TurboC
高级程序员编程指南

中国科学院希望高级电脑技术公司

HOPE

TurboC 高级程序员编程指南 陈昇东 编译
中国科学院希望高级电脑技术公司

Turbo C 高级程序员编程指南

陈捍东　编译

（V2.0 版）

北京希望电脑公司
一九九二年二月

前 言

Turbo C 的到来，总的来说给程序员尤其为 C 程序员提供了一个以其环境和编译器速度而引人注目的令人激动的实现．Turbo C 获得了迅速的成功和老资格及初学者程序员的接受．本书是以所有级别的 C 程序员为目的的．本书论讨了与经常遇到的编程的各个方面相关的话题，如控制台 I／O，鼠标器管理，弹出窗口，串，动态变量，通用编程，图形，文件 I／O，和调试．

前两章提供一个对 C 和 Turbo C 的简短的介绍（或复习，如果你愿意的话）．第一章讨论 C 程序的基本构成，而第二章由对函数的讨论开始．

第三章讨论键盘、鼠标器和屏幕 I／O 的实用基础．该章讨论过的技术和程序在其它章中将用做基础．第四章讨论弹出窗口管理和错误报告窗口．窗口的数据结构随堆栈、隐藏、显示窗口和执行窗口 I／O 的技术给出．

第五章给出一个简明而透彻的对 Turbo C 支持的文件 I／O 系统的说明．这些话题所覆盖的内容是处理文本和二进制文件 I／O；使用文件指针与文件句柄、DOS 文件信息、标准 I／O、及文件缓冲控制．

第六章给出了两个提供到达 Turbo C 的串库例行程序的另一途径的基本串库．这些例行程序使用索引来支持各种各样的串函数．第七章讨论操作指针和处理内程分配的高级技术．具体地说．该章讨论动态串结构的实现．

第八章处理通用排序和查寻话题．首先检查 Turbo C 的通用排序和查寻例行程序．随后有一个关于如何建立你自己的通用例行程序的讨论．例子显示了通用外壳和插入排序函数；合并排好序的数组，及对排好序的数组；二进制查寻，双向线性查寻，及索引表查寻．

第九章包括了执行 DOS 目录操作的 C 函数．这包含有这样一些函数，它们能执行带有多个文件指定的扩展 DIR，带有多个文件指定的文件拷贝实用程序，多文件列表，和一个灵巧的目录跳转．

第十章引人一个变长记录 (ULR) 包．随着显示许多实用的文件 I／O 函数，弹出窗口和动态串软件包也进人应用．该章以一个非常简单的"幻灯片"程序结束，其中图形物体可以用变长记录有存储和恢复．

第十一章指出在使用图形例行程序中的技巧和陷井．该章包括有使用鼠标器及随后显示的样例图形弹出窗口软件包．克服 Turbo C 图形的某些限制的方法也在此给出．

第十二章是使用前些章开发的各种技术和工具的多维文本系统的应用章节．它使用了弹出窗口，鼠标器和键盘 I／O，变长动态串，及用变长记录的文件 I／O．

第十三章讨论 Turbo C 2．O 的调试器和精选的各种流行的错误．

目　录

第一章　C 程序组成

本章浏览 C 程序单元的基本构成并对如下话题作一简短回顾:

- 数据类型
- 变量和常量说明
  编译器指令
- 指针
- 操作符
- 表达式
- 决策结构
- 循环结构

1.1　预定义数据类型

Turbo C 支持一大组预定义的简单数据类型。把基本类型标识符和类型修正标识符组合起来即可使之成为可能。数据类型标识符是:

Data Type Identifier	Byte Size	Class
char	1	character
int	2	integer
float	4	floating
double	8	floating
void	0	typeless

类型修正符是:

Type Modifier Identifier	Effect
short	Reduces valid range of values.
long	Extends valid range of values.
signed	High bit is used as a sign bit.
unsigned	High bit is not used as a sign bit.

