、休戚与共的紧密关系。UNIX系统是美国贝尔实验室的K.Thompson和D.M.Ritchie从1969年开始，用不到两个人年的时间研制成功的。该系统最早运行在DEC的PDP-7上，第一版本是在GE635机上产生并通过纸带把可执行代码传送到PDP-7上的。在UNIX上实现了汇编语言后，UNIX系统又以汇编语言编写。由于汇编语言的不可移植，以及描述问题的效率不如高级语言，特别是可读性差，所以K.Thompson决定开发一种高级语言来描述UNIX系统，1971年K.Thompson在PDP-11/20上实现了B语言并用B写了UNIX操作系统和绝大多数实用程序。B语言主要思想源于BCPL语言。BCPL是M.Richards基于CPL语言在1969年发表的一种语言，它是一种单一数据型语言，至今它仍显示出足够的生命力。不过，由于下列几点原因，B没有盛行起来：第一，PDP-11是字节编址的，而B却面向字，这样就妨害了对单个字节进行存取的能力。第二，现代的高级语言都要求有强有力的类型结构，而BCPL和B语言无类型性(或者说只是一种机器字类型)在描述客观世界许多事物时会遇到相当多困难。Algol-68和Pascal语言是强类型型语言，它们（特别是后者）有丰富的数据类型，这是其最大的优点之一。第三，B编译程序产生的是解释执行的代码，运行速度比较慢。当然还有一些次要的原因。总之，C语言的开发已是势在必行了。D.M.Ritchie从1971年开始，用了一年时间写了第一个C语言编译程序。1972年C便投入了使用。

1973年K.Thompson和D.M.Ritchie把UNIX系统用C重写了一遍。系统的代码量比以前的版本大了三分之一，加进了多道程序设计功能，特别是整个系统（包括C语言编译程序本身）建立在C语言基础上，而C语言又具有良好的可移植性，所以第五版的UNIX系统（UNIX V_5）就奠定了UNIX系统的基础。以后UNIX V_6，V_7，System Ⅲ和最新的System V都是在V_5基础上发展、扩充的。

今天UNIX系统已在全世界范围内取得了巨大的成功，它几乎成了16位微型机的标准操作系统，在PDP-11小型机，VAX-11超级小型机，甚至象IBM370，4300，UNIVAC，Honeywell等大型机上也有它的踪迹。从某种角度可以说，没有C语言就没有UNIX今天的巨大成功。当然，没有UNIX，C语言也不会有今天。

虽然最初的C语言是附属于UNIX系统且在PDP-11上实现的，但是目前C语言却独立于UNIX系统，独立于PDP-11机而蓬勃发展，它适应的机种从8位微型机（比如以Z80为CPU的Cromemco）到Cray-1巨型机。它附着的操作系统可从8位微型机上的单用户CP/M直到大型机的IBM VS/370。它与FORTRAN、Pascal等语言一样已经成了微、小、超小、大、超大和巨型机上共同使用的语言。

人们喜欢用Pascal与C比较，其实它们主要都是Algol语言系统的发展，并各具特点。图1-1列出了Algol语言属系的一些主要演变情况。

1.1.2 C语言特点：C语言的特点是多方面的，人们从不同的角度总结出众多特点。根

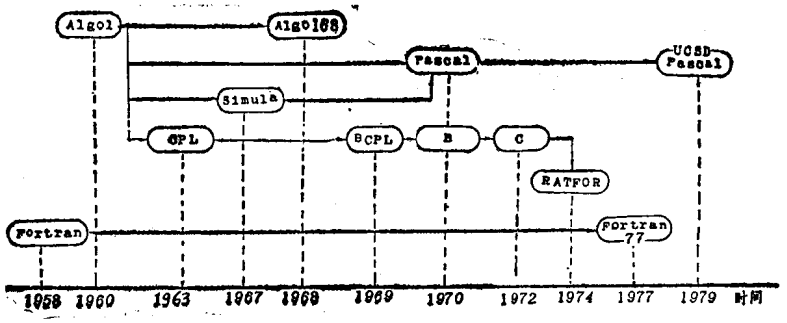


图 1-1 几种主要语言的演变

据我们的体会并参照流行说法大致归结有以下几点：

1．语言表达能力强，它可以直接处理字符、数字、地址，可以完成通常要由硬件来实现的普通的算术及逻辑运算。它反映了当前计算机的性能，所以有效到足以取代汇编语言来编写各种系统软件和应用软件。最明显的例证是UNIX系统。UNIX 系统主要分三层：核心（操作系统）、与外层的接口shell命令解释程序以及外层大量的子系统，包括各种软件工具，实用程序和应用软件。这些程序中除了核心内部的1000行左右（整个核心的10%以下）因为效率、机器硬件表达需要用汇编语言写外，其余的都是用C语言描述的。当然C同样可以表达数值处理，字处理功能，所以它又是一种通用语言。

2．具有数据类型构造能力，并具有很强的控制流结构。它可以在基本类型（如：字符、整数、浮点数等）的基础上按层次方法逐层构造各种构造类型（如：数组、指针、结构和联合等），所以它的数据类型较为丰富。另一方面它的各种控制语句如if, while, do while, switch 等，功能很强，足以描述结构良好的程序。此外，它的有些存储类（如静态 static，外部extern）在功能上有助于数据隐藏的模块化结构程序设计。

