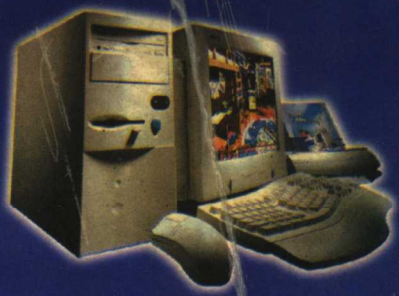
C语言进阶诀窍



陈 童 巩丹宏 编著

西安交通大学出版社

为什么有些人学过C语言后，能写出CCED之类的程序？

内容简介

本书从实用的角度出发，根据作者多年的程序设计经验，系统地介绍了C语言的编程技巧，并深入结合计算机软硬件知识，讲解了如何利用C语言编写高水平的应用程序。

本书内容丰富，逻辑性强，文字流畅，通俗易懂。每章都给出了丰富的实例，便于读者掌握各章内容。这些实例也可作为编写C程序时的子程序库。本书可作为C语言读者的教材，又可作为C程序设计者的使用手册。

通过本书的学习，定会提高读者的C语言编程能力。

(陕)新登字007号

C语言进阶诀窍

陈　童
巩丹宏　编著

责任编辑　林　全

责任校对　陈　宏

*

西安交通大学出版社出版发行

西安市咸宁西路28号　邮政编码:710049　电话:(029)3268316

西安华宇印刷厂印装

各地新华书店经销

*

开本:787×1092　1/16　印张:28　字数:685千字

1997年3月第1版　1997年3月第1次印刷

印数:1——5000

ISBN7-5605-0900-2/TP·152　定价:35.00元

前　言

近年来，功能强大的C语言成为最流行的程序设计语言，学习C语言的人难计其数。然而，大多数人学过C语言后，并没有发挥C语言和计算机的强大功能，他们通常只使用C语言非常简单的功能，根本不能利用C语言进行应用程序设计。这其中的原因何在呢？为什么有些人学过C语言后，能写出CCED之类的好程序，而有些人学过C语言之后，简直与没学过一样呢？其中的一个原因是，C语言一些新的数据类型和结构，如指针（POINTER），结构（STRUCT）等，给学习带来困难；另外计算机知识的缺乏限制了对C语言的使用；然而，最重要的一点是：实践太少。从本质上看，前面提到的两点都是可以通过实践来加强的。因而，学好C语言的最好方法是：结合计算机知识在实践中不断学习。本书正是基于这种考虑而编写的。它结合计算机知识，介绍C语言的编程技巧。书中所附的大量实例都很典型，具有一定的代表性，并经过上机调试通过。

全书共分三篇。第一篇C语言快速入门和系统功能的调用方法。第二篇主要结合计算机知识讲解如何利用C语言编写高水平应用程序，包括输入与输出，磁盘与文件，键盘与鼠标，图形与汉字，图象与色彩，声响与音乐等，另外在综合技术中，还介绍了混合编程的方法，下拉菜单的编程，以及在C语言中使用扩充内存和扩展内存的方法等等。第三篇中，讲解了几个大一点的示例程序。CTT是一个集成实用工具集，用下拉式菜单驱动，其功能包括文件拷贝、移动、改名、删除、磁盘编辑、音乐等等，而且CTT还实现了OS SHELL，背景音乐，去除CMOS口令等功能。俄罗斯方块是大家比较熟悉的游戏，实例篇的“疯狂俄罗斯”游戏是俄罗斯方块的一个微机版本，读过它后，您会发现，其实C语言并不难学。EDITOR是一个功能比较全面的编辑器。最后是一个商业级的游戏——麻将，运行这个游戏，你会发现它和流行的麻将相比毫不逊色。学过本书后，你也可以写出这样的程序，成为一个高水平的C程序员。

作者

目　录

基础篇

应用篇

实例篇

附 录

基础篇

第 1 章　C 语言快览

C 语言是由贝尔实验室的 D.M.Ritchie 和 K.Thompson 在 1973 年共同开发的。由于它功能强，效率高，很快成为世界上最流行的程序设计语言。和其它程序设计语言相比，C 语言有以下优点：