表 1.1 显示了能由组合基本类型和类型修正符来使用的简单数据类型组合.

typedef 使你能够用单个字标识符代替多一字数据类型标识符。这在下例中说明。

```
typedef unsigned int word;  /* define word     */
typedef unsigned char byte; /* define byte     */
typedef double extended;    /* define extended */
```

Table 1.1　Predefined Data Types in Turbo C

Simple Data Type	Byte Size	Value Range	Sample Constant(s)
char	1	-128 to 127	-5, 'a'

signed char	1	-128 to 127	5, 'b'
unsigned char	1	0 to 255	5, 'x'
int	2	-32768 to 32767	-234
signed int	2	-32768 to 32767	-344
unsigned int	2	0 to 65535	65000
short int	2	-32768 to 32767	1230
signed short int	2	-32768 to 32767	345
unsigned short int	2	0 to 65535	40000
long int	4	-2147483648 to 2147483647	1000000
signed long int	4	-2147483648 to 2147483647	-2000000
unsigned long int	4	0 to 4294967295	300000000
float	4	3.4E-38 to 3.4E+38 and -3.4E-38 to -3.4E+38	-1.23e-02
long float	8	1.7E-308 to 1.7E+308 and -1.7E-308 to -1.7E+308	2.3e+100
double	8	1.7E-308 to 1.7E+308 and -1.7E-308 to -1.7E+308	-4.32e-100
long double	8	1.7E-308 to 1.7E+308 and -1.7E-308 to -1.7E+308	12.34e+100

1.2 用户定义类型

Turbo C 支持三类用户定义的类型：枚举、结构，和联合。

1、枚举类型使你能够定义一串具有绝对的特殊意义值的标识符。一个枚举表中的元素不能出现在任何别的枚举表中。说明一个枚举类型的总的语法是：

```
enum <enumerated type name> {<list of members delimited by commas>};
```

说明枚举类型的例子有：

```
enum booleans { FALSE, TRUE };
enum colors { red, blue, green, yellow, white, cyan };
enum error { no_file, no_mem, no_disk, no_printer };
enum days { Sun, Mon, Tue, Wed, Thu, Fri, Sat };
```

枚举表中的第一个元素由系统设置（对所有给出的例子都是这样）被赋为 0，第二个被赋为 1，等等。你可以用赋值给全部或部分枚举表元素的办法显式改变数字的上升序列。考虑枚举类型天数情况。你可以把 1 赋给元素 Sun（Sunday 的简记）而非 0，因为它是一个星期的第 1 天而非"第 0 天"！上面的枚举类型说明可重写如下：

```
enum days { Sun = 1, Mon, Tue, Wed, Thu, Fri, Sat };
```

这也把 2 赋给 Mon，3 赋给 Tue，等等。在这个例子中，只有一个枚举元素被赋值。一个极端的情况涉及到给枚举表中每个元素赋一具体值，如下例所示：

```
enum error { no_file = 5,
             no_mem  = 7,
             no_disk = 11,
             no_printer = 21};
```

2、结构使你的应用能定义含有逻辑相关的域的数据类型。这些域可以有预定义或用户定义的类型。这样你就可包括进数组、枚举类型，其它结构，和联合。在C中用下面的总的语法说明结构:

```
struct <structure name> {
                          <type 1> <field 1>;
                          <type 2> <field 2>;
                          <type 3> <field 3>;
                          ..................
                          <type n> <field n>;
                        };
```

说明结构的例子是:

```
struct complex_math {
                      double real;
                      double imag;
                     };
struct pixel_point {
                      int x_coord;
                      int y_coord;
                      enum colors color;
                    };
struct mail_rec {
                  char name[31];
                  struct address_rec address;
                  double loan_amount;
                };
```

Turbo C 等支持位域 (bitfields)，它们是允许你存取具体位的特殊结构。其说明的总的语法是:

```
struct <bitfield name> {
                          <type 1> <field 1> : <bits 1>;
                          <type 2> <field 2> : <bits 2>;
                          <type 3> <field 3> : <bits 3>;
                          ..................
                          <type n> <field n> : <bits n>;
                        };
```

位域从最不重要的位到最重要的位映像到内存位置。位域的例子是:

```
struct two_chars {
                   unsigned int char1 : 8;
                   unsigned int char2 : 8;
                  };
struct keyboard_status {
                    unsigned int capslock  : 1;
                    unsigned int scrollock : 1;
                    unsigned int numlock   : 1;
                   };
```

```
struct two_chars myword;
struct keyboard_status kbd_st;
myword.char1 = 'a';
myword.char2 = 64; /* ASCII code of character 'A' */
if myword.char1 == myword.char2
    kbd_st.capslock = 0;
else
    kbd_st.capslock = 1;
kbd_st.capslock = 1 - kbd_st.capslock; /* toggle key status */
```

