C语言实用编程

主　编　崔　超　曲伟建
副主编　柴宝仁　闫公敬
主　审　张宪忠

東北林業大學出版社

C 语言实用编程

主　编　崔　超　曲伟建
副主编　柴宝仁　闫公敬
主　审　张宪忠

东北林业大学出版社

图书在版编目（CIP）数据

C 语言实用编程/崔超，曲伟建主编. —哈尔滨：东北林业大学出版社，2009.4
ISBN 978-7-81131-436-6

Ⅰ. C… Ⅱ. ①崔…②曲… Ⅲ. C 语言—程序设计—教材 Ⅳ. TP312

中国版本图书馆 CIP 数据核字（2009）第 058646 号

责任编辑： 张红梅
封面设计： 彭 宇

NEFUP

C 语言实用编程
C Yuyan Shiyong Biancheng
主 编 崔 超 曲伟建
副主编 柴宝仁 闫公敬
主 审 张宪忠

东北林业大学出版社出版发行
（哈尔滨市和兴路 26 号）
哈尔滨市工大节能印刷厂印装
开本 787 × 1092 1/16 印张 18 字数 426 千字
2009 年 4 月第 1 版 2009 年 4 月第 1 次印刷
印数 1—1 000 册
ISBN 978-7-81131-436-6
定价：35.00 元

目　录

1 C语言概述

1.1 C语言的特点

C语言的特点主要归结为以下几点：

(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，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1 000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编写外，其余都是用C语言描述的。当然C语言同样具有数值处理、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及很强的控制流结构。在基本类型(如字符、整数、浮点数等)的基础上，它可以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。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。

(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，特别像操作系统，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语言移植后效率降低不多，因此C语言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，所以根本不可移植，而一些高级语言，如Fortran语言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语言，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，大部分却是由C语言编译移植得到的。统计资料表明，不同机器上的C语言编译程序80%的代码是公共的。由于C语言的编译程序便于移植，使得在一个环境上用C语言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,但也有一些不足之处,例如运算符和优先级较多,不便于记忆,有些还与常规约定有所不同(如字位与或级别较低);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。

由于上述的几个突出优点,人们对 C 语言倾注越来越多的关心,以至于在世界上使用、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 语言程序的一些概貌,一方面使读者对 C 语言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 语言进行程序设计的讨论。另一方面,对于 C 语言程序编制、编译、运行的过程也做一概括介绍,以便有助于读者上机进行实际练习。

1.2.1 词汇表

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:

(1)数字:0,1,2,…,9。

(2)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,所以仅有大写字母的终端在使用 UNIX 时会遇到困难。

(3)下划线字符:_,这个下划线字符起一个英文字母的作用,因此在构成标识符等语法成分时,以"_"打头,在 C 语言中是合法的,而在通常的语言中是不允许的。

(4)特殊符号,主要包括下列一些运算符和关键字:

①运算符:+,-,*,/,%(取余),=(赋值),<,>,<=,>=,!=(不等于),==(全等比较),<<(左移),>>(右移),&(逻辑位与),|(逻辑位或),&&(与运算),||(或运算),^(逻辑位异或),~(按位求反),(、),[、],→(指向),.(成员),!(反),?:(三元运算符)等(详见表 2-2)。

②关键字:int, char, float, double, struct, union, long, short, unsigned, auto, extern, register, static, typedef, goto, return, sizeof, break, for, continue, if, else, do, while, switch, case, default, enum。

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 及 endif。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 语言也有字特定的语法规定。

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他符号、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,不允许将二数相乘的乘号“*”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(至少是表面上),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。

1.2.2 C语言程序结构

(1)C语言程序和主函数。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,可以是常量,也可以是变量。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语言程序的基本结构,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1.1]

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

针对这一例子说明如下:

①C语言程序一般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语言程序总是从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。

②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,注释行被忽略掉,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1.1中有一对花括号,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语言程序,实际上它什么也没有干。

