C语言高级程序编制技巧

曾琪明 编译

中国科学院希望高级电脑技术公司

C 语言高级程序编制技巧

曾琪明　编译

中国科学院希望高级电脑技术公司
一九九〇年九月

前　言

我有幸写一本我一直想要得到的有关程序编制的书。数年前，当我刚开始编程时我试图图找到一本这样的书，它直截了当地给出排序、链表、模拟和表达式语法分析程序等工作的算法。我渴望有一本这样的书，它能告诉编程的奥秘，并且随时能从中找到我所需要的东西。但遗憾的是我从未如愿。因此我决定写一本这样的书。

本书内容广泛，它给出了许多用C语言写成的有用的算法、函数和实例，C语言既是一种事实上的系统程序设计语言，又是目前可得到的最流行的通用专业编程语言之一。实际上现在所有的计算机上都装有许多廉价的C语言编译程序。我在IBM PC上使用的是Aztec C 86编译程序。除少数例外，任何与UNIX系统V 7版兼容的C语言编译程序都能运行本书中的程序。

第一章首先扼要地介绍了C语言历史，回顾了C语言的主要内容。第二章讲述了数据和磁盘文件的排序。第三章讨论了栈、队列、链表和二叉树（你也许认为在一章中讨论这些内容太多了，但将这些相关内容归在一起既易读又可构成一个充实的单元）。第四章阐述了动态分配的方法，第五章综述了操作系统接口和汇编语言的连接。第六章介绍了统计并给出了一个完整的统计程序。第七章讲述了编码、密码和数据压缩，并简要介绍了密码学的历史。在第八章中详细介绍了几种随机数生成程序，并讨论了它们在两个模拟中的应用。第一个模拟是商店中的结帐队列，第二个是随机行走的证卷交易管理程序。

第九章给出了一个递归下降语法分析程序的完整程序。数年前，我曾愿以任何代价去交换去获得它。如果你要计算表达式，该章恰好满足你。第十章和第十一章分别讲述了其它语言向C语言的转换，以及效率、移植和调试等内容。

目　录

第一章 C语言回顾

本书采用解答问题的方式来说明C程序设计语言中的高级概念。它分析常见的程序设计任务，并给出解法，它着重于编程的风格和结构。通过这种途径，本书讲述了C语言的各种高级专题和细微之处，以及每个解法所涉及到的一般编程理论。你应该具有一定的C语言程序设计知识，但不必有很多的经验。本章的后半部将对C语言进行回顾。

本书可以避免变量名与函数名的混淆。例如，一个叫“test”的变量，印成test，同名的函量无论在何处都印成test（）。

本书中列举的例子及程序都经IBM PC上的AztecC和Super Soft C编译程序编译并运行过。一般地与UNIX7版兼容的各种编译程序如Lattice或者Microsoft编译程序都能编译运行本书中的程序。大多数计算机都有多种编译程序可用，找到适合你需要的应当没有什么困难。必须记住，所有的编译程序都略有不同，尤其是它们的库，因此必须阅读你所采用的编译程序的用户手册。

§1.1 C语言起源

C语言是由Dennis Ritchie发明的，并首先在使用UNIX操作系统的DEC PDP—11上实现的。C语言是从着手开发一个叫做BCPL的老的语言过程中产生出来的。BCPL语言目前仍在使用，主要是在欧洲。BCPL是Martin Richard开发的，它对一种称为B的语言有影响。B语言是由Ken Thompson发明的，它导致了C语言的研制。

虽然C语言有七种内部数据类型，但与Pascal或Ada相比，它不是一种强类型化的语言。C语言允许几乎所有的类型转换，且在大多数表达式中可以自由地混合使用字符型和整型。它不做运行时错误检查，如数组越界检查或变量类型一致性检查，因此这些就成为程序员的责任了。

C语言的特殊之处在于它允许直接进行位、字节、字和指针的操作。这就使得它很适合于系统程序设计，在这方面上述操作经常进行。C语言的另一个优点在于它仅有28个关键词，它们组成了C语言的命令。相比之下，IBM PC BASIC却有159个关键词！