(1)它是一种结构化语言，很适合大型程序的模块化设计。

(2)适合系统软件和应用软件的开发。

(3)运算符丰富，程序设计简洁，灵活。

(4)具有一定的汇编语言特征，可以编写出功能强大的应用程序。

正是由于这 些优点，使得 C 语言风靡全世界，成为微机、小型机、中型机、大型机和巨型机上的通用程序设计语言。

1.1　C 语言概述

在开始讲解 C 语言之前，先让我们来看一个 C 程序：

```
/*
* * 程序 1-1 C 语言示例程序
*/
#include <stdio.h>
main()
{
    int i;
    for(i=0;i<10;i++)
    printf("%d* %d = %d \n",i,i,sub(i));
    return(0);
}
int sub(int i)
{
```

```
    return(i * i);
}
```

从上面的程序可以看出,C 语言程序是由一个 main()函数和一些其它函数组成的。其它函数可以没有,而 main 函数是用来标识主程序的,它是整个程序的入口,也就是说,程序执行时,将从这里开始。函数是由一对大括号标识开始和结束的,大括号里面是函数体。函数体内是 C 语言的语句,C 语言语句是用分号";"隔开的。在程序开始的 # include 语句是头文件声明语句,头文件可以是 C 编译系统提供的标准头文件,也可以是用户自己编写的头文件,头文件中的内容一般为函数声明,宏定义等。

1.1.1 基本数据类型

(1)整数

整型数是用两个自己表示的,有符号整数大小范围为 - 32768 ~ + 32767,无符号整数的范围为 0 ~ 65535。在 C 语言中整数可以用十进制、八进制和十六进制三种形式来表示。若整数书写时,前面加上一个 0 则表示此数为八进制数,若前面加上 0X 或 0x 则为十六进制数,否则为十进制数。

在声明整型变量时,使用 int 来说明,例如:

```
int i;
int i,j;
int abc123;
```

(2)实数

实数由整数部分、小数部分和小数点组成,也可以用科学计数法表示(使用 e 或 E),下面都是 C 语言中有效的实数表达式:

```
123.456
1.1E-2
1.1e-2
```

C 语言中实数分浮点数和双精度数,微机上浮点数用 4 个字节表示,双精度数用 8 个字节表示。在声明浮点数变量和双精度变量时用 float 和 double 关键字:

```
float temp;
double d = 3.141 592 653 59;
```

(3)字符

C 语言中字符是用一个字节表示的,字符使用单引号括起,如'A'. C 语言中使用"\"来引出一些特殊的字符,如:

```
'\0'    空字符
'\t'    制表符
'\n'    回车符
'\"'    双引号
'\\'    反斜线本身
```

C 语言中声明字符变量时,使用 char 关键字:

```
char    ch;
```

(4)字符串

字符串是用双引号引起来的一组字符，如“this is a string”，字符串除了本身包含的字符外，还有一个空字符‘\0’跟在字符串末尾，用来判断字符串的结束。

1.1.2 运算符

(1)算术运算符

算术运算符共有5个，它们是：

+	加法运算符	/	除法运算符
-	减法运算符	%	求余运算符
*	乘法运算符		

注意：求余运算只用于整型和字符型。

(2)关系运算符

关系运算符共有6种，它们是：

>	大于	<=	小于等于
<	小于	==	等于
>=	大于等于	!=	不等于

(3)逻辑运算符

逻辑运算符共有3种，它们是：

&&	逻辑与运算符	!	逻辑非运算符
||	逻辑或运算符		

(4)位运算符

位运算符共有6种，它们是：

&	按位与运算符	^	异或运算符
|	按位或运算符	<<	左移位运算符
~	按位求反运算符	>>	右移位运算符

(5)一元运算符

C语言中有一些特殊的运算符，如*，&，++，--以及sizeof等，下面举例说明它们的含义：

*p	取得指针p所指地址中的内容	a++	先存取a值，再将a值加1
&a	取得v的地址	--a	先将a值减1，再存取a值
sizeof a	取得a所占的存储空间数	a--	先存取a值，再将a值减1
++a	先将a值加1，再存取a值		