上述程序也可以写成一行。

[例 1.2]

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

如例 1.1 和例 1.2 所示,C 语言程序的一般函数或“main( )”之后及最后的闭花括号“}”之后都没有通常的“;”或“.”之类语法符号。

C 语言程序包括程序模块或包含(include)模块。这里简单讲述一下程序模块中的主函数。在一个完整的 C 语言程序中必须有一个主函数,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。一个最简单的 C 语言程序,可以无参数,函数体为空,如例 1.1 或例 1.2 所示。

前面已说过,一个 C 语言程序可由若干函数(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。一个训练有素的 C 语言程序员,在 C 语言程序设计中,应善于用多个小函数来构成一个大程序,同时应能把一个大程序分解成若干个能单独完成特定功能的子程序。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 1.3]

```
/ * smallsub. c smallest C Program with a subProgram * /
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权返回主函数 main。

函数调用语句通过列出被调用函数的名字并给出实在参数来表示。本例中实在参数为空。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(如例 1.3),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(如例 1.3 中的调用语句)之后的分号“;”在 C 语言中作为语句终结符,而不单单是作为语句分隔符存在的;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“}”。

最后要说明一点,C 语言中构成一个程序的若干函数,其组织顺序通常是无关紧要的,

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做任何规定。

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说明;而函数主要包括内部数据说明和语句表。有关函数定义我们在第四章中还要详谈。

基本函数如例 1.3 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,但在简单情况下,可以什么也没有,如例 1.1;也可以只有一个语句,如例 1.3 的 main;复杂的情况下也可以带有说明和多种语句,下面例 1.4 就是复杂情况中的一个例子。该程序要做的工作是把三个整数相加并且打印它的和。

[例 1.4]

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

其中,前面的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“%”之前是要直接打印出的字符串。每个“%”都指示其他(第二个,第三个,……)参数要被替换成值并指出用什么格式来打印出它。所以,其他参数即是要输出的变量。

[例 1.5]

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

这是 printf 的一种简单的使用,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 1.6 产生同样结果。

[例 1.6]

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

其中\t 是制表跳格(一般为 8 格),"\n"是换行符。

例 1.6 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 1.5 相同的结果。而在例 1.4 中,如果 sum 是 6,则输出将是 sum is 6,这是一个比例 1.5、例 1.6 复杂的例子。因为"printf(" sum is % d\n" ,sum) ;"中,第一个参数含有"% d",这意味着参数表中下一个参数(即变量 sum)将作为一个十进制的数被打印出来。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,表 1 - 1 中列出常用的一些打印格式。

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 = 0x16AF)
% %	% 本身	printf(" … % % \n") ;	…% % 在打印式中有特殊含义,% % 取消了% 的特殊含义,从而只打印出一个%

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,表示打印出的数字要占 6 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 6 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直使用。

[例 1.7]

```
#include <stdio. h >
main( )
{
int n;
```

```
n = 511;
printf("what is the value of %d in octal?",n);
printf("%s%d decimal is %o 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 的格式控制串中没有换行标志"\n",所以第二个 printf 的输出与第一个 printf 的输出连在一起了。第二个 printf 中依次有%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。参数区分说明可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。而对于其他类型的返回值,一定要加以说明。

至此,我们已经对 C 语言中的一般表示概括地做了介绍。初学者可以按本节中的一些例子来编写一些小型程序作为练习。

1.3 C 语言程序的编写、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,这里只能做一概括介绍,读者可参阅有关的使用手册和资料以应付出现的各种复杂情况。熟悉 UNIX 系统的读者可以跳过本小节。对于非 UNIX 系统其执行过程也有类型之处,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 中你还未开过账户,那么你首先得请系统管理员为你开列一个账户。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账户时的名字一致。如有口令,则系统还要求你打入正确的口令。一旦你注册进入系统之后,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"键,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"键退出系统。