3．语言本身简洁，编译程序小。C语言除了在表示法上尽可能简洁(比如，以{ 、} 代替通常的 begin、end 做复合语句括号，运算符尽量缩写等）以外，语言的许多成分，都通过显式函数调用来完成，比如C语言中没有I/O设施，也没有并行操作、同步或协同程序等等复杂控制。另一方面，它在运行时所需要的支持少，占用的存储空间也小。

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，特别象操作系统，编译程序等，除了语言表达能力以外还有一个重要因素是该语言的代码质量。如果代码质量低，系统开销会增大，实际上不可行，所以直到现在汇编语言仍是编写系统软件的主要工具。许多高级语言相对汇编语言而言其代码质量要低得多，但C语言则不然。许多试验表明，针对同一个问题，用C语言描述，其代码效率只比汇编语言低10—20%。而由于用高级语言描述比汇编语言描述问题编程迅速，可读性好，特别是C语言还比较容易移植，而其效率降低不大，所以C就成了人们描述系统软件和应用软件比较理想的工具。在代码效率方面的确可以和汇编语言媲美。

5．可移植性较好。这是指程序可以从一个环境不加或稍加改动就可搬到另一个完全不同的环境上运行。汇编语言因为依赖于机器硬件，所以根本不可移值，而一些高级语言，如Fortran等的编译程序也不可移植。目前许多不同机器上几乎都配有Fortran，但这基本上都是根据国际标准重新实现的。

而C语言目前在许多机器上出现，大分部却是由C语言编译移植得到的。统计资料表明，不同机器上的C编译程序80%的代码是公共的。C语言的编译程序便于移植，也就使一个环境上用C编写的许多程序可以很方便地移到另一个环境上。

C语言的优点很多，但也有一些不足和缺点，虽然有些已得到了弥补。这些不足和缺点包括：运算符优先级太多，不便于记忆，有些还与常规约定有所不同；类型检验太弱，转换比较随便，所以不太安全（当然，目前有一个预处理程序lint，它主要用来检验类型协调匹配，帮助解决移植中的一些问题）。尽管如此，C语言仍不失为一个实用的通用程序设计语言，特别是一种强有力的系统程序设计语言。我们说它实用，是指它表达能力很强。C语言在理论研究方面可能不如Pascal、Algo168等语言，但从出产品的实用角度来看，C要比它们强有力，这与Algo160和Fortran之间的关系有相似之处。由于它上述的几个突出优点而使人们对它倾注越来越多的关心，以至在世界上使用、研究C语言的人数正以爆炸般的方式迅猛扩大。

1.2 C语言的一般介绍

本节以若干小型例子给出C程序的一些概貌，使读者对C程序有一个初步的印象，便于后面几章深入地开展对于C语言及用C进行程序设计的讨论。另一方面，对于C语言程序编制、编译、运行的过程也作一概括介绍。本来程序的编制、编译、运行等应是使用系统，如UNIX，而不是C语言本身讨论的课题，但是鉴于许多读者对C语言还感到生疏，所以在这里作些简略介绍可能有助于读者在阅读本书的同时，上机进行实际练习。我们坚信学习新语言的唯一途径是用此语言编写程序并且获得预定结果。

1.2.1 词汇表与语法图： 任何一种高级语言，都需要有自己的基本词汇表，C语言也不例外，其基本词汇表由下列几部分构成：

数字：0，1，2，…，9；英文字母：A、B、C、…、Z，a、b、c、…、z；由于UNIX系统喜欢小写字母，在内部处理时往往以小写字母为主，所以只有大写字母的终端在使用UNIX时会遇到困难。

下线字符：—，这个下线符起一个英文字母的作用，所以在构成标识符等语法成分时，以下线“—”打头，在C中是合法的，而在通常的语言中是不允许的。

特殊符号，主要包括下列一些运算符和关键字：

运算符有：+、-、*、/、%(取余)、=(赋值)、<、>、<=、>=、!=(不等于)、==(全等比较)、<<(左移)、>>(右移)、&(逻辑位与)、|(逻辑位或)、&&(合取)、||(析取)、^(逻辑位异或)、～(按位求反)、(、)、[、]、→(指向)、。(成员)、；(语句结束符)、！(反)、？：(三元运算符)

关键字有：int、char、float、double、struct、union、long、short、unsigned、auto、extern、register、static、typedef、goto、return、sizeof、break、continue、if、else、do、while、switch、case、default、entry

关键字entry目前还未由哪一个编译程序实现，但保留给将来使用。在某些具体实现中还保留了如下两个字：fortran和asm。

下面几个字虽然不严格地属于关键字，但建议读者把它们看成关键字，而不要在程序中随意使用，以免造成混淆，这些字是：define、undef、include、ifdef、ifndef、endif及line。这些字主要用在C语言的预处理程序中。