虽然最初开发的C语言是在UNIX操作系统下运行的，但现在C语言已相当普及。实际上所有计算机和操作系统都有其编译程序，这意味着C语言程序在计算机之间和操作系统之间非常容易移植，从而使得一次编写的程序到处可用。

§1.2 结构化的C语言

C语言与Algol和Pascal语言有些类似，它也是一种结构化的语言。从学术的意义上讲，虽然把C语言称为模块结构化的语言不严格，但它仍是这类语言集合中非正规的一部分。模块结构化语言的显著特点在于代码和数据的划分，这意味着语言可以把执行特定任务所必需的全部信息和指令与程序的其余部分划分开并隐蔽起来。通常是对具有局部变量的子程序进行划分，局部变量是暂时的。因而用这种方法可以编写出这样的子程序，即在子程序

内部发生的事件对程序的其它部分没有副作用。过多地使用全局变量（贯穿整个程序的已知变量）也许会使得错误和不希望的副作用潜入程序中。在C语言中，所有的子程序都是分立的函数。

函数是C语言的组成部分，所有的程序活动都以函数的形式出现。它们允许分别定义和编写一个程序中的特定任务。当调试完一个仅用局部变量的函数后，它就可以被信赖地在各种情况下正常工作而不会给程序的其它部分带来副作用。在一个函数中说明的所有变量仅在该函数中有效。

在C语言中使用代码块也使得程序结构化。代码块是一组有着逻辑关联的程序语句，可以将其视为一个单元。如下所示，将几行代码放在一对大括弧中可构成一个代码块.

```
if (x<10) {
        printf ( "too low, try again" ) ;
        reset_counter (-1) ;
```

在这个例子中，如果X小于10就执行if后面大括号中的两条语句。这两条语句与大括号一起构成了一个代码块。这两条语句相互联系，一条语句不执行则另一条也不执行。在C语言中，每条语句可以是一条单一的语句也可以是一个语句块。使用代码块更容易写出逻辑上可读性强的程序。

C语言是程序员的语言。与大多数高级计算机语言不同，C语言很少将限制强加于你。除了在最需要的情况下，一般使用C语言可以避免使用汇编代码。事实上发明C语言的动机之一就是要取代汇编语言程序设计。

汇编语言采用计算机能直接执行的直正二进制代码的符号表示。每个汇编语言操作都变换成计算机可执行的单一任务。虽然汇编语言具有给程序员最灵活和最有效地完成任务的潜力，但众所周知开发和调试的工作却十分困难。进而言之，由于汇编语言在性质上是非结构化的，最终程序往往变成“死面团”——跳转、调用和寻址纠缠在一起。这使得汇编语言难以阅读、扩充和维护。

最初，C语言用于系统程序设计。系统程序是指构成计算机操作系统或其支撑应用程序家族的一部分。例如，下面这些一般叫做系统程序：

- 操作系统
- 解释程序
- 编辑程序
- 汇编程序
- 编译程序
- 数据库管理程序

随着C语言的普及，并由于C语言的可移植性及其高效率，许多程序员开始用C语言编写各种任务的程序。实际上，因为所有计算机上都有C语言编译程序，所以在一台计算机上编写的程序，随后很容易在另一台计算机上做些修改或者不做任何修改便能编译、运行。这种可移植性节省了时间和财力。另外，C语言编译程序生成的目标代码紧凑、快速。例如，它能产生比大多数BASIC编译程序更小更快的代码。

也许C语言被用于各种类型的程序设计任务的直正原因是程序员喜欢它。C语言有汇编语言的速度和FORTH语言的可扩充性，而没有Pascal的限制。C语言程序员可以建立和维

护适合于他（她）自己的一个函数库。因为C语言允许（事实上是鼓励）分别编译，以利于管理大型项目。

本书中的许多程序都用到一个叫getnum（）的函数。C语言没有从键盘输入十进制数的内部方法。与通常看法相反，标准库函数scanf（）一般并不符合人们的使用习惯。因此，当需要从键盘上输入十进制数时，就要使用特殊函数getnum（）。getnum（9）的源程序如下：

```
getnum()  /* read a decimal number from the
             keyboard */
{
        char s[80];

        gets(s);
        return(atoi(s));
}
```