(6)条件运算符

条件运算符的形式为：

E1 ? E2 : E3

它的运算结果为：

若E1为真，结果为E2之值；

若E1为假，结果为E3之值。

(7)赋值运算符

赋值运算符有很多种，除了“=”外，一些运算符都对应有赋值运算符，如“-”对应的“-=”，“≪”对应的“≪=”。

赋值运算的左操作数是一个能保存值的表达式，赋值运算将右操作数的值存入左操作数中。

(8)运算符的优先级

C语言的运算符优先级别如下所示，图中同一行内的运算符优先级相同，不同行的运算符，上面的优先级高。

运算符	结合性
() [] ->	从左至右
! ~ ++ -- -(负) *(一元) &(一元) sizeof	从右至左
* / %	从左至右
+ -	从左至右
<< >>	从左至右
< <= > >=	从左至右
== !=	从左至右
&	从左至右
^	从左至右
|	从左至右
&&	从左至右
||	从左至右
?:	从右至左
= += -= *= /= %= &= |= ^=	从右至左
,	从左至右

1.2 基本语句

1.2.1 条件语句

条件语句的一般形式为：

```
if(表达式)
    语句 1
else
    语句 2
```

若表达式的值非 0，执行语句 1，否则执行语句 2。

1.2.2 循环语句

(1) while

while 语句的一般形式为：

```
while(表达式)
语句
```

若表达式的值非 0，循环执行语句，一直到表达式的值为 0。

(2) for

for 语句的一般形式为：

```
for( 表达式 1;表达式 2;表达式 3)
语句
```

for 语句等价于下面的 while 语句：

```
表达式 1
while(表达式 2)
{
    语句
    表达式 3
}
```

(3) do - while 语句

do - while 语句的一般形式为：

```
do
    语句
while(表达式)
```

do - while 语句的含义为，先执行语句，然后计算表达式的值，若值非 0，循环执行语句。

1.2.3 switch 语句

switch 语句为一个多路选择语句，它的一般形式为：

```
switch(表达式)
{
  case CH1: 语句 1
```

```
    case CH2: 语句 2
      ⋮
    case CHn: 语句 n
    default: 语句 n+1
  }
```

若表达式的值为 CH1,执行语句 1,若表达式的值为 2 执行语句 2,若表达式的值不在 CH1 - CHn 之间,执行语句 n+1。使用 switch 时,case 后面的语句中,应该有一个 break 语句,当执行到 break 语句时,跳出 switch 语句,否则将进入下一个 case 语句。

1.2.4 break 和 continue 语句

(1) break 语句

break 语句用于跳出 switch 和循环语句。

(2) continue 语句

continue 语句用于跳过循环的后续部分,开始一个新的循环。

下面的程序 1-2 演示了 switch,break 和 continue 语句的用法:

```
/*======================================*\
    程序 1-2
\*======================================*/
#include <stdio.h>
#define ESC 27
#define ENTER 13
main()
{
    int ch;
    while(1)
    {
        ch = getch();
        if(ch == ENTER )
        {
            printf("\n");
            continue;
        }
        if(ch == ESC) break;
        switch(ch)
        {
        case '\t':
            printf("You pressed Tab \n");
            break;
        case ' ':
            printf("You Pressed SPACEBAR \n");
            break;
        case '0':
```

```
            case '1':
            case '2':
            case '3':
            case '4':
            case '5':
            case '6':
            case '7':
            case '8':
            case '9':
                printf("You Pressed a digit key \n");
                break;
            default:
                printf("Not Tab,SPACE,Digit key \n");
            }
        }
    }
```

1.3 预处理

在C语言的开头常常使用预定义常量、宏定义及包含文件。下面讲解它们的用法。

1.3.1 预定义常量

为了使程序便于修改和具有较好的可读性,C程序中可以将一些常量用一个符号来表示,这样一来,看到这个符号就知道它代表的含义,在需要常量时,也只需要将预定义的值修改即可,不必对程序中出现它的地方都作修改。