在C语言中，关键字都有固定含义，不能作为一般标识符使用。

如同自然语言及编织、音乐等活动中的专用语言一样，C语言也有它特定的语法规定。利用基本词汇表中的一些符号和关键字，按照给定的语法规则，就可以进一步构造其它符号、语句和程序。在程序中不允许出现上述词汇表中没有的符号，比如，不允许将两数相乘的乘号＊写成×，也不允许出现违反语法规则所构成的符号。另外，在C语言中有些符号在不同的上下文中具有不同的含义，比如＊，在变量的类型说明中，可能解释成“某某变量是一个指针”，即＊是指针类型的标志；但在赋值语句赋值号的右边，它又表示“把某某变量的内容赋给左边的量”，即＊又是“变量内容”的标志。这些地方需要特别小心，必要时我们会时常提醒读者注意。

编写出的程序是否合法，完全取决于它们是否遵守给定的语法规则。通常，表示语法规则的形式化方法有两种：即BNF (Backus-Naur Form) 和语法图。BNF首先用于描述Algol 60的语法规则，沿用至今已有20多年的历史。这种方法的最大优点是清晰、严谨，在语言形式化描述，编译程序自动生成等研究领域中它是一种最有效的工具。语法图最早盛行于描述Pascal的语法规则，至今也已有10年的历史。这种方法的最大长处是直观形象，在语言数学中是一个得力的工具。本书主要是为了讲述C语言，而不是研究C语言，所以采用了语法图形式，在附录B中我们给出C语言语法规则的完整的BNF表示，以便读者参考。要说明的是，书中所述无论是语法图还是BNF表示，都是我们根据自己的学习体会并参照有关资料编制而成的，在某些方面可能有疏漏和不严格之处，望读者注意。下面我们给出若干C语法成份的语法图。

[例1.1] 在C语言中无符号整数 unsigned int（或直接写成unsigned）的语法图如图1-2所示。

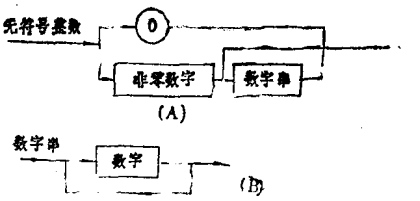


图 1-2 C语言无符号整数语法图

a. 无符号整数语法图，

b. 数字串语法图

方框中的数字是指词汇表中的一位数字，沿着箭头的方向，通过语法图的一条路线就定义了unsigned的语法规则。由此可知：C语言中的一个无符号整数至少包含一位数字，它是以某一数字开头，后面接着一串（包括空串）数字如：5，67，2057，32765。这些都是C语言中合法的无符号整数。

[例1.2] 在C中，十进制整数的语法图如图1-3所示。

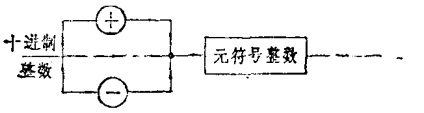


图 1-3 十进制整数

十进制整数前可冠以正负号(＋或－)，正负号缺省时就认为该数是正数。

C中的十进制整数int是机器字长的自然表示，在16位机（PDP-11系列，16位微机）上它的取值范围是－32768～＋32767，在32位机(IBM370，Interdata 8/32、VAX-11系列）上其取值范围是－2147483648～＋2147483647。在C中合法的十进制整数如：

4096，＋10，－100，－32767，0等。

[例1.3] 在语言中，常需要对一些变量、常量、程序等对象命名，一般均采用标识符来代表它们的名字。在C语言中，构造一个标识符的语法图如图1-4所示。

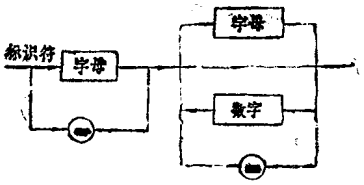


图 1-4 标识符语法图

字母是指英文26个小写和26个大写字母。C与一般语言标识符不同之处是，下线字符“—”也起一个字母的作用，所以可作标识符打头字符，后跟字母或数字串（包括空串和下线符），所以合法标识符是：

xyz, text, sp15, _global, _NFILE等。在C语言中，一般喜欢以小写字母作为标识符，以下线字符打头的标识符一般在内部使用，以示区别。所以用户一般不要以下线符作打头字符。另外，标识符最好起得有意义，便于联想和记忆。

从上述几个简单的语法图例可初步知道如何用语法图来描述相应的语法规则。在语法图中，从箭头的入口到出口，按箭头的指向有一条以上的路线。按每条路线所定义的符号，语句或程序都是合法的。如果读者没有使用一般计算机语言的知识，那么在开始学习用C语言编写程序时，应经常参照语法图或BNF，以检查所编写的程序是否符合语法规则。如果读者有使用计算机语言的基本知识，那么重点应放在C与别的语言的不同点上，通过类比就可以较快地掌握C语言的使用方法。

1.2.2 C语言程序结构

(1) C程序和主函数

一个计算机的程序主要由两部分内容组成。一部分是关于程序所要实现的算法的描述，这种描述一般由一系列语句组成，而这些语句又可能按描述对象的要求以及语言本身的规则构成复合语句、分程序或函数、过程等。另外一部分是关于这些算法所要操作的对象（通常用数据来表示）的描述。在程序中出现的数据，或者是常量或者是变量。对数据的描述就是对程序中所用到的常量和变量进行相应说明，特别是对变量要进行类型说明。下面举几个简单例子，说明如何用C语言来编写程序以及C程序的基本结构，使读者对它有一个直观的了解。应注意，与BASIC等语言不同，C语言中是不允许有行号的。