(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"vi ×××.c<r>",其中"×××"是 C 程序文件名,一般不超过 8 个字符,"c"是 C 语言源文件标志,"<r>"是回车。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 1.4 来说明一般的 C 程序的编辑过程。

①打入"vi sum.c"。

②系统回答"? sum.c"后,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。

④等光标移到屏幕最左边后,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"1␣p <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>",将第 1 行到最后一行(用 $ 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>",把正文保留在文件中,因为在此之前正文还只在临时区中,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,因受篇幅所限,就不做进一步介绍了。表 1-2 列出了一些基本的编辑命令(详见有关手册或书籍)。

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

续表 1-2

命令表示	基本功能
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,βmγ	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,βtγ	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,βp	把指定范围(α到β)的行印出
α,βs/x/y	把指定范围(α到β)的行中的x换成y

注:其中α,β,γ表示行号,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。要运行这个程序,打入下述命令即可:

a.out <r>

a.out是运行程序的缺少文件名,为了长远地保留编译后的可运行程序,需要把它放在永久文件中,因为a.out只能临时存放编译好的程序。为此,一种办法是在编译时指定永久文件,比如sum1,这样,编译命令应按下列方式打:

cc - o sum1 sum.c <r>

编译命令cc后的任选项-o就是用于指定永久文件(本例中是sum1)的。以后运行时就要打入:

sum1 <r>

因为此时的目标程序不在a.out而在sum1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。

这里提一下预处理和分别编译。

C语言整个源程序的代码可分散在多个文件里。将源程序放在分散的文件里是为了便于模块设计、开发和调试。事实证明,包含几个子程序(典型的是几百行代码)的文件,将比包含几千行代码的大文件更容易了解和工作。一个文件的程序能分成几个文件,比如将子程序或数据结构收集成几个组,将它们分别放在独立的文件里。但子程序或一个数据说明这种逻辑实体是不能分割在两个文件中的。

把一个程序分成几个文件将会引起一些新问题。一个问题是某个文件里的子程序需要知道另一个文件里有关的子程序和数据说明。C语言有一种特别的说明叫外部定义。外部定义的目的是为了引用及描述那些不能在本文件里定义的项。如果引用其他文件中的项目个数很少,则少量外部定义可放在本文件里。可是,若这个数目太大,最好建立一个包括外

部定义的文件,并使用 C 编译程序的 include 特性去蕴含这个文件。通过使用 include 伪指令,把文件包括进去。在一个 C 程序里,下面行

#include "defs. h"

将蕴含文件"def. h"的内容。

另一个问题是:整个程序需要知道有关的常量和单词。例如在一个表格处理的程序里,表格中项目个数的最大值是一个关键常量,需被所有子程序知道。在 C 程序里,define 伪指令能用于建立一个指定的常量。下面行

#define LISTLEN 20

在 C 程序里将定义一个名叫"LISTLEN"的常量。在一个程序里,每当遇到名字"LISTLEN"时,就用常量"20"替代它。一个 include 文件通常包括了整个大文件都使用的全部定义。define 伪指令实际上是一个宏置换机制,它可有参数,也能用于生成复杂的 C 语言代码序列。

在一个 C 语言程序里,"#"开关的各行均被用做像 include 或 define 一样的伪指令(或称预处理语句)。所有这些伪指令均由一个称做 C 语言预处理程序的模块进行处理,这个预处理程序是 C 语言编译程序的一个部分。C 语言预处理程序也能实现其他工作,像从一个程序里删去注解,等等。有时,你若想了解 C 语言预处理程序是否按你的要求去做,可看一下它的输出。你也可用这个技术去核实一个注解,看它是否已经不知不觉地走出了它预定的限制。如 UNIX shell 命令:

cc - p myprog. c

将对文件"myprog. c"进行预处理,并将输出放在文件"myprog. i"里。

C 语言编译程序的正常目标是编译一个完整的程序,以便产生一个可执行的模块。如果你已将一个大程序分成几个文件,那么你可以通过在命令行中提及全部文件以实现一个完整的编译:

cc fileA. c fileB. C fileC. c

如果都工作正常,没有错误查出,则将产生四个文件:名为'a. out'的文件包括了可执行的程序,"fileA. o"、"fileB. o"及"fileC. o"三个文件包括了相应源文件的目标代码。

