计算机语言技术系列丛书(二)

实用TurboC程序设计与范例

胡书仿
涂　疑　编著

计算机语言技术系列丛书（二）

实用 Turbo C 程序设计与范例

胡书仿
涂　疑　编　著
胡　芳　审　校

学苑出版社
1994

(京)新登字 151 号

内 容 简 介

本书比较详细地介绍了几种实用程序(例如中文菜单、西文菜单、文件列表选择、造字等)的设计思想和方法。第一章,概括性介绍了 C 语言程序设计的基础知识。第二章,介绍了视频显示的基本原理。第三章,介绍如何识别键盘扫描码。前三章为后面的程序设计打下基础。第四、五章,介绍下拉式汉字菜单、下拉式西文菜单的设计。第六章,介绍文件列表选择程序设计。第七章,介绍造字原理和方法。第八章,介绍图形方式下产生闪烁光标及进行文本输入。第九章,介绍如何进行屏幕打印。

本书内容详实,附有大量的完整的调试通过了的程序。

本书的所有程序都是在 TURBO C 2.0 环境下调试通过的。对从事 C 语言开发与教学的广大科技人员,本书提供有力的帮助。

欲购本书的朋友可直接与北京 8721 信箱联系,邮政编码 100080,电话2562329。

计算机语言技术系列丛书(二)

实用 Turbo C 程序设计与范例

编　　著:胡书仿　涂疑
审　　校:胡　芳
责任编辑:甄国宪
出版发行:学苑出版社　　**邮政编码**:100036
社　　址:北京市海淀区万寿路西街 11 号
印　　刷:北京市朝阳区小红门印刷厂印刷
开　　本:787×1092　1/16
印　　张:13.25　　**字　数**:302 千字
印　　数:1～5000 册
版　　次:1994 年 7 月北京第 1 版第 1 次
ISBN7－5077－0905－1/TP・29
本册定价:19.00 元

前 言

随着计算机应用的深入和普及，C 语言在程序设计中的地位和作用越来越重要，引起越来越多的人的喜爱和重视。这一点勿容置疑。

如果你是一个 C 语言初学者，随着学习的深入，你一定会为 C 语言的魅力和强大的功能而兴奋不已，你可能跃跃欲试地想设计一些有实用价值和有实际意义的程序，如菜单设计，造字等，而一旦你着手这些工作时，又觉得头绪太多而无从下手。编写本书的目的之一就是为已有 C 语言基础而缺乏有 C 程序设计经验或经历的 C 爱好者提供参考。一个 C 语言学习者，很少不为 C 语言的复杂性、灵活性而苦恼，特别是 C 语言中的指针、结构、函数更是让人头脑发涨，这时他需要更多的实例来加深他对 C 的理解和应用，本书的实例将对希望提高、巩固对 C 基础知识的认识和理解的 C 学习者也是一个很好的帮助，这是编写本书的另一个目的。如果这两个目的能得到体现，哪怕是一点点，作者也感到非常欣慰。

本书中所介绍的一些程序设计方法，不一定的是最好的或最简单的，本书只力求本书中的方法能尽快地、自然地被读者所接受和理解，特别是对于那些 C 语言初学者或只有较少 C 程序设计经验的读者。本书丝毫不追求高深的编程技巧，哪怕是这种技巧能使程序短小，代码优化。本书在程序设计上，力求通俗易懂、可靠、易扩展。

限于作者水平有限，经验不足，本书中难免存在错误和不足之处，希望广大读者和同行批评指正。

作者　1994.5 月于中船总 710 所

目 录

第一章　C语言程序设计基础

第一节　常　量

计算机的基本功能是进行数据处理，在C语言中数据处理的基本对象是常量和变量。本节和下一节主要讨论这两种数据类型。

常量是程序中数值不发生变化的量。C语言中的常数有三类：数，字符和字符串。

1.1.1　数

C语言中使用整数和实数两种数。

整数可以使用十进制，八进制和十六进制。八进制整数第一位为0，十六进制前两位为0x，其它为十进制。例如，数值20可以用下面三种不同的形式表示：

十进制　　20
八进制　　024
十六进制　0x14

C语言中，整数的取值范围随着不同CPU的机器和不同的编译系统而不同，一般是由CPU所处理的机器字的位数决定的，如IBM－PC，PDP－11等16位(bit)机器中整数的取值范围是：