函数atoi（）是一个标准库函数，它可将数字串转换成整数。如果你的编译程序有了类似于getnum（）的函数；是否替换它，任你选择。

§1.3 简要回顾

在开始考虑各种程序设计问题和解法之前，你应该读读本章的其它部分，回顾一下C语言。如果你是一位有经验的C语言程序员，则可跳阅到第二章。

参阅附录A，它对C语言中大多关键词的语句进行了小结，并回顾了预处理程序指令。附录B介绍了本书中用到的一些标准库函数。

下面28个关键词与正规C语言语法相结合构成了C程序设计语言：

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

C语言的关键词都用小写字母表示。在C语言中，大写字母与小写字母有区别，也就是说，else是一个关键词，而ELES则不是。

1.3.1 变量——类型和说明

C语言有七种内部数据类型，如下所示：

数据类型	C语言对应的关键词
字符型	char
短整型	chort int
整　型	int
无符号整型	unsigned int
长整型	long int
浮点型	float
双精度浮点型	double

有些C语言的实现还支持无符号长整型和无符号短整型。

变量名是由一个或几个字符组成的字母串；变量名的最大长度取决于你所用的编译程序。为了清楚起见，下划线也可用作变量名的一部分（例如，first_time）。记住，在C语言中大写字母与小写字母不同——test和TEST是两个不同的变量。

所有变量在使用前必须进行说明。说明的一般形式是

类型　变量名；

例如，要说明x是一个浮点型，y是整型，以及ch是一个字符型，应该这样书写

```
float  x;
int    y;
char   ch;
```

另外对于内部类型，可以用struct和union建立内部类型的组合。也可以用typedef生成变量类型的新名称。

结构是指组合在一起并用一个名字引用的变量集合。结构说明的一般形式是

```
Struct  结构名 {
        元素 1 ;
        元素 2 ;
        ⋮
        ⋮
      } 结构变量;
```

例如，下面的结构有二个元素：name，是一个字符型数组，balance是一个浮点型数：

```
struct  client
        char name[8];
        float balance;
        } ;
```

为了引用单个结构元素，如果结构是全局的或在引用它的函数中被说明，则使用点运算符；其它情况下使用箭头运算符。

当二个或更多个变量共享同一存贮空间时，可定义为联合（union）。union的一般形式是

```
union   联合名 {
        元素 1 ;
        元素 2 ;
        ⋮
        ⋮
      }  联合变量;
```

union的元素互相覆盖。例如，下面说明union t，它在内存中的形式如图1—1所示。

```
union tom {
        char ch;
        int x;
} t;
```

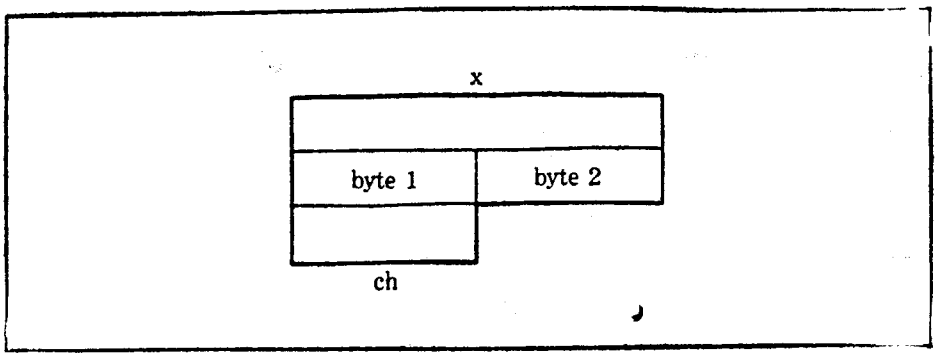

Figure 1-1. The union t in memory

如果union是全局的或在说明它的同一个函数中引用包含在union中的单个变量时，则用点操作符，其它情况下则使用箭头操作符。

C语言有三种类型修饰符——extern，register和static，它们用于改变C语言处理其后续变量的方式。如果extern修饰符放在一个变量名之前，编译程序将会知道该变量已在别处说明了。extern修饰符常用于有两个或多个文件共享同一全局变量的时候。

register修饰符仅用于局部整型或字符型变量。它使编译程序试图将其值保存在CPU的一个寄存器中。这样就使得对变量的所有引用非常迅速。本书中register变量将用于循环控制。如下面函数是用register变量做循环控制。

```
f1 ( )
 {
    register int t;
    for (t=0;t<10000;++t) {
          :
    }
}
```

static修饰符指示C语言编译程序在程序生命期内保存局部变量，而不是建立和消除它。当函数完成后返回时将放弃局部变量的值。static的使用保存了在函数调用之间变量的值。