[例1.4]

```
/*small.c The smallest C program*/
 main (    ) /*There is no executable code
 */
{
}
```

C程序一般是由一个或多个函数组成，这些函数可驻留在一个或几个源文件中，这些文件都以.c结尾，比如本例中，C语言程序只有一个函数，并且驻留在一个文件small.c中。组成一个程序的若干函数中必须有一个且只能有一个名为main的函数，可运行的C程序总是从main开始执行。一个函数在其名字之后一定要有一对圆括号“(”和“)”，这是函数的标志。圆括号中放参数，参数可有可无，根据具体情况而定，如本例中就没有参数。

程序中以“/*”开头到“*/”结尾所表示的意义是一个注释，注释帮助读者阅读和理解程序，但在编译时，注释行是要忽略掉的，即，它不产生代码行。注释在各种语言中都是希望有的，甚至是越多越好。注释行可以在程序的开头，这一般说明整个程序的功能和注意事项；也可以插在程序语句行中或在某行语句的尾部，主要用来说明一段程序的功能或某一行程序的作用。在本例中，程序开头及第一行后都有注释。要注意，C语言中注释不能嵌套。

在例1.4中第2，3行实际上是一对花括号，在这里花括号可以看成程序体括号，类似于Pascal等语言中的“begin”和“end”，这里不过是一种简略表示，它也可以用来括

起任何一组语句，从而构成一个有意义的复合语句或分程序。花括号中可以有任何意义的语句，包括空语句在内。要注意在一个函数中至少要有一对花括号，也就是程序体括号，而不管程序体是否为空(就如本例那样)。本例是一个有意义、正确的C程序，实际上它什么也没有干。

上述程序也可以写成一行

[例1.5]

```
/ *small2.c The smallest C program,on one line */
main ( ) { }
```

如例1．4和例1．5所示，C的一般函数或“main ( )”之后及最后的闭花括“}”之后都没有通常的“;”或“.”之类语法符号。C语言程序结构如图1-5所示。

图 1-5　C程序结构语法图

C程序包括程序模块或包含(include)模块。在图1-5中上方的路线代表的就是程序模块，而下方的路线代表的就是包含模块。在程序模块中，外部说明部分主要包括了函数说明和数据说明；外部定义部分主要包括了函数定义。外部说明和外部定义我们在以后将详细介绍。这里简单讲述一下程序模块中的主函数。在一个完整的C程序中必须有一个主函数。主函数的语法图如图1-6所示。

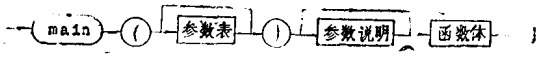


图 1-6　主函数语法图

主函数以特定的标识符(函数名) main为标记，后有一对圆括号，最后是函数体。在圆括号中参数可有可无，与之对应参数的说明也可有可无。函数体的语法图如图1-7所示。

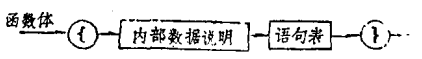


图 1-7　C函数体语法图

其中，内部数据说明主要包括数据说明(以后详述)；而语句表则是一组语句，包括空语句和空在内。注意，在C语言中空语句和空有根本区别。

由图1-5，1-6和1-7可得到一个最简单的C程序，它无参数，函数体为空，这种程序如例1．4或例1．5所示。

前已说过，一个C程序可由若干函数(这些函数可在同一个或多个文件中)组成(其中，最少要有一个函数main)。在C语言程序设计中，鼓励你养成良好的设计风格，即用多个小函数来构成一个大程序。一个训练有素的C程序员应该把大程序分裂成小型并各自完成单独一个功能的若干个程序。如果每个功能都能独立，那么程序员可以把这些功能都看成“零件”，需要时可以用这些“零件”组成各种“大机器”(程序)，而不必每次都重新构造大型程序。在下面例子中main调用一个函数doit，它什么也不做。这不过是主程序调用另一个函数(子程序)的一个最为简单的例子。

[例1．6]

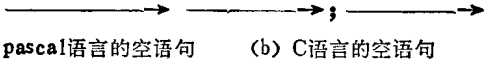
```
/ * smallsub.c smallest C program with a sub-
program * /
    main ( )
    {
```

```
    doit(    );
}
/* doit doesn't do anything */
doit (    )
{
}
```

在这个例子中，main主函数有一个可执行语句——调用语句“doit ( )；”。虽然doit ( ) 函数什么也不干，但是它在执行后要把控制返回主函数main。

函数调用语句通过列出被调用函数的名字并给出实际参数来表示。本例中实际参数为空。另外，C语言中在许多场合下对一般过程和函数过程不加区分，统称为函数。如果调用函数只要求控制返回而不要返回值（如本例所示），那么就可以把它看成是通常语言中的一般过程。而如果调用语句出现在赋值号（=）右边，那么此时不仅要求被调用程序返回控制而且要返回值，这样我们也可以把它看成为函数过程。对某些情况我们以后还会着重介绍，以使大家加深认识。