－32768～＋32767

超过这个范围的整数，可以使用长整数，长整数的取值范围一般是整数的两倍长，如上述16位机器中，长整数的取值范围是：

－2147483648～＋2147483647

长型整数的表示方法是在整数后接一个l或L，如：

211L，0x48bL，0364l

都是长型整数。

实数又称浮点数。C语言中，实数只使用十进制，实数分为单精度实数和双精度实数，它们的表示方法相同。实数有一般形式和指数形式两种，如：1.57，10.8，－110.01都是一般形式，特别大或特别小的数，要用指数形式表示，如：2.9e10，－7.2－e8都是指数形式的实数。

实数取值的绝对值范围在16bit机器中一般为10^{-38}到10^{38}，实数的精度在使用单精度实数时具有7位有效数字，而使用双精度实数时，具有16位有效数字。

1.1.2　字符常量

字符常量都是一个单一的字符，其表示形式是由两个单引号包围的一个字符，如：

'C'，'8'，'!'，'f'

都是字符常量，其中引号作为定界符使用，并不表示字符常量本身。在两个单引号中包围的字符不能是单引号'和反斜线\，即'''和'\'是错误的表现形式。

在C语言中，字符常量具有数值。字符常量的值就是该字符的ASCII码值，因此，可以说字符常量实际上是一字节的正整数，如字符常量'A'和'?'的数值分别是65和63。

在C语言中，当把字符常量赋予某一变量时，实际上就是把该字符常量的代码值赋予该变量，而且字符常量可以象数一样，在程序中参与各种运算。如：

```
x='D';
y='a'+5;
z='!'+'G';
```

它们的运算相当于

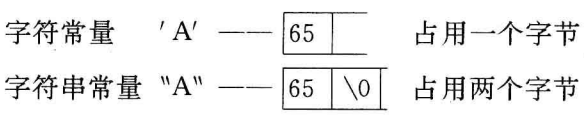
```
x=68;
y=97+5;
z=33+71
```

1.1.3 字符串常量

字符串常量用双引号包围的一字符串表示，这串字符不能包括双引号" 和反斜线\，如"This is a string","String","A"都是字符串常量，其中双引号仅作为定界符使用，不是字符串中的字符。

C语言的字符串常量有与其它语言不同的独特性质。字符串常量在内存中存贮时，自动在其尾部加上一个NULL字符，它也是一个字节(8bit)代码，在ASCII码中，其代码值为0。NULL字符常用\0表示。因此，长度为n个字符的字符串常量，在内存中占用n+1个字节的空间。

字符常量与字符串常量在表现形式和存贮性质上是不同的，如'A'和"A"是两个不同的常量。



1.1.4 换码序列

换码序列是C语言中使用字符的一种特殊形式，换码序列常用于表示ASCII字符集中的控制代码和某些用于功能定义的字符，如象单引号'，双引号"和反斜线\等。换码序列用反斜线\后面跟一个字符或一个数字表示。

控制代码一般是计算机发向外部设备的命令码，它们仅仅控制设备实现某些特定动作，并不是供给用户的输出信息。在ASCII集中，代码值为00－0x1F的代码就是控制代码，在C程序中，可以在字符常量或字符串中包含有控制代码，控制代码的换码序列如表1.1所示，它们的表现形式是在反斜线后面跟有一个小写英文字母。

