C语言工具库

《美》克里斯·詹姆萨 著
罗正棣 译

北京科海培训中心
一九八七年七月

编辑： 科海培训中心教材部
发行： 科海培训中心资料组
地址： 北京2725信箱 科海培训中心
资料组
（北京海淀路332路黄庄站旁）
印刷：河北省蓟县印刷厂

译 者 的 话

本书向广大的C语言的使用者提供丰富的编程工具，帮助您设计出高效的C语言程序，在可读性、可重用性上都具有高水平。

为数达125个之多的各类例程使本书成为一本实用性很强的资料。这些程序包括串操作、指针、I/O操作、数组操作、递归、分类排序、文件操作、三角函数等各方面的短小精悍的程序，从而构成了一个完整的通用C语言工具库。对于C语言的初学者来说，本书可以作为一个编程示范；对于那些熟练者，你们将发现本书中的例程将是你们以后设计C语言程序的极其方便、有效的工具。

无论是大、中、小型计算机，只要装备了C语言编译程序，本书中的程序工具就可以使用。在目前普及的IBM PC、PC/XT、PC/AT个人机及其兼容机上，也已经有很好的C编译程序，如Turbo-C Lattice C，Microsoft C因此本书尤其适合于广大PC机使用者。

为了进一步推广C语言的应用，发挥C语言的效力，译者从众多的C语言论著中选译了这本以实用性见长的工具书，希望能得到广大C语言使用者的喜爱，并热诚地希望各位使用者交流体会、不吝赐教，让C语言的工具库更丰富，使用起来更得心应手。

最后，译者在此鸣谢资料提供者，并对科海培训中心的编辑同志的辛勤劳动表示深深的谢意。

译者

1987.7.10.

序　言

我们每天都在使用工具，我们的生活变得轻松愉快。人类一直是聪明的工具发明家，而计算机似乎是人类所发明的最有用的工具了。无论我们是在装配一辆汽车，还是在把一个文件加到另一个文件上去，工具总是使工作更容易。本书的目的是给出许多C语言编程工具，读者在开发C语言程序时可以利用它们。

最近几年来，C程序设计语言开始被广泛地使用起来。虽然当初C语言只是一种系统程序设计语言，但是，它并不只限于用在系统程序设计方面，现在有许多程序设计者在一般的应用程序设计方面挖掘C语言的潜力。C语言有于一般应用的特点有：可移植性、模块化、以及通常只有汇编语言具备的机器操作访问能力。

过去，如果我们需要把一种计算机上运行的程序移到另一种计算机上，那末就要改动程序的相当大的一部分。C语言的主要设计目标就是打破这种机器相关性。因而，C语言成为现存的程序设计语言中可移植性最好的几种之一。用C语言为某种计算机写的程序，如IBMPC，只要作很少的改动或根本不要修改，就可以自然地在另一种计算机上运行，如APPLE机器。

任何一个程序设计者都希望能把庞大的任务分解成几个更小更容易实现的任务。许多初看起来大得无法编的程序往往可以剖开成为几个更小的子任务，这些子任务则比较容易设计和编码。把程序分解为几个功能部分的另一个好处是，一个程序的功能部分往往可以不经过多少改动就又被一个没有关系的应用程序使用。如果我们的大多数程序都用许多功能部分来实现，那末就增加了代码的可读性并减少了程序的编制和调试时间。另外，也可以写一系列可以放入库中的例程，以便其它程序来共享。我们的目标是大量的只执行一个基本任务的函数。这样就可以增加我们的例行程序的可重用性。

关于UNIX

七十年代早期，贝尔实验室开发出了UNIX操作系统。由于UNIX极其灵活，又给用户提供了大量开发工具，近几年来它被广泛地使用。UNIX正顺利地渐渐成为操作系统的工业标准，另外UNIX操作系统的大部分代码是用C语言写的，因此在不远的将来，用C语言开发的应用程序和对C语言程序设计员的需求量都会大大增加。