3、联合是其域在内存中被覆盖的结构。每个域提供一个不同的存取数据的格式并且不补充别的域。说明联合的总的语法是:

```
union <union name> {
                    <type 1> <field 1>;
                    <type 2> <field 2>;
                    <type 3> <field 3>;
                    ..................
                    <type n> <field n>;
                   };
```

在下例中有一联合被说明为每个域占 8 个字节:

```
union eight_bytes {
                        char   c[8];
                        int    i[4];
                        float  x[2];
                        double y;
                      };
```

1.3 变量和常量说明

在 C 中用下面的总的语法来说明变量:

```
<data type>  <list of variable names>;
```

表中的变量由逗号分开并且可被初始化。说明使用预定义的或用户定义的类型的数量变量的例子是:

```
int i, j, k = 12;
char ch1 = 'a', ch2 = 65 /* ASCII code of 65 */;
double pi = 31.4;
struct complex a, b, c = { 2.34, 74.5 };
union eight_byte c;
```

C 中用指定各维大小的方法说明数组。每个数组维的下限固定为 0。每个数组维由单独一套括号表示。说明并同时初始化数组的例子是:

```
int numbers[10] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 0 };
char digits[10] = { '1', '2', '3', '4', '5',
                    '6', '7', '8', '9', '0' };
```

```
struct complex x[2] = {  { 1.0, 11.0 },   /* value for x[0] */
                         { 2.3. 9.43 } }; /* value for x[1] */
double matrix[2][2] = { { 1.0, 2.0} ,  /* values for x[0][0..1] */
                        { 3.0, 4.0} }; /* values for x[1][0..1]  */
```

这些例子也反映出多维数组存储最右索引（或下标）比其左边的那个变化得快的这样的值。如果在初始化中数据目标的个数也是被寻找的数组的大小，则被初始化的数组可以使其大小除去。这样上面的例子可以重写如下以利用这一特性:

```
int numbers[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 0 };
char digits[] = { '1', '2', '3', '4', '5',
                  '6', '7', '8', '9', '0' };
struct complex x[] = { { 1.0, 11.0 },   /* value for x[0] */
                       { 2.3. 9.43 } }; /* value for x[1] */
double matrix[][] = { { 1.0, 2.0} ,  /* values for x[0][0..1]  */
                      { 3.0, 4.0} }; /* values for x[1][0..1]  */
```

串被 C 当作以空字符做为串定界符的字符数组。当确定一个串的大小时，必须有一个元素用于空字符。Turbo C 提供了一个广泛的串管理函数的库。文件 sting.h 中含有这些串函数的原型。

Turbo C 支持两个存取修正符: const 和 volatile。

常量或用#define 指令（见下节）说明为宏，或用新的 ANSI 常量说明。后一种说明一个标识符有一个固定值并且不可改变。说明一个常量的总的语法是:

```
const <type> <constant identifier> = <value>;
```

如果略去了常量类型，则系统假定是 int 类型。说明常量的例子是:

```
const DAY_IN_WEEK = 7;
const double PI = 3.14;
const char DELIMITER = '|';
```

Turbo C 实现另一种变量访问修正符，名为 volatile 修正符。它告诉编译器一个变量的内容也可由潜在的系统改变。

Turbo C 支持下面 4 种存储类:

1、auto 存储说明符显式地说明局部变量是自动的。函数中的局部变量在函数一旦终止其执行并返回其调用者时就消失。

2、static 存储说明符表明一个局部变量应存储在一个不变的内存位置并在函数调用之间保存其值。

3、extern 说明符解决在连接时间对同一目标的引用。它普遍应用于多文件项目中。另外，extern 说明符用于访问全程、编译器、和意义的变量，也用于写运行时间库函数的原型。

4、register 说明符指示编译器使用硬件 CPU 寄存器来存储关键数据以获得高速运行

1.4 编译器指令

Turbo C 有下列指令

1、#define 指令定义带有可选择参量的宏。总的语法是:

```
#define <macro> <macro text or value>
#define <macro(<list of argument>) <macro expression>
```