为了减少以后编译的工作量,几个独立源文件的目标代码被保留下来。如果"fileA. c"修改了,而"fileB. c"和"fileC. c"没有改变,则已过时的目标文件只有一个"fileA. o"。目标模块"fileB. o"和"fileC. o"仍能够使用,因为它们所依赖的源代码并没有改动。通过只对"fileA. c"的部分编译来实现了整个系统的重新编译,然后将这三个目标文件重新连接而产生一个新的可执行文件"a. out"。上述工作可通过打人下列命令来完成:

cc fileA. c fileB. o fileC. o

C 编译程序的一个重要特点:它能从后缀确定是哪一种文件。根据后缀".c",C 编译程序知道"fileA. c"是必须编译的源程序;根据后缀".o",它知道"fileB. o"和"fileC. o"是只需要(和"fileA. o")连接的目标文件,通过连接可产生一个可执行的"a. out"文件。

要记住哪些文件已过时,哪些未过时是十分困难的。一种方法是经常将最新的目标文件保存下来。每当一个源文件改变时,相应目标文件应立即予以重新建立。部分编译这个词是说明一次编译的目标仅仅是一个模块,而不是整个软件系统,C 编译程序的任选项"-c"通常用于部分编译。命令:

cc fileA. c

将产生一个最新的“fileA. o”。如果不使用任选项“ - c”,当 C 编译程序发现文件“fileA. c”中的源代码不是一个完整程序时,它将送出一个出错消息。

送入下列命令,可从一组最新的目标文件产生“a. out”:

cc fileA. o fileB. o fileC. o

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。不过,作为基本功能,上面不带任选的 cc 命令已经够用了。

其实,在连接目标模块同时,cc 命令还把程序中用到的许多库函数(比如输入/输出函数、数学子程序等)的目标模块都连接在一起,构成一个完整的可执行文件(缺省时就是 a. out)。

另外,当源程序交 cc 编译时,第一步得到的是内部汇编代码,要经过汇编才能得到相应的目标模块。不过这种汇编代码及汇编过程外人是不知道的。

图 1 - 1 列出了 C 语言程序编辑、编译和运行的全过程。

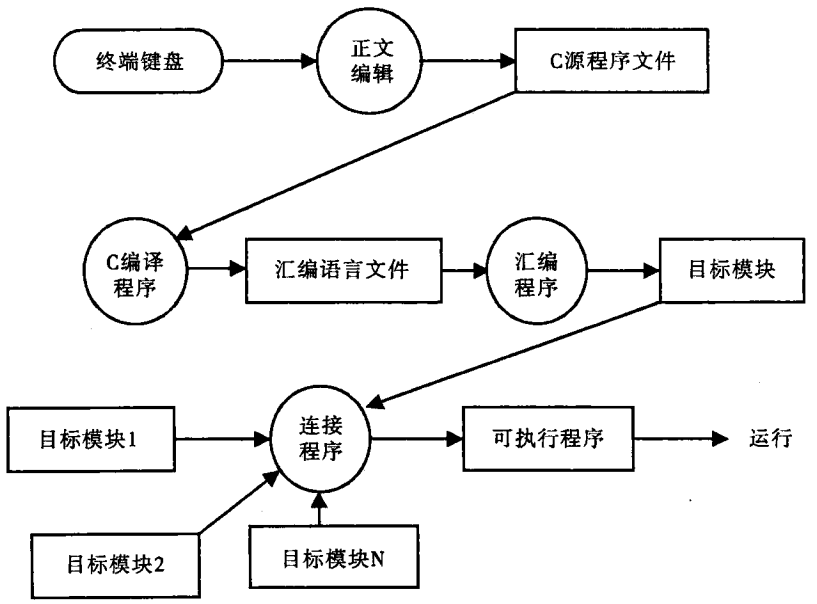

图 1 - 1 C 语言程序编辑、编译和运行

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

(1)语法出错。在编译过程中产生的错误通常称语法出错,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,变量名不符合规则,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精确地定位错误。