1.3.1.1 数组

可说明前面讨论过的任何数据类型的数组。例如，要说明一个有着100个元素的整型数组x，可以写成

```
int x[100];
```

这样就建立了一个有100个元素的数组；第一个元素的标号是0，而最后一个标号是99。如下面的循环把数字0到99送入数组x：

```
for (t=0; t<100; t++) x[t]=t;
```

说明多维数组是将附加维的大小放在另一对括号中。例如，可用如下形式说明一个10乘20的整型数组

```
int x[10][20];
```

1.3.2 运算符

C语言有一个丰富的运算符集合，它可以分成以下几类：算术运算符、关系与逻辑运算符位运算符、指针运算符和杂项运算符。

1.3.2.1 算术运算符

C语言有七种算术运算符：

算术运算符	操　作
-	减法，一元负号
+	加法
*	乘法
/	除法
%	模数除法
--	减1
++	加1

这些运算符的优先级是

高最　++　--　-（-元负号）

　　*　/　%

最低　+　-

同级运算符从左到右运算。

1.3.2.2 关系与逻辑运算符

关系和逻辑运算符用来产生TRUE/FALSE（真/假）结果，二者经常一起使用。在C语言中，任何非零数生成TRUE值。但是，C语言的关系或逻辑运算符用数1代表TRUE，数0代表FALSE。下面是关系和逻辑运算符：

关系运算符	含　意
>	大于
>=	大于或等于
<	小于
<=	小于或等于
==	等于
!=	不等于

逻辑运算符	含　意
&&	AND（与）
||	OR　（或）
!	NOT（非）

这些运算符的优先级是

最高级　　!

　　> >= < <=

　　== !=

　　&&

最低级　　||

例如，下面表达式的结果为TRUE：

(100<200) &&10

1.3.2.3 位运算符

与其它大多数程序设计语言不同，C语言提供的位运算符能对变量内部实际上的位进行操作。可以使用如下运算符，但只能用于处理整型或字符型：

位运算符	含　意
&	AND (与)
|	OR (或)
^	XOR (异或)
~	反码
>>	右移
<<	左移

AND、OR和XOR的真值表如下：

&	0	1
0	0	0
1	1	1

|	0	1
0	0	1
1	1	1

^	0	1
0	0	1
1	1	0

当按位执行AND、OR和XOR操作时，这些规则用于字节中的每一位。例如，

```
  0 1 0 0   1 1 0 1
& 0 0 1 1   1 0 1 1
-----------------
  0 0 0 0   1 0 0 1

  0 1 0 0   1 1 0 1
| 0 0 1 1   1 0 1 1
-----------------
  0 1 1 1   1 1 1 1
```

```
  0 1 0 0   1 1 0 1
^ 0 0 1 1   1 0 1 1
-----------------
  0 1 1 1   0 1 1 0
```

在程序中，&、| 和^可以象其它运算符一样使用，如下所示：

```
main()
{
        char x,y,z;
        x=1; y=2; z=4;
        x=x & y;  /* x now equals zero */
        y=x | z;  /* y now equals 4 */
}
```

反码运算符（～）对字节中的每个位求反。例如，如果字符变量ch有如下位模式

0 0 1 1　1 0 0 1

那么

ch＝～ch；

将把位模式

1 1 0 0　0 1 1 0

存入ch中。

右移和左移运算符根据指定的位数将一个字节或字中的所有位向右或向左移动。当位移动时，空出的位填入零。移位运算符右边的数字说明了移位的数目。移位运算符的一般形式是

变量〉〉移位数

变量〈〈移位数

例如，给出如下位模式

0 0 1 1　1 1 0 1

右移一位变成

0 0 0 1　1 1 1 0

而左移一位产生

0 1 1 1　1 0 1 0

右移一位相当于除以 2，而左移一位相当于乘以 2。例如，下面的程序段，x中的值首先乘以 2，然后再除以 2。

```
int  x;
x=10;
x=x〈〈1;
x=x〉〉1;
```