定义预定义常量的方法是:

define 标识符 常量值

如:

define PI 3.141 593

以后就可以在使用3.141 593的地方用PI替代。

1.3.2 宏定义

宏定义可以代替一个表达式,并且可以带上参数。如:

define MAX(a,b) ((a) > (b) ? (a) : (b))

在这以后,若使用MAX,编译预处理程序会把它展开,如使用MAX(10,20),将会被展开为:

((10) > (20) ? (10) : (20))

结果为20。

1.3.3 包含文件

C语言中,使用# include可以将另外一个程序包含进来。被包含的文件通常都是一些头文件,其中定义一些宏、常量以及函数说明等,但是,只要您愿意,您可以将整段的C程序作为包含文件包含进来,编译器在处理时,它将被包含文件内容插入到程序中。

包含文件用尖括号或双引号括起来,尖括号表示系统头文件,在指定的目录中查找;双引号表示用户文件,在当前目录查找。使用包含文件的方法为:

include <文件名>

1.3.4 条件编译

在编写 C 程序时,常出现这种情况:某些代码需要在特定的情况下编译,在另外一些情况不需要编译。如为了调试程序而写的输出语句,在程序调好后就不需要被编译了。这时可以使用条件编译。

条件编译的常用语句为:

```
# ifdef  标识符 1
  语句 1
# else
  语句 2
# endif
```

它的含义为,若已经定义了标识符 1,则编译语句 1,否则编译语句 2。

1.4 函数

1.4.1 函数的一般形式

函数的一般形式如下:

```
返回值类型  函数名 ( 参数表 )
参数说明
{
    函数体
}
```

说明的一般形式中参数说明也可以放在参数表中,下面给出一个简单的示例:

```
/*============================*\
  程序 1-3
\*============================*/
# include <stdio.h>
# define PI 3.141593
float area(float r)
{
    return(PI * r * r);
}
main()
{
    float r;
    printf("Please input R of circle: ");
    scanf("%f",&r);
    printf("Area of the circle is: %f \n",area(r));
```

```
}
```

1.4.2 函数参数

在C函数中,参数和函数中的普通变量一样使用,这些参数是将调用函数的实参复制了一份给被调用函数,因此在被调用函数中改变参数的值,不会改变调用函数中的内容。若要被调用程序更改调用程序中的变量值,可以将此变量的地址作为参数传给被调用函数。如下面的程序所示:

```
main()
{
    int a,b,c;
    add(a,b,&c);
}
add(int a,int b,int *c)
{
    *c = a+b;
}
```

1.4.3 函数的返回值

一个函数通常都返回值,返回的值使用 return 语句传递给调用函数,形式为:

```
return(表达式);
```

若一个函数 不返回值,可以将它说明为 void 型,如:

```
void add(int a,int b,int *c)
```

C语言中,若不说明函数的返回值类型,缺省为整型,如说明的 add 函数。

1.5 结构与指针

1.5.1 结构

C语言中可以将一组相关的数据描述为结构,例如可以将姓名、地址、电话、邮编等数据声明为一个结构:

```
struct callingcard
{
    int no;
    char name[20];
    char address[80];
    char tel[10];
    char pocode[6];
};
```

声明了结构后,就可以定义结构变量了:

```
struct callingcard card;
```

定义了一个结构后,要访问结构元素时,在结构名后加上".",如:

```
card.no = 1;
name = "KonHou Lee";
```

```
a = card.no + 5;
```

1.5.2 指针

指针是一个变量,它的内容是另外一个变量的地址,若指针 p 的内容为变量 a 的地址,则称 p 指向 a。指针是有类型的,指向整数的指针称为整型指针,指向字符的指针称为字符指针,另外,C 语言还有一种无类型(void)指针,它只是简单地保存一个地址,这个地址中的内容类型可以在使用它时用类型转换的方法指定。声明指针的方法如下:

```
int *a;
char *b;
float *c;
void *d;
struct callingcard *e;
```