可执行语句（如本例中的调用语句）之后的分号“；”在C中是作为语句终结符（如在PL/1中那样）而不单单是语句分隔符（象在pascal和其它类Algol语句中那样）存在的，换句话说，分号必须在每一个合法的可执行的C语句后出现。所以既然不是语句分隔符，在“main ( )”之后就没有必要带“；”，反之它既然作为语句终结符，所以在第3行语句doit ( ) 之后要有“；”，尽管这个语句之后就是“}”。而在pascal程序中最后的那个语句与“end”之间的“；”都要省掉，如果有了，则可看成在这个语句和“end”之间还有一个空语句。从pascal和C语言的空语句语法图中我们也可以看出分号的作用，见图1-8。



(a) pascal语言的空语句　　(b) C语言的空语句

图 1-8　两种语言中分号的不同作用

在C语句中空语句和空（如例1．4中的{　　}中就为空）是有区别的。

最后要说明一点，C语句中构成一个程序的若干函数，其次序是无关紧要的，当然在一般情况下，把main函数放在最前面，这主要是有助于阅读和理解，编译程序并没有对次序作任何规定。

至此，我们已经定义了C语言函数，其语法图表示如下（图1-9）：

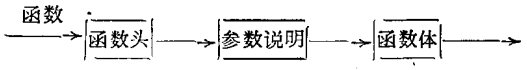


图 1-9　C函数定义语法图

C语言的函数由函数头，参数说明和函数体三部分组成。函数头主要说明了函数返回值的类型及参数；参数说明主要包括函数参数（区分）说明；而函数体中主要包括内部数据说明和语句表，其语法图已如图1-7所示。有关函数定义我们在第四章中还要详谈。

基本函数如例1．6中的main（主函数）或doit（main函数的子程序函数）。这里要说明以下几点。

花括号中通常是由说明和语句组成的分程序，但在简单的情况下，可以什么也没有，如例1.4所示；也可以只有一个语句，如例1.6的main，复杂的情况下也可以带有说明和多种语句，下面例1.7是复杂情况中的一个简单例子。该程序要做的工作是把三个整数相加并且打印它的和。

[例1.7]

```
/* sum.c, The sum of a, b, c */
main (    )
{
    int a, b, c, Sum;
    a=1;
    b=2;
    c=3;
    sum=a+b+c;
    printf ("sum is%d", sum);
}
```

其中，第4,5,6三个赋值语句行可以合写成

一行：

```
a = 1; b = 2; c = 3;
```

只要写得下，并且表达清晰就行。

第3行即为说明语句，它说明a、b、c、sum四个变量都是整数，关于类型说明我们以后还会详谈。

第8行printf是一个很复杂的能产生格式输出的函数，或者称为通用格式转换函数。这里我们仅讨论它的一些简单用法。

printf第一个参数是格式信息，在"%"之前是要直接印出的字符串。每个"%"都指示其它（第二个，第三个，……）参数要被替换成值并指出用什么格式来印出它。所以，其它参数即是要输出的变量。

[例1.8]

```
/*hello.c greet the world, introduce
output in C*/
main (    )
{
   printf ( "Hello, world! \n" );
}
```

这是printf的一种最简单的使用，字符串"Hello, world!"被打印出来，而\n是换行符只产生动作，不会打印出来的。例1.9产生同样结果。

[例1.9]

```
/*Stream.c print "Hello,world" as stream
        output*/
main(    )
{
printf( "\t" ),
printf ( "Hello" ) ;
printf ( ( ", " ) ;
printf ( "  " ) ;
printf ( "world" ) ;
printf ( "! " ) ;
printf ( "\n" ) ;
}
```

其它\t是制表跳格（一般为8格），"\n"是换行符。例1.9中因为只有最后一行有"\n"所以前面的输出都在一行上，从而产生与例1.8相同的结果。而在例1.7中，如果sum是6，则输出将是sum is 6,这是一个比例1.8，例1.9复杂的例子。因为printf ("sum is %d \n", sum)；中，第一个参数里含有"%d"，这意味着参数表中下一个参数（即变量sum）将作为一个基数为10的数被打印出来。而"%d"之前的"sum is"作为字符串原封不动地打印出来，从而形成上面的输出形式，下面列出常用的一些打印格式（详见7.1.2）。

表 1-1 常用的一些打印格式

表示	功能	例子	输出
%d	十进制整数	printf("…%d\n" ,sum);	…6(设sum = 6)
%f	十进制浮点数	printf("…%f\n" ,sum);	…145.7(设sum = 145.7)
%c	单个字符	printf("…%c\n" ,c);	…A(设c = "A")
%s	整个字符串	printf("…%s\n" ,save);	…good bye! (设save 是一个字符数组，其中内容是good bye!
%o	八进制数	printf("…%o\n" ,oct);	…1576(设oct = 01576)
%x	十六进制数	printf("…%x\n" ,hex);	…16AF (设hex = ox16AF)
%%	%本身	printf("…%%\n");	…% %在打印式中有特殊含义 %%就取消了%的特殊含义，从而打印出了一个%

