ANSI C

彭光泽 编译
杨旭明 主审

C语言应用

C LANGUAGE

电子科技大学出版社

ANSI C

C 语言应用

彭光泽 编译
杨旭明 主审

电子科技大学出版社

内容提要

本书全面地介绍了C语言应用于实际中的具体做法及其原理。书中所阐述的灵活运用C语言的许多处理手法和诀窍，对提高C语言程序设计人员的实际工作能力很有用处。

本书内容符合ANSI规范，大量的程序举例不仅与各种最新的处理系统相对应，而且还可供读者直接使用。

ANSI C

C语言应用

彭光泽 编译

杨旭明 主审

*

电子科技大学出版社出版

（中国成都建设北路二段四号）

电子科技大学出版社激光照排中心照排

国营成都市农垦印刷厂胶印

四川省新华书店经销

*

开本 787×1092 1/16 印张 16.625 字数 400千字

版次 1992年9月第一版 印次 1992年9月第一次印刷

印数 1—6000册

中国标准书号 ISBN 7-81016-418-X/TP·32

[川]016 （15452·181） 定价（压膜）7.00元

编译者序

随着C语言的迅速普及，利用C语言编程的人日渐增多。各种版本的C语言读物，使众多读者对C语言已经有了相当程度的认识。不过，即便是那些较为熟练地掌握了C语言程序设计技术的人，在实际工作中也还会遇到一些单靠C语言本身的知识所难以逾越的障碍。因此，掌握一些能够解决C语言实际应用过程中所面临问题的手法与诀窍，将是非常必要和有用的。我们编译《C语言应用》一书，目的是帮助从事C语言编程的读者，更好地掌握C语言的编程技巧。本书特色在于从理论与实践两个方面，向人们介绍了在C语言的应用过程中解决实际问题的具体做法。

本书所选择的内容，虽不能说已覆盖了关于C语言应用的全部知识，但却基本上包罗了大多数C语言程序员在C语言的应用实践中所必将直面的问题。为了帮助读者能够真正做到对C语言运用自如，书中还罗列了丰富的程序举例，对诸如与汇编程序的连接、汉字处理、局部区域网络中的程序设计、高速化、UNIX环境、EUC代码等等问题进行了详尽的阐述与演示。程序举例中的许多有用的处理手法及取样程序本身，均可供读者在实际工作中进行直接套用。

当然，解决C语言应用问题的实际工作能力，往往来源于对计算机系统内部结构的深刻理解。本书的另一特色，就是着眼点并不局限于C语言自身范畴内的种种运用形式，而是在于把C语言当作一个窗口，来展示灿烂多彩的计算机应用世界。因而，本书虽然属于有关C语言应用程序设计的读物，但它同时也包括了硬件、操作系统等有关计算机的实际知识。这一点，对于缺乏硬件知识的读者理解书中的内容是大有裨益的。

另外，自从ANSI规范推出之后，各种版本的C语言和各种最新的计算机处理系统都竞相与之对应。本书亦无例外地完全遵循ANSI规范，书中内容可与各种最新的处理系统相适应。

值得一提的是，电子科技大学杨旭明教授在百忙之中详细地审阅了本书的各个章节，提出了许多宝贵意见。在此谨表示衷心的感谢。

彭光泽

一九九二·五·于蓉

目　录

第 1 章　程序开发基础知识

C 语言的各种教科书，都是以最基本的编程知识为中心来进行解释的。这些基本程序的执行，并不依赖于特定的计算机和编译程序。但是，在实际程序开发过程中，却会遇到许多单靠 C 语言的知识所不能解决的问题。特别是在微机上进行应用程序的开发时，由于要求画面的高速控制和高度的用户接口，因此比起 C 语言本身的知识来，如果对特定计算机系统不甚了解，那么是很难编制出经得起实践考验的程序来的。本章以微机上的开发为中心，介绍编制实用程序所需具备的基础知识。

§1.1　与 CPU 相关的程序设计

C 语言如同汇编程序，可以对 CPU 进行细微的控制。反过来讲，一旦对 CPU 的体系结构理解不深，则不能编制出实用的程序。在 C 语言和 CPU 的关系中主要出现的问题有如下几点：