使用#define 指令的例子有:

```
/* define data type macros */
#define BOOLEAN char
#define BYTE unsigned char
#define REAL double
#define INTEGER int
/* define macro keywords */
#define PRINT printf
#define WRITE printf
#define INPUT scanf
#define READ scanf
#define ReadKey getch()
#define ReadChar getche()
#define WRITELN printf("\n")
#define WRITELN2 printf("\n\n")
#define WRITELN3 printf("\n\n\n")
/* macros for popular shorthand command sequences */
#define readvar(msg,frmt,var) printf(msg); scanf(frmt, &var)
#define read2var(msg,frmt,x,y) printf(msg); scanf(frmt, &x, &y)
#define readchar(msg,var) printf(msg); var = getche()
/* define screen macros (somewhat similar to those in conio.h).
   Requires that the ANSI.SYS driver be in CONFIG.SYS  */
#define clrscr printf("\x1b[2J")
#define gotoxy(col,row) printf("\x1b[%d;%dH",col,row)
#define clreol printf("\x1b[K")
/* define boolean constants */
#define FALSE 0
#define TRUE 1
/* boolean pseudo-functions */
#define boolean(x) ((x) ? "TRUE" : "FALSE")
#define yesno(x) ((x) ? "Yes" : "No")
/* macros that define pseudo one-line functions */
#define abs(x) (((x) >= 0) ? (x) : -(x))
#define max(x,y) (((x) > (y)) ? (x) : (y))
#define min(x,y) (((x) > (y)) ? (y) : (x))
#define sqr(x) ((x) * (x))
#define cube(x) ((x) * (x) * (x))
#define reciprocal(x) (1 / (x))
/* macros used for character testing */
#define islower(c) ((c >= 'a') && (c <= 'z'))
#define isupper(c) ((c >= 'A') && (c <= 'Z'))
#define isdigit(c) ((c >= '0') && (c <= '9'))
#define isletter(c) ((c >= 'A') && (c <= 'z'))
/* macros used in character case conversions */
#define tolowercase(c) (c - 'A' + 'a')
#define touppercase(c) (c - 'a' + 'A')
```

下面是一个带有参数的宏怎样被预处理器扩展的简单例子:

```
int a = 4, b = 7;
printf("Is %d greater than %d : %s",
       a, b, boolean(a > b))
```

这被预处理器转化为下面的行:

```
int a = 4, b = 7;
printf("Is %d greater than %d : %s",
       a, b, ((a > b) ? "TRUE" : "FALSE"))
```

1. 在说明一个带有参数的宏时，要把宏表达式中每个参数都括在括号中，并把整个宏表达式也括在括号中。这可确保基于宏的伪函数对表达式类型的参量及在别的表达式中都可正常工作。

2、用#define 创建的宏可用#undef 指令撤销定义。总的语法是:

```
#undef <macro name>
```

3、#include 指令用于从别的文件中包含进源代码。

```
#include <filename>
```

或

```
#include "filename"
```

这两种形式的差别在于被包含文件的寻找方式。如果使用小括号，则只有被包含文件的特殊目录被查寻。当前目录不被检查。

4、#error 错误信息指令在遇到错误时停止程序编译并显示一个伴随的错误信息。总的语法是:

```
#error <error message text>
```

5、#if 、#else，#elif 和#endif 等条件编译指令以与 if 语句类似的方式起作用。唯一区别是结果涉及是否编译某部分代码。下面是使用这套指令的一个例子。

```
#include <stdio.h>
#define FORMATTED_INPUT 1
main()
{
   char string[81];
   printf("Enter string : ");
   #if FORMATTED_INPUT == 1
      scanf("%s", string);
   #else
      gets(string);
   #endif
   printf("\n\nYou entered ");
   puts(string);
}
```

6、#line 指令允许你给预定义的宏__LILE__赋一个值和给宏__FILE__赋一个文件

名。使用这一指令的总的语法是:

```
#line <line number> ["filename"]
```

7、#pragma 指令不严格地由 ANSI 标准定义为一组通用指令，用下面的形式:

```
#pragma <directive name>
```

#pragma 可随支持它们的 C 实现而变化。所用的规则是如果指令不可识别，则连同#pragma 指令一起乎略。有两个 Turbo C 支持的#pragma 指令:

7.1 #pragma inline 指令告诉编译器你的源程序中含有插入在行中的汇编语言代码。

7.2 #pragma warn 使 Turbo C 越过警告信息。你可包含打开或关闭设置，或恢复文件编译开始时的警告值。这些设置显示如下:

Directive	Meaning
#pragma warn +<wrn1>	Warning <wrn1> is turned on.
#pragma warn -<wrn2>	Warning <wrn2> is turned off.
#pragma warn .<wrn3>	Warning <wrn3> is restored.