这里还要指出，%d前面可以有数字，比如，%4d，表示打印出的十进制数至少占4位。%f前面也可以有特别标志，如%6f，表示印出的数字要占六个字符的位置，%.2f表示小数点后面数字占两位，而%6.2f则表示数字共占六个字符的位置，其中小数点后占两个位置。上述几种输出格式可以混合在一起使用。

[例1.10]

```
main (
{
int n;
n=511
printf ("what is the value of% d in
    octal? ", n) ;
printf ("%s%d decimal is%0 octal\n",
 "right", n, n) ;
}
```

将打印出：

what is the value of 511 in octal?
right 511 decimal is 777 octal。因为第一个printf后中没有换行标志“\n”，所以第二个printf的输出与第一个printf连接在一起了。第二个printf中顺序有%s，%d和%o三种输出格式，正好对应于后面的三个参数“right”，n和n，按要求分别输出字符串right，n的十进制数表示511，和n的八进制数表示777。

（2）参数和参数区分说明

函数如果带有参数的话，则要对参数的类型加以区分说明。参数区分说明应紧接在函数的参数表之后（闭圆括号之后）而在函数体开始之前（即{ 之前）。有关参数和参数区分说明等在第四章还要专门讲述

（3）函数返回值类型：函数一般情况下都要有返回值，调用者将利用返回值进行下步处理。很显然，返回值的类型可以多种多样，如果函数名之前类型位置为空，则按缺省约定将认为函数返回一个整型值。而对于其它类型的返回值，一定要加以说明。

至此，我们已经对C语言中的一般表示作了一个概括地介绍。初学者可以按本节中的一些例子来编写一些小型程序做为练习。

1.3 C语言的编写、编译和运行

本节对于初次上机的读者是有帮助的，它可以带领你进入UNIX系统，编写并运行你的C语言程序以便得到所要求的结果。当然，这里只能作一概括介绍，读者应不断地参阅有关的使用手册和资料以应付出现的各种复杂情况。对于UNIX已有了相当使用经验的读者可以跳过本小节。对于非UNIX系统其执行过程也有类似之处，可供参考。

1.3.1 UNIX系统文本编辑程序——预备知识

（1）进入和退出UNIX系统：如果在UNIX中你还未开过帐户，那么你首先得请系统管理员为你开列一个帐户。之后，你上机时，一旦终端有空，你就按一下return键或别的键（随系统而定），如显示屏或打印机上出现“login：”字样时，你就从键盘上打入你的注册名，这个注册名应与你要求系统管理员开列帐户时的名字一致。如有口令，则系统还要求你打入正确的口令。一旦你正确地注册进入系统之后，你就可以编辑你的C语言程序正文了。

当工作完成时，保留好你的文件，就可下机，此时如果处在shell命令输入态，则只要按control-d，终端上再次出现“login：”字样时，你就正常退出了。如果你还处在编辑程序态，则要先按q键，退回到shell命令输入态，然后按control-d退出系统。

（2）编辑C语言程序正文：你进入系统之后，就可以在适当的目录下建立文件以存放你的C语言源程序。为此你必须打入“ed ×××.c 〈r〉”，其中×××是C语言文件名，一般不超过8个字符，.c是C源文件标志，〈r〉是回车符。如果×××.c原来已有内容，则系统显示出“ddd…d”，ddd…d表示你原有文件的字符数：如果原先无此文件，则系统在临时区为你建立这个文件，并用“？×××.c”作为回答。

下面我们以例1.7来说明一般的C程序的编辑过程。

①打入ed sum.c

系统回答“? sum.c”后

②打入a 〈r〉进入附加方式，这是在文件sum.c中编写正文的一种主要手段。系统

并没有回答什么信息只是把光标移到下一行。

③打入正文，此时打入的每一行信息都作为正文行保留在文件sum.c中：

```
/*sum.c Adds three integers and prints
their sum */
main ( )<r>
{ <r>
   int a, b, c, sum; <r>
   a=1; b=2; c=3; <r>
   sum=a+b+c; <r>
   printf ("sum is%d\n", sum) ; <r>
}<r>
```

如果正确无误就应退出附加方式。等光标移到屏幕最左边后，

④打入。〈r〉，退出附加方式，从而回到了编辑状态，以后打入的都作为编辑程序的命令解释。

为了查看打入的内容是否正确，可打入1p〈r〉显示第一行，以后每按一个〈r〉就显示下一行。也可以打入1，$p〈r〉从第1行到最后一行（$表示）都显示出来。

如果发现打错了若干字符，可以用替换命令s进行替换，其一般形式是"s/老的正文/新的正文/"。如果丢了若干行，可以先用p命令走到漏行的下一行，打入"i〈r〉"命令进入插入方式，然后打入漏掉的若干正文行，这样在该行前面就插入了漏打的行。插入完成后，要打入"．〈r〉"退出插入方式。反之如果有多余的行，可以用"x，yd〈r〉"把第x行到第y行的内容删去（d的含义）。