1. int 类型变量的位长差异
2. 指针变量的位长差异
3. 寄存器配置的差异

下面来讨论这些问题。

★ int 类型变量的位长差异

int 类型变量的位长，一般被分配的是可同时进行 PUSH、POP 的寄存器位长。但是，其值因 CPU 的不同而迥异。表 1-1 显示了具有代表性的 CPU 上 int 的位长。如果是 68000 等工作站或是小型机中所使用的 CPU，则 int 为 32 位，亦即 long。如果是 8086 或 80286 等微机中所使用的 CPU，则 int 为 16 位。因此，那些使用位字段* 的程序，就必须对其源码的一部分进行重写。

	8080 系统 CPU	6800 系统 CPU	8086 系统 CPU	68000 系统 CPU	80386 系统 CPU
CPU	8080 Z80 等	6800 6809 等	8088，8086 80188，80186 80286　等	68000 68010 68020 等	80386
int 的位长	16	16	16	32	32 MS-DOS 中为 16

表 1-1　典型 CPU 上 int 类型变量的位长

在下面的程序举例中，编程者想根据 for 来增加变量 i 的值，以便把 i 作为循环计数器使用。但是，当 int 的位长为 16 位时，结果会产生溢出，一旦 i 的值超过 32768，程序就会出现非正常现象。

---

* 位字段只在 int 类型时有效

```
int     i;
        ⋮
for(i=0; i<60000; ++i)
        {                    ——此值若为 16 位则会发生溢出，
        ⋮                      若为 32 位则不会发生溢出
```

★ 指针变量的位长差异

指针的操作与地址的处理是有区别的。但是，成为其操作对象的数值，却明显地是地址本身。因而，如果把地址值转换成 int 等其他整数值来使用，就会产生问题。

UNIX 上的 C 语言编译程序，无论 int 或指针均为 long (32 位)。因此，那些对指针类型和 int 类型只作粗略处理的程序也能顺利通过。然而，如果是 MS-DOS 和 16 位系列 CPU 的 C 语言，则必须对这些类型进行严格的校验，否则，有时程序不会运行。另外，UNIX 上的源表中，往往没有对函数的返回值作明确的说明，而是假设 int 将返回。但是，int 的位长在 UNIX 和 MS-DOS 上的 C 语言中是不相同的，这就成了问题的起因。如果要把以 UNIX 上的 C 语言所编写的源表移植到 MS-DOS 上，就应当特别地注意到这一点。

当然，指针变量的指针值（地址）是保存在 int 类型变量中的，可当 int 的位长为 16 位而指针变量的位长为 32 位时，指针值（下面 p 值）将不能得到正确的保存。

```
char        *p;
int         px;
            ⋮
px=p;……保存地址
            ⋮
p=px;……返回地址
            ⋮
```

当在 MS-DOS 上的 C 编译程序中使用大模式时，便会出现上述问题（关于大模式请参照下节）。所以，在进行复杂的指针运算时，编码需格外谨慎。

★ 寄存器配置的差异

CPU 的体系结构是各不相同的，其寄存器的配置区别很大。因此，在使用寄存器变量时往往会发现这种情况，即，程序表中明明分配有寄存器变量，可实际上编译程序还是自动地将数据分配到了存储器上。寄存器变量在 8086 系统 CPU 中可使用 2 个（通常使用 SI 寄存器和 DI 寄存器），而在 68000 系统 CPU 中可使用 7～8 个。

寄存器变量的上述情况，在同汇编程序接口的时候应给予充分注意。特别是在编制与汇编程序相连接的程序时，还是尽量不使用寄存器变量为好。另外，也需要预先意识到寄存器的位长问题，否则，在汇编程序的编码时会出现麻烦。

§1.2 8086 系统 CPU 与指针变量的处理

在许多 16 位微机所采用的 8086 系统 CPU 中，指针的问题变得更为复杂。因此，下面首先详细阐述 8086 系统 CPU 的地址表达方式以及使用 C 语言编程时遇到的“存储模式”问题。