上面声明了几个指针变量,a 是指向整数的指针,b 是指向字符的指针,c 是指向浮点数的指针,d 是一个无类型的指针,e 是一个指向 callingcard 结构的指针。

给指针赋值时,使用一元运算符 &,当然指针向指针赋值和通常的方法一样:

```
int a = 3;
int *b;
int *c;
b = &a;
c = b;
```

上面的两个赋值语句的含义是,首先将 a 的地址赋予 b,再将 b 的值赋予 c,这样,b 和 c 都指向 a,都可以改变 a 中的内容:

```
*b = 4;        /* a中的内容变为4 */
*c += 2;       /* a中的内容变为6 */
```

第 2 章　系统调用

要使用 C 语言编写一些实用的程序，通常都需要进行一些系统功能的调用，来实现特定的功能，因而在应用篇开始之前，首先介绍在 C 语言中系统调用的方法。

2.1　BIOS 调用

BIOS 提供了丰富的键盘，显示，打印及磁盘等调用，C 语言中使用这些调用的方法较多，常用的函数有 int86，int86x，intr 等，另外还有专用的函数 bioskey，biosprint，biostime，biosquip 等。int86，int86x，intr 等函数要使用寄存器结构，它们定义在 < dos.h > 中，可以分不同的情况选择使用它们。下面是 int86，int86x，intr 的使用方法：

(1)int86

```
#include <dos.h>
int86(int intno,union REGS *in_regs,union REGS *out_regs);
```

int86 执行中断号为 intno 的中断，执行前将 inregs 中的内容拷贝到各个寄存器中，中断返回后，再将各个寄存器的内容拷贝到 outregs 中，允许 inregs 和 outregs 使用同一变量。

REGS 的具体内容为：

```
union REGS {
  struct WORDREGS x;
  struct BYTEREGS h;
};
struct BYTEREGS {
  unsigned char al, ah, bl, bh;
  unsigned char cl, ch, dl, dh;
};
struct WORDREGS {
  unsigned int ax, bx, cx, dx;
  unsigned int si, di, cflag, flags;
};
```

(2)int86x

```
#include <dos.h>
int int86x(int intno, union REGS *inregs,
                      union REGS *outregs,
```

```
                    struct SREGS * segregs);
```

int86x 和 int86 实现相同的功能，只是 int86x 可以使用段寄存器，在调用中断前，int86x 会将 segregs 拷贝到段寄存器，中断返回后，又将段寄存器的内容拷贝到 segregs。int86x 中的 inregs，outregs 和 int86 中的相同。

SREGS 的具体内容为：

```
struct SREGS {
  unsigned int es;
  unsigned int cs;
  unsigned int ss;
  unsigned int ds;
};
```

(3)intr

```
#include <dos.h>
void intr(int intno, struct REGPACK * preg);
```

intr 执行中断号为 intno 的中断，执行前将 preg 中的内容拷贝到寄存器，中断返回后，再将寄存器的内容拷贝到 preg 中。

REGPACK 的具体内容为：

```
struct REGPACK {
  unsigned r_ax, r_bx, r_cx, r_dx;
  unsigned r_bp, r_si, r_di;
  unsigned r_ds, r_es, r_flags;
};
```

2.2 DOS 调用

可以直接使用 int86 等开始来实现 DOS 调用，C 语言提供了几个更特别的 DOS 功能调用函数，如 intdos，intdosx，bdos 及 bdosptr 等。下面介绍它们的用法：

(1)intdos

```
#include <dos.h>
int intdos(union REGS * inregs,
           union REGS * outregs);
```

intdos 和 int86 相似，只不过 intdos 省略了中断号 intno，因为 DOS 调用的中断号为 0x21。intdos 中的 REGS 和 int86 中完全相同。

(2)intdosx

```
#include <dos.h>
int intdosx(union REGS * inregs,
            union REGS * outregs,
            struct SREGS * segregs);
```

与 intdos 一样，intdosx 和 int86x 相似，其中的 REGS，SREGS 和 int86x 的完全相同。

(3)bdos

```
#include <dos.h>
int bdos(int dosfun, unsigned dosdx,
```