表 1.1　换码序列

\\	表示反斜线(\)
\'	表示撇号
\"	表示引号
\n	表示回车加换行
\t	表示水平制表符
\b	表示退格
\r	表示回车符
\f	表示清屏符
\xxx	表示 ASCII 码为 xxx 的字符,其中 xxx 是 1 到 3 位八进制数(0－7);
\xHHH	表示 ASCII 码为 HHH 的字符,其中 HHH 是 1 到 3 位 十六进制数字(0－9,A－F)

如 C 语言中表示字符串 I SAY "GOODBY!"和\C PROGRAM\的表示方法分别为"I SAY :\"GOODBY! \""和"\\C PROGRAM\\"。

用换码序列形式还可以表示任一字节代码,其表示形式是在反斜线后面跟有代码值,其代码值必须用三位八进制或三位十六进制表示,其中十六进代码必须以 x 字符打头。

利用换码序列,字符 A 可以表示为\101 或\x041,控制字符\b 表示为\ 010 或\x008,反斜线\可以表示为\113 或\x05c。

1.1.5　符号常量

C 语言中,常量可以用符号代替,代替常量的符号称为符号常量,为了便于与一般变量区分,符号常量一般使用大写英文字母,符号常量在使用之前必须预先定义,其定义的一般格式如下:

```
#define 符号常量 常量
```

例如:

```
#define ENTER 13
#define NULL 0
#define EOF －1
```

每个符号常量定义式只能定义一个符号常量,并且占据一个书写行,该行必须以#号打头,而且后面不能加分号。实际上,它们是 C 语言程序语句,而且发布给编译系统的预处理命令。

符号常量一经定义,就可以在程序中代替常量使用。

符号常量在程序中代替具有一定实际意义的常量,如上面定义的 ENTER,表示 ASCII 码值为 13 的回车键,NULL 表示 ASCII 值为 0 的空字符,EOF 表示文件结尾。使用符号常量增强了程序的可读性,容易理解。此外,程序中某些常量,在调试,扩充和移植时要求修改其值,这种常量通常也定义为符号常量。当它们需要修改时,只需修改其定义即可。在一个

符号常量大量使用的大型程序中，充分体现了这种优越性，同时也避免了一个常量在多次修改中，因失误造成的不一致性而导致的错误。

第二节 变 量

1.2.1 变量的定义

C 语言中的变量定义就是声明某一字符序列(变量标志符或变量名)将被 当作某一事先定义的数据 类型，标志符必须满足以下三条规则的任意一条：

(1) 所有标志符均以字母或下划线开头；

(2) 标志符的其余部分由字母，下划线和数字组成；

(3) 标志符不能与 C 语言的关键字相同。

注意 C 语言区分大小写字母，因此变量 Averag，AVERAG 和 AVeRaG 互不相同。

变量在程序中使用时，必须说明它们的存贮类型和数据类型，变量说明的一般形式是：

```
存贮类型 数据类型 变量名
```

例如，下面给出对变量 data 和 c 的说明

```
static int data;
       char c;
```

具有相同存贮类型和数据类型的变量可以一起说明，它们之间用逗号隔开，例如：

```
int number,i,j,k;
```

变量说明在程序的数据说明部分或函数的参数说明部分说明。编译系统在处理变量说明语句时，根据其存贮类型和数据类型，在特定的存贮区域为该变量分配一定的存贮空间。

1.2.2 变量的数据类型及类型转换

C 语言中的变量，根据其数值的性质，分为以下几种不同的数据类型。

(1) 按照数据的长度，有 8 位，16 位，32 位和 64 位四种。

(2) 按照数据是否带有符号，分为带符号型和无符号型。

(3) 按照数据的数系性质，分为整数型和浮点小数型(又称实数型)。

各种数据类型和它们的表现形式，如表 1.2 所示。其中 char 型又称字符型，这是因为计算机中所使用的字符代码都是 8 位整数。

除了表 1.2 中列出的数据类型外，还有一种更经常使用的数据类型，即 int 型。它是整数型，并且可以分为带符号的 int 型和不带符号的 unsigned int 型。它的数据长度，随着计算机的 CPU 的种类不同而不同，即它与 CPU 所能够处理的数据长度有关。如表 1.3 所示。