★ 8086 系统 CPU 的地址表达方式

在进行 8086 系统 CPU 中的程序设计时，肯定会碰到存储模式的问题。这是因为，8086 系统 CPU 的地址，是根据一个叫做段的逻辑空间来进行分段表达的。最早，建立段的概念，是为了在多任务操作系统中进行多进程管理，而无需把这一概念用于单一任务的操作系统。

在 8086 系统 CPU 中，可以单纯地把段理解为“地址的高位”。地址的低位称为偏移，存储器上的地址就是由这个偏移和段来决定的。

这种由段和偏移所表达出来的地址叫做**逻辑地址**。逻辑地址说明了一种逻辑性的地址指定方法，但它却不能代表实际分配到存储器上的地址（称为**物理地址**）。实际上的物理地址，是偏移和段相加后的值。但是，这里所说的相加后的值，并非是随便将两个值加在一起就能够得到的。简单地讲，就是要将段值往高位错移 4 位，然后作为 20 位的值与偏移值相加，这才能获得实际的物理地址。这是因为，8086 系统 CPU（或虚拟 8086 方式）的物理地址长只有 20 位（即可以处理 1M 字节的地址空间），而表达该地址空间的段和偏移的寄存器长却各有 16 位，所以，非进行上述麻烦的计算不可。

另外，仅用 C 语言编制程序时（不处理寄存器变量的情况下），可不对寄存器加以考虑。但是，在编制使用系统调用的程序和与汇编程序连接的程序时，以及在利用排错程序时，就必须预先了解清楚各类寄存器及其作用。

在 C 编译程序中，有的是将物理地址作为指针来直接传递的（地址以 20 位表达，具有 long 的类型），有的却只能按逻辑地址使用（地址成为 32 位的 long 类型）。Microsoft C（下称 MS-C）和 Turbo C 等较新的处理系统便属于后者。

根据编译程序的存储模式的不同，有时指针运算只限于偏移部分。这时，程序的执行速度变得非常高，被编译的程序尺寸也变得非常小。可是，在这种情况下如果不经常考虑到段界处的问题（计算偏移后出现位溢），那么在处理大数据时就容易出错。另一方面，为了解决上述问题，有的处理系统提供了以库函数进行指针运算的存储模式。这时，段界处的问题没有了，指针的运算可不再顾虑 8086・CPU 的特性。可是相反，由于每当指针运算的时候就要进行一次段与偏移的计算，因此那些要求执行速度的应用程序又该多加留意了。

★ 存储模式的概念

存储模式将 8086・CPU 的存储器的使用方法分成几个典型的部分。计算机内的数据区域分为程序进入的**代码区域**和数据进入的**数据区域**。除此之外，还有专为使用栈而开设的**栈区域**。存储模式将对这些区域进行段处理并指定怎样将其分成数据或代码的段空间。

典型的存储模式有：

小模式（Small Model）

将代码区域、数据区域分别纳入一个个的段内。程序执行过程中段值不发生变化。也就是说，无论代码或数据，指针的位长均为 16 位。这表明代码在 64K 字节以内，数据也在 64K 字节以内。

中模式（Medium Model）

代码区域的段在程序执行过程中经常发生变化。也就是说，指向程序（C 语言中为函数）的指针的位长为 32 位。指向数据的指针的位长为 16 位。当然，在程序执行过程中，指向数据的指针的段值是不会变化的。

大模式（Large Model）

数据区域的段在程序执行过程中经常发生变化。亦即，指向数据（C 语言中为变量）的指针的位长为 32 位。指向程序的指针的位长也为 32 位。当然，在程序执行过程中，该指针的段值不发生变化。

以上是具有代表性的存储模式。最近还出现了其他一些存储模式，例如**紧致模式**（与中模式相反，代码段值固定、数据段值可变的存储模式）等。其中，根据处理系统的不同，有的名称相异，有的编译程序的功能不一。譬如在 MS-C 中，支持一种叫做**巨模式**（huge）的存储模式。该巨模式由编译程序的代码来随时管理数据段的值，它可以存取 64K 字节以上大小的一个数据的“块”（比如 1 个数组）。这与大模式及其汇编程序级的数据处理是完全一致的，但其思维方式却不同。就是说，巨模式是由特定的编译程序所支持的存储模式，与严密说法上的存储模式稍有区别。