如何使用本书

这里假定读者对C语言已经比较熟悉，或是正在学习C语言。第1章是C语言的简单概况，并不是C程序设计语言的入门指导。若读者正在学习C语言，则本书中的许多程序可以帮助你在最短的时间内写出功能强大的程序。此外，考察这些带有详明的例行程序，你也能学到很多东西。若读者是个高级C语言程序设计员，本书中的许多例程将向你介绍如何编写良好的程序设计工具，以及在开发与UNIX操作系统所支持的程序相似的实用程序时，需要考虑什么。

读者手里最有用的工具是调试写语句。本书中的例程都包括详细的说明，但只有用后才能完全理解它们。读者最好在每个例程中都使用一下调试写语句，以便加深对处理过程

而理解。

本书的每一章都在前面概念的基础上引入新的课题。第1章是C程序设计语言的简单概述，但不是C语言的入门指导，只是一个快速参考指南。第2章介绍常量和宏功能。后面的各章将要用到这些常量和宏功能，它们被放在文件defn.h，math.h和strings.h中。所有访问常量和宏功能的程序都要包括这三个文件。第2章给出这些文件的内容。在第3章里给出几个串操作例程。第4章考察各种指针及它们在串操作中的用法。第5章集中于用户接口和开发良好的I/O例程。第6章给出一些为一般的数组类型设计的数组操作例程，每个例程都可用于需要int数组，float数组，或double数组的应用程序。第7章考察递归功能，以及怎样用它来简化复杂的程序设计任务。每个递归例程都有非常详尽的解释。第8章介绍分类算法，具体为冒泡法、shell法和快速分类算法。同样地，这些例程也设计得允许有int，float或double三种数组类型。另外，排序顺序的改变（up或down）不需要多个例程。第9章给出一系列执行三角函数功能和字符转换功能的例程。在第10章里，我们要介绍许多与UNIX环境中的实用程序相似的文件操作例程，以作为第1章到第9章中设计工具的综合演示。第11章介绍UNIX管道的概念，以及如何开发支持管道的例程。

读者现在设计可被其它程序利用的例程所花的额外精力在以后会成倍地节省你的时间和精力。

克里格·詹姆萨

一九八五年

目　录

第一章　C语言概况

本章的目的是作为读者在考察书中例程时使用快速参考手册。除了C语言概况外，本章还给出C语言语法图，以及这些语法图的解释，和它们在设计C语言程序时的用法。如果读者对语法图感到陌生，这也不用担忧，到了本章的末尾，读者就能靠简单的解释理解用语法图给出的信息了。

指针和地址

本书中的大部分程序都用指针（pointer）来记录一个位置。有些原因使许多程序设计员对指针感到难以使用。首先，某些程序设计语言不允许使用指针。其次，许多程序员不用指针设计程序，因为他们不习惯指针操作，好象只会增加麻烦。最后，大多数的书籍没有恰如其份地对指针作解释。因此，在介绍C语言的指针用法之前，让我们回头温习一下理解指针用法的几个关键性概念。

使用指针的根本目的在于标识存储位置。存储体被划分为许多位置，每个位置都可以存放信息，因而就需要一个方法来将一个值放入某个特定的位置及随后能找到这个值。解决问题的方法是给每个位置赋以一个唯一的地址。例如，你将一个值放入地址是1000的存储位置，以后就能找到这个值，因为你知道其存放地点。但是，如果你一定要用实际地址来记录存储位置的话，那末你的程序就将非常复杂以致不可理解。相反地，程序设计语言让你用变量来存储数据。一个变量可以看作仅仅是你赋给一个或一系列存储位置的一个有意义的名字。因此，一个变量与两个值相关联，第一个是你赋给这个变量的值，第二个是包含这个值的地址，或存储位置。