表 1.2　变量的数据类型

数　系	符　号	数据长度	表现形式	数值范围
整数型	带符号	8	char	−128～+127
		16	char	−32768～+32767
		32	long	−2147483648～+2147483647
	无符号	8	unsigned char	0～255
		16	unsigned short	0～65535
		32	unsigned long	0～4294967295
浮点小数型	带符号	32	float	约 10^{-35}～10^{38}(七位精度)
		64	double	约 10^{-306}～10^{306}(十六位精度)

表 1.3　int 型变量的数据长度

CPU 种类	int 型数据长度
Z80、8080 等 8bit 系列 8086、80286 等 16bit 系列	16bit(同 short 型)
68000 等 16bit 系列 各种 32bit 系列	32bit(同 long 型)

在 C 程序的语句或表达式中通常应使用同一类型的变量和常量。但是，如果混合使用各种类型，C 语言也不会象 Pascal 那样停止运行，或象 FORTRAN 那样产生奇怪的结果。C 可以根据规则自动进行类型转换，类型转换的规则是：

(1) 如果一个操作中包含两种类型，则低级别的类型转换为级别高的类型，这一过程称为提升。各种类型级别从高到低的顺序是：double，float，long，int，char ，每种类型的符号型高于该类型。

(2) 在赋值语句中，右边表达式的最后结果转换为左边变量类型，这可能导致升级，也可能导致降级。

除了上述的由 C 编译系统自动地实现数据类型转换之外，还可以在程序中强制地进行这种转换。其表现形式是：

(数据类型) 变量名或运算式

它将把后面指出的变量或运算结果强制地转换成指定的数据类型。

例如，sqrt()函数调用时，要求参数是 double 型数据。若变量 n 是 int 型，且作为该函数的参数使用时，则必须按下列方式强制进行类型转换。

```
sqrt((double)n)
```

又如，设 i 为 int 型，则语句 i=10 * (1.55+1.67)将使 i 等于 32(由 32.2 截尾得到)；而语句 i=10 * ((int)1.55+1.67)将使 i 等 于 26(由 26.7 截尾得到，因为(int)1.55 的结果是 1)。

需要指出的是，无论是自动地还是强制地实现数据类型的转换，仅仅是为了本次运算或赋值的需要而对变量的数据长度进行一时性的转换，并不能改变数据说明时对该变量规定的数据类型。

1.2.3 变量的存贮类型和使用范围

变量的存贮类型规定了该变量数据的存贮区域。而变量的存贮区域和变量在程序中的说明的位置决定了它的使用范围。

C 语言中的变量有四种存贮类型，它们是：

auto——堆栈型或称自动型

register——寄存器型

static——静态型

extern——外部参照型

auto 型变量存贮在内存的堆栈区，它们在堆栈区域中属于一时性存贮，并不长期占用内存。其存贮空间可以被若干变量多次覆盖使用。因此，C 语言程序中大量使用的变量为auto型。其目的之一就是为了节省内存空间。

register 变量存贮在 CPU 中的通用寄存器中。通常，使用频率较高的变量设定为register型，目的是提高运算速度。

使用 register 变量是 C 语言所具有的汇编语言特征之一，因而与计算机硬件关系较密切。不同的 CPU 具有个数不等的通用寄存贮，其中供 C 编译系统使用的个数也不同。所以在 C 程序中设定 register 变量的个数不是任意的，通常以两个左右为宜。C 编译系统对于超过 CPU 中可供使用的寄存器个数的 register 变量作为 auto 变量处理。此外，数据类型为 long，double 和 float 的变量不能设定为 register 型，因为它们的数据长度超过了通用寄存器本身的位长。

static 变量存贮在一般的内存区域中。这类变量在数据说明时被分配了一定的内存空间，并且该空间在整个程序运行中，自始至终都归该变量使用。

extern 型变量一般用于在程序的多个编译单位之间传送数据。在这种情况下，指定为 extern 型的变量是在其它编译单位的源文件中定义的，它的存贮空间需参照本身的编译单位外部而决定，所以称为外部参考型。