各厂家 C 编译程序的存储模式，大致归纳在后述表 1-3 中，请读者参照。

★ 不同存储模式时的指针位长比较

在 8086・CPU 中，C 语言的指针位长是程序设计时的一个重要因素。“char * data”变量的位长，在不同存储模式的情况下是要发生变化的，同时，根据其对象是函数或是变量的不同也要发生变化，因此不可等闲视之。表 1-2 中，列出了几个典型的存储模式，并且对不同存储模式下的指针的位长作了对照比较。表中 FAR 为 FAR 指针，代表位长为 32 位的指针。NEAR 代表位长为 16 位的指针。

说明 \ 存储模式		小	中	紧致	大
针对数据的指针	〈type〉 *	16	16	32	32
	〈type〉 near *	16	16	16	16
	〈type〉 far *	32	32	32	32
针对函数的指针	〈type〉 (*) ( )	16	32	16	32
	〈type〉 (near *) ( )	16	16	16	16
	〈type〉 (far *) ( )	32	32	32	32

〈type〉 ……char，int，long，float，double 等

表 1-2　不同存储模式下指针变量的位长比较

在以下几种场合，如果使用大模式，编程就会轻松一些：

・编制大规模程序时

・与汇编程序的接口成问题时

・伴有硬件的直接存取时

相反，如果程序的规模较小，处理的数据也不多，并且无需直接处理硬件的话，那么还是以小模式编程为好。至于其他的存储模式，可分别根据各自的情况选择使用。

★ 直接存取实地址区域的程序

在 PC-9801 系列中，如果向被称为视频 RAM（V-RAM）的存储器上的特定地址写入字符

码及其属性，硬件便会在读取该区域后将字符显示于画面。

向画面书写字符的程序，以 MS-DOS 的系统调用或 C 语言的 printf 函数都可描述。但是，这些方法不能快速地将字符书写于画面。如果想以更快的速度进行画面显示，则可使用下面的 C 语言程序，一个字符一个字符地快速显示于画面。

如果按照 printf 函数的一般顺序，则应显示于画面的数据的流程是：

printf 函数→MS-DOS 的字符输出系统调用→PC-9801 的 ROM 内程序→V-RAM

如此是相当费时的。而以下所要介绍的程序，却可直接向 V-RAM 书写字符数据（当然，其代价是牺牲了移植性）。

PC-9801 的 V-RAM 区域，在 CPU 的实地址中是从 A0000h 号地址起始的。决定色彩与属性的区域是从 A2000h 号地址起始的。各机种的使用手册中都有关于存储变换的说明，据此可找到文本 V-RAM 的起始地址。