(3)逻辑性出错。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与工作。

UNIX 系统中有专门的调试工具可供使用。在此不做详述了。

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图 1-2 中给出。

按(r)或类似的键出现"login"字样,打入注册名(和口令)。

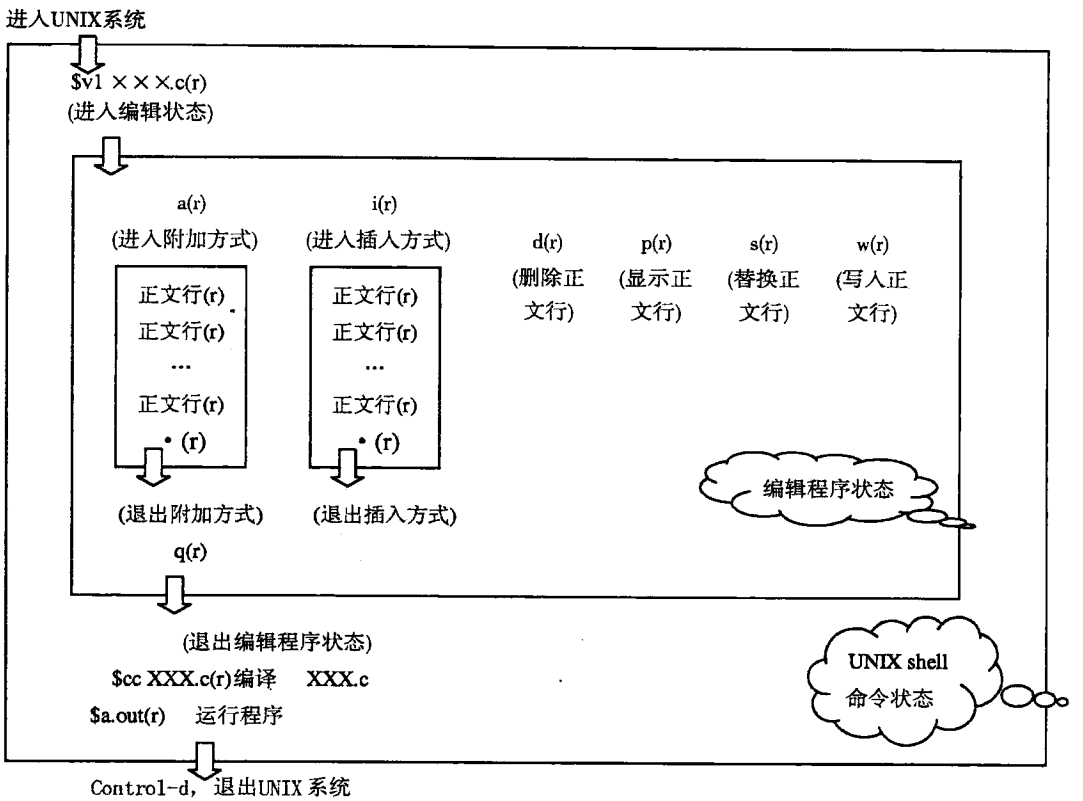

图 1-2 C 语言程序编辑、编译和运行环境

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;1.2 节通过若干例子对 C 语言程序做了一般性描述,指出了它与别的语言的一些不同之处;1.3 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,所以本章是每一个实际运行 C 语言程序的人都必须很好掌握的。