C 语言的变量，由于其存贮类型不同而有不同的存在寿命，即其使用范围不同。有些变量的寿命限于某个程序范围之内，脱出这个范围后，它们就不再存在。这样的变量具有局部寿命，称为局部变量，与此相对应，有些变量存在于整个程序运行期间，它们具有全局寿命，称为全局变量。

此外，变量说明可以出现在程序不同的位置上。在函数内部，更广义地讲，在某个大括号对包围的程序范围内部被说明的变量，称为内部变量，与此相对应，在所有函数外部被说明的变量称为外部变量。

内部变量可以是 auto 型，register 型和 static 型。其中类型名 auto 可以缺省。即内部变量没有指明存贮类型时，意味着它是 auto 型。

C 语言规定，auto 型和 register 型变量只能是内部变量。它们都是局部变量，其存在寿命和使用范围仅限于包围着该变量说明的大括号对{ }之内。当程序执行完该范围的所有语

句，程序控制脱出{ }以后，这些变量占用的存贮空间被释放，它们就不再存在。由此可知，在不同大括号对包围的程序范围内，例如不同的函数内出现的同名 auto 型和 register 型变量是毫无关系，相互独立的变量，它们不会发生冲突。

static 型变量可以是内部变量，也可以是外部变量。static 型内部变量不同于 auto 型和 register 型内部变量。当程序控制退出这个程序之后，它并不释放占用的内存空间。该空间在整个程序运行期间都由该变量占用。所以，即使退出了它的大括号之后，该变量的数值仍然保留在其内存空间中。因此，static 型内部变量是全局变量。但是这类变量的使用范围仅限于包围其说明的大括号对之中。这称做在该程序范围内它们是可见的。脱出该程序范围后，虽然其值仍然保留着。但它不能被使用，也就是说，它们在包围其说明的大括号外部是不可见的。当程序控制再次进入该大括号包围的程序范围之后，它们又成为可见的。这时它们在内存空间保留的值可以再度使用。所以我们说，static 型内部变量具有全局寿命和局部可见性。由此可知，不同大括号对内部说明的同名 static 型变量，占用各自不同的内存空间，有其各自的使用范围，所以它们不会冲突。

1.2.4 变量的初始化

变量在被说明的同时可以赋予初值，这称为变量的初始化。例如：

```
char a=3;
static int b=5;
```

不同的存贮类型，其初始化的意义不同。auto 和 register 变量若被初始化，则每当程序进入该程序块后，都执行该变量的初始化赋予初值。这两种类型的变量在数据说明中不进行初始化的话，则在程序中必须给它们赋予初值后才能使用。这就是说，在没有进行初始化的 auto 型和 register 型变量，其初值不定，它们不能直接在程序中使用（参与运算或将其值赋予其它变量等）。

static 变量和外部变量在数据说明中不进行初始化的话，自动赋予零值 。与 auto 和 register 不同，static 内部变量的初始化仅执行一次。即第一次进入该程序块时，该变量被赋予初值，而在此后再次进入该程序块时，则不能执行赋初值的功能。

指定为 extern 型的变量不能进行初始化。

static 变量和外部变量初始化时必须使用常量给其赋初值，而 auto 和 register 变量则可以用常量或变量进行初始化。

第三节 操作符

在定义变量的大小和类型之后，需要利用变量进行某种运算，这是通过操作 符完成的，C 语言比大多数高级语言的操作符要多，除了通常的赋值操作和算术操作外，C 还有位操作和一个完整的逻辑集。