指针是包含存储地址的变量。在C语言中，用符号&和＊来使用指针。与号（&）用来描述变量的地址，而不是变量中的值。例如，如果你预先说明了整数类型的变量my-data和指针int-pointer，你就可以把变量my-data在内存中的地址赋给指针int-pointer，如下式：

```
int my-data;                    /*变量说明*/
int *int-poioter;               /*指针说明*/
int-pointer = &my-data;         /*地址赋给指针*/
```

一旦int-pointer赋以my-data的地址，则两个变量将引用同一个存储位置。星号（＊）具来返回针所指位置中的值。表达式＊int-pointer引用的值，包含在int-pointer中的地址当下。

假如指针string被说明为指向串“Computer”的指针，那末变量string中实际包含的值是该串第一个字符（C）的地址。如果你打印string中的值，则会显示出字母C在内存中的地址。如果想要打印出实际的字母，就必须用到星号了，如下：

```
putchar (*string);
```

如果想要打印出整个串，可以先打印出变量string包含的位置中的字符，然后增加变量string的值，让它指向下一个字符，如下句：

```
while (*string != '\0')
    putchar (*string++);
```

后缀表达式*string++印出string引用的存储位置中的字符，然后使string增值，让它指向下一个存位置。

使用指针是理解它的最佳办法。第4章里给出一些对字符串使用指针的例程。实际用这些程序，看看你是否得到预期的结果。最好在每个这样的例程中使用一下调试写语句。如果你打印出了指针包含的地址和它引用的值，就可以很好地理解指针操作了。

语法图

在读语法图之前，你应该掌握下列概念：

表达式是表达一个数字运算的一个或一系列符号。下面是几个正确的C语言表达式：

```
a = a + 1
a                    与表达式a = a+0或a = a - 0相同
a + + < = 17
```

标识符是标识一个客体的唯一名字。变量名是标识符的特例。

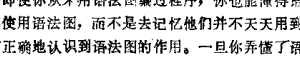
文字是常量值。本章后面的语法图中，文字通常指C语言中的保留字和标点符号。

读语法图时要从左向右，按照把图中的符号连接起来的箭头的方向读。因为语法图是由许多符号组成的，即使你从未用语法图编过程序，你也能懂得语法图语言。许多经验丰富的计算机科学家都使用语法图，而不是去记忆他们并不天天用到的语法结构。不过，我们中的许多人都没有正确地认识到语法图的作用。一旦你弄懂了语法流图，它就将成为很有用的工具。

表1-1 是用于C语言语法图的各种符号。

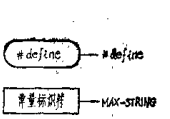
让看下例：

```
#define MAX_STRING 255
```



如果你从这个语法图的第一项开始，沿着箭头的方向走，则这个语法给出下列信息：

图中的前两个符号的意义是直观的。第三个符号，表达式操作数，需要进一步的解释。包括在矩形框中的词语表达式操作数指示着表达式操作数是用另一个语法图定义的。如果考察一下表达式操作数的语法图，就会发现它可以是下面任何一个：



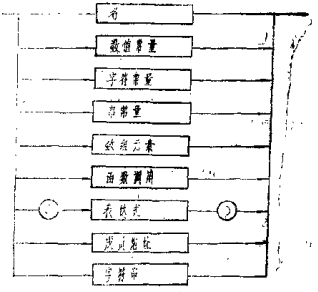


表1-1 用于语法图的符号

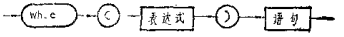
（圆角矩形）	圆角矩形符号中包含一个文字值，它在程序中按原样写出 例：（ask）（file）（continue）
○	圆形符号包含一个文字值，它通常是一个特殊符号或标点 例：（;）（<）（"）
（矩形）	矩形符号包含一个有另外的语法图定义的语言构造。 例：（标识符）（while循环）（if语句）