在清单 1-1 的程序中，被指定的 1 字符将按照指定的属性而被写入指定的位置。

```
1:  /*
2:      Display 1-characters for TEXT-RAM Directory
3:  */
4:                  行的值(0～24)
5:                  |
6:  void        dspkl(int law, int col, unsigned int moji, int color _ f, int color _ m)
7:                        |
                          位的值(0～79)
8:      /* color _ f                */
9:      /* 字符色 (0 to 7)          */
10:     /* 0 ..... 黑               */
11:     /* 1 ..... 蓝               */
12:     /* 2 ..... 红               */
13:     /* 3 ..... 紫               */
14:     /* 4 ..... 绿               */
15:     /* 5 ..... 浅蓝             */
16:     /* 6 ..... 黄               */
17:     /* 7 ..... 白               */
18:
19:     /* color _ m                                    */
20:     /* 字符方式 (0 to 7)                            */
21:     /* 0 ..... Normal                               */
22:     /* 1 ..... Brink                                */
23:     /* 2 ..... Reverse                              */
24:     /* 3 ..... Brink & Reverse                      */
25:     /* 4 ..... Under-Line                           */
26:     /* 5 ..... Under-Line & Brink                   */
27:     /* 6 ..... Under-Line & Reverse                 */
28:     /* 7 ..... Under-Line, Reverse & Brink          */
29:
30:     {
31:     unsigned long adr _ tmp;
```

```
32:     int  adr_val;
33:     char far *adr_ch00;
34:     char far *adr_ch01;
35:     char far *adr_ch02;
36:     char far *adr_ch03;
37:     char far *adr_at00;
38:     char far *adr_at01;
39:     char far *adr_at02;
40:     char far *adr_at03;
41:
42:
43:     adr_tmp=160L*(unsigned long) law+2L*(unsigned long) col;……偏移地址计算
44:                                  段地址        偏移地址
45:     adr_ch00=(char far *) (0xA0000000L+adr_tmp);
46:     adr_at00=(char far *) (0xA2000000L+adr_tmp);
47:
48:     adr_val=(color_f<<5) | 0x01 | (color_m<<1);……字符色/字符方式的决定
49:
50:     if(moji<0x0100)……非汉字码时的处理
51:         {
52:         adr_ch01=adr_ch00;
53:         adr_ch01++;
54:         adr_at01=adr_at00;
55:         adr_at01++;
56:         *adr_ch00=moji;
57:         *adr_ch01=0;
58:         *adr_at00=adr_val;
59:         *adr_at01=0xFF;
60:         }
61:     else……汉字码时的处理
62:         {
63:         moji=sjis2jis(moji);……移位 JIS→JIS 的转换
64:         adr_ch01=adr_ch00+1;
65:         adr_ch02=adr_ch00+2;
66:         adr_ch03=adr_ch00+3;
67:         adr_at01=adr_at00+1;
68:         adr_at02=adr_at00+2;
69:         adr_at03=adr_at00+3;
70:
71:         *adr_ch00=*adr_ch02=(moji>>8)-' ';
72:         *adr_ch01=moji;
73:         *adr_ch03=moji | 0x80;
74:
75:         *adr_at01=*adr_at03=0xFF;
```

实地址指针变量

存入地址的计算

```
76:             *adr_at00=*adr_at02=adr_val;
77:             }
78:         }
79:
80:
81:  sjis2jis(int c)……移位 JIS→JIS 转换程序(见第 3 章)
82:  {
83:         int hi, lo;
84:
85:         hi=(c)>8) & 0xff;
86:         lo=c & 0xff;
87:         hi-=(hi(=0x9f) ? 0x71 : 0xb1;
88:         hi=hi*2+1;
89:         if (lo)0x7f)
90:             lo--;
91:         if (lo)=0x9e) {
92:             lo-=0x7d;
93:             hi++;
94:         } else
95:             lo-=0x1f;
96:         return hi((8 | lo;
97:  }
```

清单 1-1　DISP.C

此程序是以使用 MS-C（或 Turbo C）为前提而编写的。指针的操作形式是“段：偏移”（第 45 行、46 行）。

以下是专为使用以上函数的标题文件。

```
1:  /*
2:                  Headers for
3:                  Display 1-characters for TEXT-RAM Directory
4:  */
5:
6:
7:  #define         C_BALCK         0
8:  #define         C_BLUE          1
9:  #define         C_RED           2
10: #define         C_MAGENTA       3
11: #define         C_GREEN         4
12: #define         C_SKYBLUE       5
13: #define         C_YELLOW        6
14: #define         C_WHITE         7
15:
16: #define         M_BLINK         (1)
17: #define         M_REVERSE       (2)
18: #define         M_UNDERLINE     (4)
```

```
#define          M_NORMAL          (0)

#ifdef  LINT_ARGS
void     dspk1(unsigned, unsigned, unsigned, unsigned, unsigned)
#else    /* LINT_ARGS */
void     dspk1( );
#endif   /* LINT_ARGS */

#define     NORMAL     C_WHITE, M_NORMAL
```

清单 1-2　DISP. H

使用这个程序在画面的左上方写一个“A”字，属性定为闪烁和下划线。

```
/*
    TEXT RAM Direct-Access procedure TEST.
*/

#include  <stdio. h>
#include  "disp. h"

main( )
{
dspk1(0, 0, (unsigned int ) ('A'), C_GREEN, M_BLINK | M_UNDERLINE);
}
```

清单 1-3　TEST1. C

编译该程序。以下是编译用的批量文件。

```
                  —AS(小)、—AM(中)均可
                 /
CL —AL  —Gs —Od —Zde  —FeDISPTEST DISP. C TEST1. C
                 \
                  —可不指定
```