1.3.1 赋值操作

最基本的操作符是赋值操作，在 C 中用符号（=）表示，等号右边的值，被赋给左边的变量，多个赋值语句可以同时进行，如 a=b=1 表示将 1 赋给 b，再将 b 赋给 a，其结果相当于 a

=1,b=1，在C语言中,a=ave(x)是一个表达式,而a=ave(x);是一个语句,编译器根据最后分号进行区分,表达式可以用在其它表达式中,因此,a=b+(c=ave(x))；是一个合法语句,该语句执行的结果是,将ave(x)的值赋给c,再将b+c的值赋给a。C还允许一个语句中含有多个经逗号隔开的表达式,其中每个表达式都从左到右求值，整个表达式(由多个表达式组成)的值就等于最后一个表达式的值,例如,a=(olda=a,ave(x))；将a的当前值赋给olda,调用函数ave(x)并将返回的结果赋给a。

1.3.2 算术操作与位操作

下面是几个常用的二元算术操作符

*	乘法
/	除法
+	加法
-	减法
%	取余

前四个操作符对于所有类型的变量(char，float,double)均有定义,取余操作符仅适用于整数。

C语言中还有三个一元算术操作符,其中负号操作符(如-i,i为int型)改变操作数的符号。另外两个一元算术操作符是增量符和减量符,表示为++和--,它们分别表示整数变量或指针加1或减1,操作数经常在表达式的中间,增量或减量操作既可以在变量使用前进行,也可以在变量使用后进行。

二元位操作通过以下符号作用于整数：

&	位与(AND)
|	位或(OR)
^	位异或(XOR)
<<	算术左移(所移位数为操作数)
>>	算术右移(所移位数为操作数)

一元位操作NOT将操作数的所有位求反,其符号为~。例如，设i声明为unsigned int，则i=-0将i设置为最大的无符号整数。

1.3.3 组合操作

C允许操作符与赋值号(=)相结合,以使任一形如

<variable>=<variable><operator><expression>;

的语句可以用

<variable><operator>=<expression>;

代替,其中<variable>代表同一个变量,例如,下述包含x和y的表达式对功能相同：

x=x+y	x+=y;
x=x-y	x-=y;
x=x*y	x*=y;
x=x/y	x/=y;

```
x=x%y        x%=y;
x=x&y        x&=y;
x=x|y        x|=y;
x=x<<y       x<<=y;
x=x>>y       x>>=y;
```

在多数情况下，用左边一列语句写出的程序更容易理解。

1.3.4 逻辑操作

与其它表达式一样，包含逻辑操作符的表达式也有值，逻辑操作的结果为 true 或 false，这可以是两个值间的比较，或一系列"与"及"或"运算的结果，如果逻辑运算的结果为 true，则其值为非零整数；如果结果为 false，则其值为 0。循环和条件语句通过检查逻辑运算的结果来控制程序流程。C 语言中的 9 个逻辑操作符为：

<	小于
<=	小于或等于
==	等于
>=	大于或等于
>	大于
!=	不等于
&&	逻辑与
||	逻辑或
!	逻辑非(一元操作符)

注意==容易与赋值操作符(=)混淆，混淆后仍得到一合法表达式，因为表达式的结果也可以解释为 true 或 false。另外，&& 与 || 不应与对应的位操作符(& 和 |)混淆，因为这可能导致难以发现的逻辑错误。

1.3.5 操作符优先权

与其它高级语言一样，C 的操作符优先权定义了一个表达式中哪些操作先完成，操作的顺序可通过括号改变，因此括号中的部分最先求值，优先权相同的项从左到右计算。表 1.4 概括了 C 语言的全部运算符和结合性规则。运算符的优先级自上而下递减(假定最高级为 15，最低级为 1)，而同一表栏中的运算符优先级相同，如果同时出现在表达式中，则按结合性自左向右或自右向左计算。

例如

a+b-c 等价于 (a+b)-c