由于机器中的负数表示方式，所以当用移位运算符进行乘法或除法运算时必须注意：

将1移入最高有效位会使计算机认为这个数是负数。

请记住：位运算符修改变量的值，它们与逻辑和关系运算符不同，后者产生TRUE或FALSE结果。

位运算的优先级如下：

最高级 ～

〉〉 〈〈

&

∧

最低级 ！

1.3.2.4 指针运算符

指针运算符在C语言中是重要的，这不仅仅因为它们允许把串、数组和结构传递给函数，而且它们允许C语言的函数修改它们的调用参量。两个指针运算符是&和*。（不巧，这两个运算符与乘法运算和位AND运算使用了相同的符号，而它们之间是毫不相关的。）

&运算符加在变量前面则给出该变量的地址。例如，如果整型变量x在内存的地址是2000，那么

```
y= &x;
```

把值2000送给y。&可读作“…的地址”。例如上边语句可读作“把x的地址送给y”。

*运算符取其后面的变量的值，并将该值作为内存中信息的地址。例如，

```
y= &x;
y*=100;
```

把值100送入x。*可读作“在地址”。上述例子可读作“在地址”。上述例子可读成“将值100存入地址y中”。*运算符也可以用在赋值语句的右边。例如，

```
y= &x;
y*=100;
z=*y/10;
```

将值10存入z中。

这二个运算符之所以称为指针运算符，是因为它们被设计成靠指针变量进行工作。一个指针变量记录着另一个变量的地址；实际上，它“指向”那个变量。如图1—2所示。

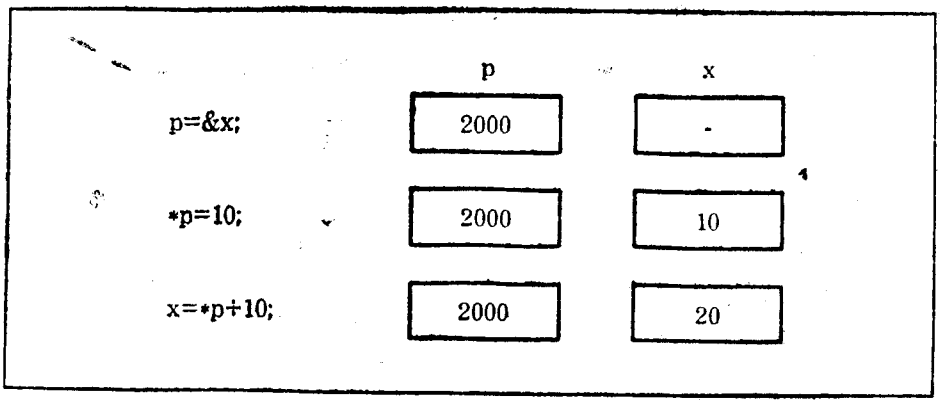

Figure 1-2. Pointer operations for character pointer p and integer x, with x at memory location 2000

1.3.2.5 赋值运算符

在C语言中，赋值运算符是一个等号。然而，C语言允许用一个方便的“简写”形式来代替一般形式的赋值。一般形式为

变量1＝变量1的运算符表达式

下面是两个例子：

```
x=x+10;
y=y/z;
```

这类赋值可以简写成如下形式

变量1　运算符＝表达式

这样，上述两个例子可简写成

```
x+=10;
y/=z;
```

有经验的C语言程序员编写C语言程序时经常使用简写形式，所以你应该熟悉它。

1.3.2.6 ？运算符

？运算符是个三元运算符，它用来替代一般类型的if语句

```
if表达式1　thenx=表达式2
  else  x=表达式3*
```

？运算符的一般形式是

变量＝表达式1？表达式2：表达式3；如果表达式1是TRUE，那么将表达式2的值赋给变量；否则将表达式3的值赋给变量。例如，

```
x= (y<10) ?20:40;
```