⑤如果经过修改，正文正确了，就要打入"w〈r〉"把正文保留起来，因为在此之前正文文件还只在临时区中，如果不用"w"命令保留起来，当你退出系统以后，刚打入的正文可能丢掉。

⑥为了能编译和运行这个程序，需退出编辑状态回到UNIX shell命令状态。退出编辑程序的命令是q，打入"q<r>"即可回到UNIX shell命令状态。

UNIX的编辑程序还有许多，而且可以把若干命令组合起来，受篇幅所限，就不作进一步介绍了，表1-2列出了一些基本的编辑命令，详见《UNIX分时系统程序员手册》中有关ed命令的解释。

表 1-2 UNIX基本的编辑命令

命令表示	基本功能
q	退出编辑程序
w	把临时工作文件写到永久的磁盘文件中。
a	进入附加方式，新的正文被放到指定行的后面。
i	进入插入方式，添加的正文被放到当前行之前。
$=	印出指定行的行号。
α，β c	修改指定范围若干行（α到β）的内容，输入的正文放在文件中被删除的正文的位置。
α，β d	删去指定范围（α到β）的行。
α，β l	列出指定范围（α到β）的行。
α，β mγ	把指定范围（α到β）的行移到由γ指定的行后面，原有行删去。
α，β tγ	把指定范围（α到β）的行的一个副本放到由γ指定的行后面，原有行保留。
α，β p	把指定范围（α到β）的行印出。
α，β s/x/y	把指定范围α到β的行中的x换成y。

注：其中α，β，γ表行行号，使用时应该以具体行号代之。

1.3.2 *程序的编译和运行*：编辑以后得到的C程序是源程序形式，必须经过C语言编译程序转换成与它等价的计算机可直接执行的机器语言程序即目标程序，然后计算机执行该目标程序，才能获得结果。

在UNIX中，编译命令是cc，要编译已编辑好的sum.c，只要打入如下命令：

```
cc sum.c<r>
```

如果编译过程中没发现什么错误，则系统不给出任何消息，只是发出另一个提示符（通常是$）。此时编译好的目标文件将在a.out中。要运行这个文件，打入下述命令即可：

```
a.out<r>
```

当然如果在编译时采用下列命令：

```
cc -o suml sum.c<r>
```

则运行时就要打入

```
suml<r>
```

因为此时的目标程序不在a.out而在suml中了。

对于本例，一旦程序运行完毕，在终端上会看到如下结果：

```
sum is 6
```

然后再发出一个提示符$，意味着这个程序运行完毕，你可以进行下面的工作了。

C语言的编译命令cc还有若干任选，为的是完成各种辅助功能，详见《UNIX分时系统程序员手册》。作为基本功能，上面不带任选的cc命令已经够用了。

当然，一个程序在编译和运行中，总会发现此程序的若干错误，大型程序更是难免，程序的错误大致分为以下三类：

（1）语法错　在编译过程中产生的错误通常称语法错，这主要是由于在程序编制过程中，某些部分违背了C语言的语法规则而引起的。比如，变量名不符规则，语句不配对等。这类错误比较容易发现和纠正。通常用编辑命令进入该文件，把出错的地方更正过来，然后再次编译，这样反复几次总可以通过编译阶段而得到目标代码。

（2）运行出错　这不是编译时所能查出的错误。比如计算一个变量的平方根，而赋给该变量的值却小于0，又如程序中变量的取值范围超出允许的范围等。对于这类错误，运行时系统会发现并输出有关的信息，同时计算机暂停运行等待修正错误。为查找这类错误，可以在程序中插入一些检测和输出语句来更精确地定位错误。

（3）逻辑性错　这类错误在程序编译和执行时均不能发现。比如对一个变量应该赋值为1000，而赋值为100。为了便于检测这类错误，可在程序中插入一些输出语句，以便在调试阶段输出一些需要检测的中间结果。在实际运行时这类输出语句并不参加工作。

UNIX系统中有专门的调试工具可供使用。在此不作详述了。

下面图1-10列出程序编译和运行的过程

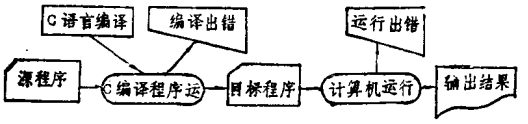


图 1-10　程序编译及运行过程

注：图中“c编译程序运”为“c编译程序运行”

最后，作为本小节的一个总结，我们把C语言程序的编辑、编译和运行的一些主要命令及它们所处的地位在下图（图1-11）中给出。

1.4 小结

本章第一节主要介绍了C语言的发展历史和它的特点。第二小节通过若干例子对C语言程序作了一般性描述，指出了它与别的语言的一些不同之处。第三小节主要介绍了在UNIX系统上编写、编译和运行C语言程序的全过程，这一过程具有代表性，在别的系统上也可参考。在以下各章中我们假定读者都是遵循这一过程的，而把重点放在C语言本身的特点及程序设计的方法上，所以本