在这个例子中，表达式操作数是数值常量255。

再看下面的while循环：

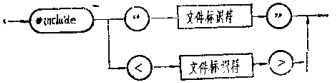
```
while (i<=10)
    i++;
```

如果考察while循环的语法图，



文字while，（，和）都是直观的。这个例子中表达式是i<=10，语句是i++;。

如果语法图中可走的路径不止一条，你应该选择适当的路径按箭头方向走下去。例如，程序中包括文件stdio.h的#include语句的语法图是这样的：



如果你已经知道操作系统是怎样处理双引号和尖括号中的文件的，那末语法图能给出正确的语法；否则，你必须查阅一下编译程序的参考资除。

语法图仅仅是个使指令语法正确的指导。如果你从未用过语法图，初看起来可能会不知所措，但要记住语法图是一个好的工具。把C语言的不同构造（while，do-while，for，if等）和它们的语法图对比一下，你就能渐渐领悟语法图的奥妙了。如果你不能理解某个特定的语法图（例如，do-while循环），那末先试着用C语言写一个这样的语句，然后作一下比较。由于语法图的使用日趋广泛，所以必须要掌握它。

下面各种语法图用来说明C程序设计语言句语法。多数的图后有程序设计的例子。有些语法图是给高级C程序设计员准备的。

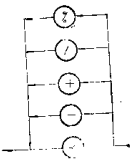
实在参数

例：

a，b，c

c

算术运算符



例：

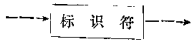
```
x = 3 * 5 - 2;
y = 7/2
remainder = 3%2;        /* 余数赋值 */
```

数组元素

例：

```
string [1]
argv [2]
argv [argc-1]
```

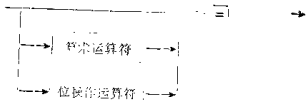
数组名



例：

string

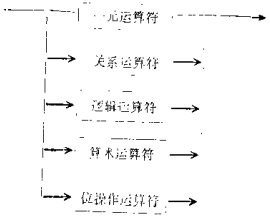
赋值运算符



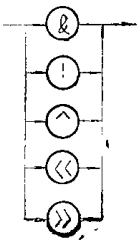
例：

```
x = 5;
x + = 5;      /* x = x + 5 */
x >> = 2;     /* x = x >> 2 */
```

二元运算符



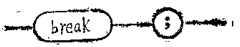
位操作运算符



例：

```
mask = x&255;
while (x<32)
    {
    printf ("%d\n", x);
    x = x<<1;
    }
```

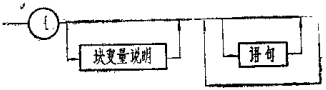
break语句



例：

```
case : {
    printf ( "The value is2\n" ) ;
    break;
    }
```

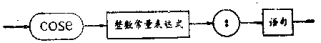
复合语句



例：

```
    {
char letter;      /*块变量*/
for (letter= 'a'; letter<= 'z'; letter++)
    putchar (letter) ;
putchar ('\n') ;
}
```

Case语句



例：

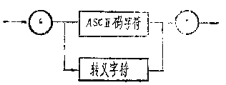
```
case5: {
        printf ( "The digit is 5\n" ) ;
        break;
        }
```

投射

例：

```
int i;
float x =3.775
i= (int) x     /*赋i为3*/
i= (int) (x+0.5) /*赋i为4*/
```

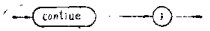
字符常量



例：

```
'A'
'\0'
```

continue语句



例：

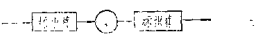
```
if (i%2 == 0)
    continue;
```

命令行说明

例：

```
int argc;
char *argv[ ];
```

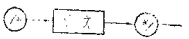
命令行参数



例：

```
argc, argv
```

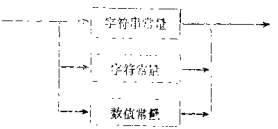
注解



例：

```
/*为文件操作包含stdio.h*/
/*注解可以
    超过一行*/
```

常量



例：

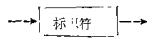
```
"This is a string constant"
'\0'
255
```