当y小于10时，把20赋给x；如果y不小于10，则将40赋给x。

之所以使用？运算符是因为C语言编译程序对这种类型的语句可以生成非常有效的代码——比等价的if/else语句快得多。

1.3.2.7 杂项运算符

．（点）运算符和→（箭头）运算符用于引用结构和联合中的单个元素。当结构或联合是全局的，或者当引用的代码与结构或联合的说明在同一个函数中时，使用点运算符；当把结构或联合传递给一个函数，或者当只能得到指向结构或联合的指针时，则使用箭头运算符。例如，给出一个全局结构

```
struct tom {
        char ch;
        float w;
} clyde;
```

可以写出

```
clyde.w=123.23;
```

将值123.23赋给结构clyde的元素W。

，（逗号）运算符通常用于for语句中。它使得一系列运算被执行。当逗号运算符用于

*原文误为2，译者订正为3。

赋值语句的右边时，整个表达式的值就是被逗号分隔的表中最后一个表达式的值。例如，执行下面语句以后

```
y=10;
x= (y=y-5,25/y) ;
```

x的值为5，因为y的原始值10减去5，再去除25，结果为5。

虽然sizeof也是一个关键词，但它是一个编译时的运算符，用来确定数据类型的字节大小，包括用户定义的结构和联合。例如，

```
int x;
printf ("%d",sizeof(x) ) ;
```

在许多微机上都显示出数字 2 。

圆括号被视为是提高了括号内运算优先级的运算符。方括号执行数组索引。

强制（cast）运算符号是一种特殊运算符，它将一种数据类型强制性地转换成另一种数据类型。其一般形式为

(类型) 变量

例如，为了在调用sqrt ( ) 中使用整型变量count（sqrt ( ) 是C语言标准库中求平方根的例行程序，它要求一个浮点型参数），强制运算符可将count处理成浮点型变量：

```
float y;
int count;
count=10;
y=sqrt ( (float) count) ;
```

表1—1列出全部C语言运算符的优先级。注意所有运算符——除去一元运算符和？，都从左向右结合。一元运算符（*、&和—）以及？运算符则从右向左结合。

Table 1-1. The Precedence of C Operators

Highest	( ) [ ] — .
	! ~ ++ — — — (type) * & sizeof
	* / %
	+ —
	<< >>
	< <= > >=
	== !=
	&
	^
	|
	&&
	||
	? :
	= += -= *= /= %= >>= <<= &= ^= |=
Lowest	,

1.3.3 函数

一个C语言程序是一个或多个用户定义的函数的集合。其中的一个函数必须是main( )，因为程序从这个函数开始执行。在历史上，main ( ) 通常是程序中的第一个函数，但实际上它可以在任何地方出现。

C语言函数的一般形式是

```
函数名 (参数表)
参数说明
 {
函数体
}
```

如果函数没有参数，则不必有参数说明。当遇到最后一个大括弧时，所有函数都自动终止并返回到调用过程。也可以使用return语句强迫函数提前返回。

所有函数都返回一个值。如果一个return语句是函数的一部分，那么函数的值就是return语句中的值。如果没有return语句，那么函数返回零。例如，

```
f1 ( )
 {
        int x;
        x=100;
        return (x/10) ;
}
```

返回值是10，而

```
f2 ( )
 {
        int x;
        x=100;
        x=x/10;
}
```

返回值是0，因为没有碰上显式的return语句。

由于所有函数都有值，所以它们可以用在任何算术语句中。例如，才入门的C语言程序员倾向于编写如下的代码：

```
x=sqrt (y) ;
z=sin (x) ;
```

而更有经验的程序员却会写成

```
z=sin (sqrt (y) ) ;
```

请记住：程序为了确定函数的值，就必须执行它。这意味着下列代码要不断地从键盘上读被按下的键，直到u被按下为止：

```
while ( (ch=getchar ( ) ) !='u') ;
```

这条代码起作用是因为必须执行getchar ( ) 以确定它的值就是被键入的字符。

1.3.3.1 函数与变量类型之间的关系