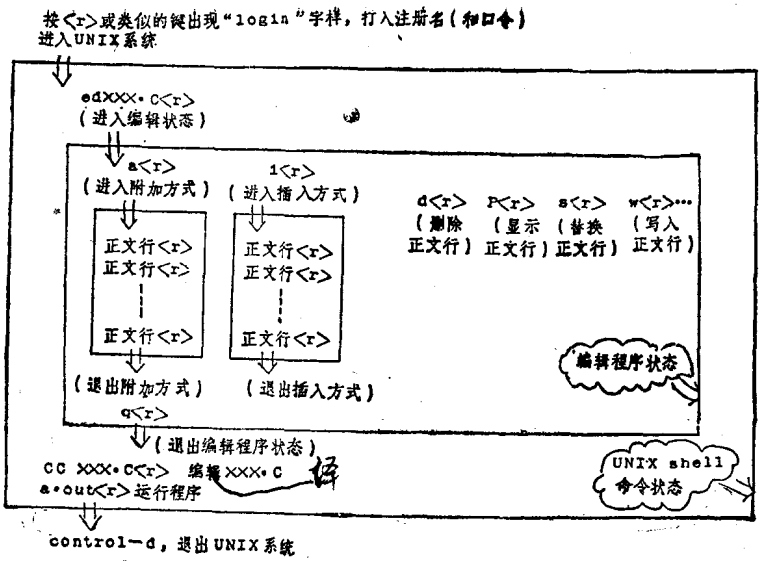


图 1-11　C语言程序编辑、编译和运行过程

小节是每一个实际运行C语言程序的人都必须很好掌握的。

1.5 习题

(1) 参阅附录中所列的参考文献，把C语言与pascal或其它语言的特性作一比较。

(2) 对于C语言的一些不足之处，以及某些人的非议，通过学习和参阅有关文献，谈谈你自己的看法。

(3) 在你的系统上运行1.4、1.5、1.6、1.8、1.10等程序。作为试验，拿掉程序的某一部分，或故意给出一些错，看看会得到什么出错消息。

(4) 试验一下，如果printf的参数串中包含\x（其中x不是表1-1中所列出的某个字符），会产生什么现象。

(5) 试把例1.7中执行语句后的分号";"去掉一两个，看看会出现什么情况。

(6) 试把某些例中开花括号移到main( )之前，看看会产生什么情况，这样可以吗?

注：图1-11中正数第9行两个方框内"正文行<r>"下加.<r>。第13行…编辑…为"编译"。

第二章　数据、表达式和赋值语句

本章介绍C语言中的一些基本概念，如标识符、变量、表达式等，一个最基本的语句——赋值语句。还要介绍C语言的一些简单（或基本）数据类型，如整数、字符和浮点数等。C语言其余的数据类型都要通过这些基本数据类型来构造。

2.1 标识符和变量

2.1.1 标识符：标识符实际上是一个字符序列，在C语言中，标识符用来标记常量、变量、数据类型、函数及程序的名字。当然标识符并不是一个随意的字符序列，根据图1-4所示的语法图，构成一个标识符必须符合下列语法规则：

a．以字母（大小写皆可）或下线符中任一字符打头；

b．在第一个字符之后，可以是任意的字母、下线和数字组成的字符序列，这个序列可以是空串。

在C语言中，标识符可以分成三类：

（1）关键字　专门用来说明某一固定含义的字。C语言共使用了27个关键字（见1.2.1或附录A）。对这些关键字不允许再赋予其它的含义（出现在说明部分）。C语言习惯用小写字母，所以这些关键字也都是由小写字母（大写的标识符常常有特定含义）构成的，又因为个数很少并且含义都很明确，在本讲义中我们就不特别指出了。

（2）特定字　具有特定含义的若干标识符，主要有7个（见1.2.1），它们主要用在C语言的预处理程序中。这些标识符虽然不是关键字，但是人们已习惯把它们看成是关键字，并赋予了特定含义，建议读者不要在程序中把它们作为一般标识符随意使用，以免造成混乱。

（3）一般标识符　通常是用户根据标识符的构成规则定义的标识符。由图1-4可知，下列的字均是合法的标识符：

nl, *nw*, *nc*, *inword*, *_file*, *file1*, *file2*, *is_long*而下面所写的字都不是合法的标识符：

1*c*　（不是以字母或下线字符打头）

c/c, *good bye*, #500（出现了非字母、非下线及非数字符号）

在C中，大小写字母有不同含义，比如标识符*nc*与*Nc*就不同。另外，标识符的字符数可多可少，但通常只有前面七个或八个字符有意义。这样，为了表示不同的对象，标识符的前七个或八个字符必须有所区别。例如，

*firsteightchars*与*firsteightint*

由于前八个字符相同，因此源程序在编译时，将被编译程序视为同一个标识符。

为了区分不同的标识符，究竟前面几个字符要不同，对于不同的机器，有不同要求。而C语言与各种汇编程序和装入程序约定使用的外部标识符，限制就更多（参见表2-1）。

表 2-1　各种机器上的C语言外部标识符

机　种	限制条件
DEC PDP-11	前七个字符，大小写不同
Honeywell 6000	前六个字符，大小写相同
IBM 360/370	前七个字符，大小写相同
Interdata 8/32	前八个字符，大小写不同
DEC VAX-11	前八十字符，大小写不同

一个好的程序，特别是一个有意义的实用程序，其中的标识符必须选择恰当，这样