清单 1-4　TESTD. BAT

这个程序执行后，便可实现快速画面显示。此程序是以大模式进行编译的。由于指定了FAR 指针，因此是带有 MS-C 的“—Zde”选择进行编译的（MS-C Ver4. 0 以后为缺席指定）。这就是说，即使以大模式以外的存储模式进行编译，效果也是一样的。相反，如果以采用大模式进行编译为前提的话，那么，去掉 FAR 这个关键字，即使不带“—Zde”选择进行编译，也会得到相同结果。

表 1-3 显示了 MS-DOS 版主要 C 编译程序的存储模式指定选择。如前所述，编制程序时，必须搞清楚各存储模式的特征。选择什么样的存储模式，程序的结构和描述就有什么样的变化。所以，在程序设计之前，就应当慎重地考虑好到底选用哪一种存储模式。

从表 1-3 来看，最近各编译程序基本上准备的都是相同的存储模式。其中，Turbo C 的超小模式，是用于编制“. COM”形式的执行文件的，其代码和数据的大小控制在 64K 字节以内。

存储模式	代码尺寸	数据尺寸	MS-C	Quick C	Turbo C	LSIC-86	Lattice C
Tiny	64Kbyte		-AT	——	-mt	——	——
Small	64Kbyte	64Kbyte	-AS	-AS	-ms	-ms	-ms
Mediaum	1Mbyte	64Kbyte	-AM	-AM	-mm	-mp	-mp
Compact	64Kbyte	1Mbyte	-AC	-AC	-mc	-md	-md
Large	1Mbyte	1Mbyte	-AL	-AL	-ml	-ml	-ml
Huge	1Mbyte	1Mbyte	-AH	-AH	-mh	——	-mh

表 1-3 C编译程序的存储模式指定选择

§1.3 与操作系统相关的程序设计

提起操作系统，就会联想到 MS-DOS 以及 UNIX 这类名称。不过，许多人把“dir”或“ls”这类指令误认为就是操作系统本身。其实，这些指令永远都是操作系统所具有的指令（用户接口），而并非原本意义上的操作系统本身。

从程序级来看，操作系统的实体不在于这类指令，而在于提供给应用程序的辅助程序。以下，对利用了这些辅助程序的程序进行讲解。

★ 什么是系统调用

可以通过应用程序加以使用的辅助程序，便叫做**系统调用**，它是操作系统中的一种功能。操作系统在大多数情况下，都是驻留在存储器上的。因此，应用程序随时都可以使用这些辅助程序。

在 UNIX 的 C 编译程序中，这些辅助程序通过库函数而得到支持。可是在微机用的 BASIC 语言等高级语言中，却未准备有直接取出这些系统调用的指令。所以，一般都是以汇编程序制作取出系统调用的程序，然后再通过高级语言将其调出并加以利用。

与之相反，在目前大多数 MS-DOS 上的 C 编译程序中，都准备有使用这些系统调用的程序库。只是，针对磁盘目录区的存取等几个系统调用还未通过库函数得到支持，因此有必要利用软件中断来直接取出系统调用（有关目录操作将在第 4 章中详细解说）。另外，这些程序库的内容，在各 C 编译程序中都是大致相同的。凡利用这类程序库的 C 语言源表，只要是在 MS-DOS 上运行，就具有相当高的兼容性。

★ 使用 MS-DOS 系统调用的程序

在 MS-DOS 中，由于使用了系统调用，因此可以针对各种各样的硬件进行存取。并且，它还支持命令总线的解析以及当前系统状态的把握。不用说，系统内藏时间的设定和当前时间的获取等，都包括在这些系统调用的功能之中。

从应用的角度来使用系统调用时，需要采用机器语言指令“INT”（INTerrupt：中断）。作为生成 INT 指令的库函数，可列举如下，它们都是具有代表性的（详见第 4 章）。

int86( )，int86x( )，intdos( )，intdosx( )，bdos( )

要使用这些库，必须嵌入 DOS. H 来作为标题。此 DOS. H 有一个定义叫做寄存器结构。在