常量说明

例：

```
#define EOF -1
#define string"This is a string constant"
#define EOL'\n'
```

常量标识符



例：

NULL

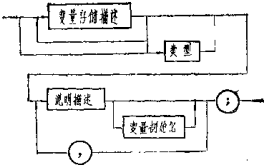
EOF

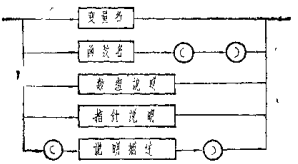
说明

例：

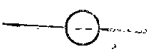
```
static long int count;
long index;
int scores [NUM_SCORES] ;
float x=3.1573
int sum, average, max, min;
```

说明描述





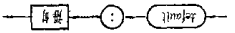
减量运算符



例：

```
for (i=10; i>=0; i--)
    printf ("%d\n", i) ;
```

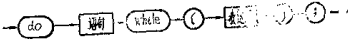
缺省语句



例，

```
case1: {
        printf ("The value is one\n") ;
        break;
        }
default:printf ("The value is not one \n") ;
```

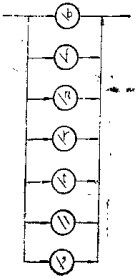
do-while 循环



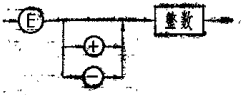
例：

```
i = 0;
do
    printf ("%d\n", i);
    while (i++ <= 100);
```

转义字符



指数值



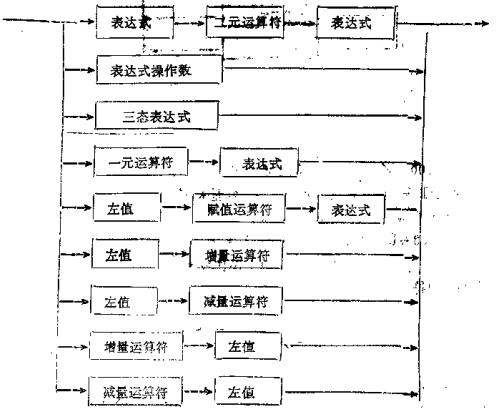
例：

E4

E+4

E-4

表达式

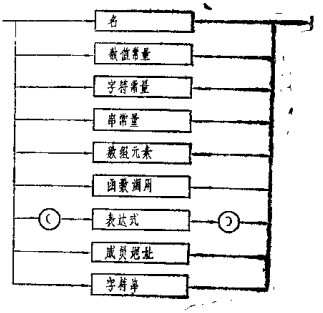


例：

4

```
a+b
a
(x>1) ? 1:0
++ string
*string++
a=a-5
```

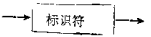
表达式操作数



例：

```
x+100  /*名字+数值常量*/
a[i]+'a'/*数组元素+字符常量*/
min(x, y)/*函数调用*/
a+(3*b)/*名+(表达式)*/
        /*(3*b)是单个操作数*/
```

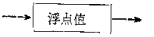
文件标识符



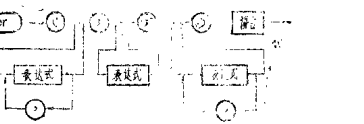
例：

```
stdio.h
```

浮点常量值



for循环



例：

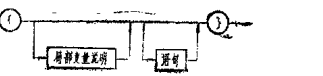
```
for (i=0; i<=10; i++)
    printf ("%d\n", i*i);
for (; ;)      /*infinite loop*/
for (i=100; j=100; j<100; i++, j++)
    printf ("%d\n", i*j);
```

浮点值

例：

```
float y=123.;
float y=123.457
double x=1.14E-5;
```

函数体



例：

```
{
} /*函数中无语句*/
{
printf ("Error opening output file \n");
}
{
int count; /*局部变量*/
for (count=0; count<=10; count++)
    printf ("%d\n", count);
```