更完整的警告清单见 Turbo C 参考手册中的附录 C (Turbo C Reference Manual)。

1.5 基本控制台 I / O

控制台 I / O 可以分为格式化的和非格式化的。

1.5.1 格式化的 I / O

这类 I / O 使你能处理一个函数调用中的多数据项 I / O 而同时控制数据的格式。该类中的冠军是能通过控制台分别输出和输入的 printf 和 scanf 函数。为使用这些 I / O 函数，C 程序必须包含头文件 stdio.h

printf 函数带有不同数目的参数，其中第一个必须总是一个串常数或一个变量。printf 可以含有一个单个的串类型的参数，这种情况下，它被用来仅仅输出该串本身的内容。然而，printf 函数是主要设计成将其第一参数用作输出格式控制的。表 1.2 表明了能由串使用的某些 C 转换序列，而表 1.3 提供 printf 的不同的选择。printf 函数对单个数据项起作用也对串起作用 (即，以空字符适当终止的字符数组)。

Table 1.2 The C Escape Sequences

Sequence	As Hex Value	Decimal Value	Task of Sequence
\a	0x07	7	Bell
\b	0x08	8	Backspace
\f	0x0C	12	Formfeed
\n	0x0A	10	New line
\r	0x0D	13	Carriage return
\t	0x09	9	Horizontal tab
\v	0x0B	11	Vertical tab
\\	0x5C	92	Backslash
\'	0x2C	44	Single quote
\"	0x22	34	Double quote
\?	0x3F	63	Question mark
\OOO			1 to 3 digits for an octal value
\XHHH and \xHHH			1 to 3 digits for a hexadecimal value

Table 1.3 Formatted I/O String Control

% [flags] [width] [.precision] [F|N|h|l] <type character>

Flag Character	Effect
-	Justify to the left within the designated field. The right side is padded with blanks.
+	Display the plus or minus sign of value.
blank	Display a leading blank if the value is positive. If the output is negative, a minus sign is used.
#	Display a leading 0 for octals. Display a leading 0X or 0x for hexadecimals. Display the decimal point for reals. No effect on integers.

Category	Type Character	Output Format
character	c	single character
integer	d	signed decimal int
	i	signed decimal int
	o	unsigned octal int
	u	unsigned decimal int
	x	unsigned hexadecimal int. The numeric character set used is [01234567890abcdef].
	X	unsigned hexadecimal int. The numeric character set used is [01234567890ABCDEF].
pointer	p	prints only offset of near pointers as AAAA and far pointers as SSSS:OOOO
pointer to int	n	stores a count of the characters printed so far in the specified pointer location
real	f	signed value in the form [-]dddd.dddd
	e	signed scientific format using [-]d.dddd e[+|-]ddd
	E	signed scientific format using [-]d.dddd E[+|-]ddd
	g	signed value using either 'e' or 'f' formats, depending on value and specified precision
	G	signed value using either 'E' or 'f' formats, depending on value and specified precision
string pointer	s	emits characters until a null-terminator or precision is attained

scanf 函数与 printf 相对。其第一个参量是串格式控制。别的参量必须是接收信息的数据目标的地址。对数量变量，&操作符用于提供数据目标的地址。下面的例子表明怎样使用 printf 和 scanf 函数:

```
#include <stdio.h>
#include "math.h"
main()
{
   int i, j;
   double x, y;
   char c, d;
   char name[81];
   struct complex {
       double real, imag;
       } a;
   printf("Enter you name : ");
   scanf("%s", name);
   printf("\nHello %s, how are you?\n\n", name);
   printf("Enter an integer : ");
   scanf("%d", &i);
   j = i + 1;
   printf("\n%d + 1 = %d\n\n", i, j);
   printf("Enter a real number : ");
   scanf("%lf", &x);
   y = x * x ;
   printf("\n%lf^2 = %lf\n\n", x, y);
   printf("Enter a character : ");
   scanf("%s", name); /* input to a string */
   c = name[0]; /* pick the first character of the string */
      d = c + 1;
      printf("\n%c follows %c\n\n", d, c);
      printf("Enter a complex number (2 reals delimited by space) : ");
      scanf("%lf %lf", &a.real, &a.imag);
      x = sqrt(a.real * a.real  + a.imag * a.imag);
      printf("\nabsolute value of complex number = %lf\n\n", x);
   }
```

1.5.2 非格式化的 I / O

非格式化 I／O 涉及串和字符. 串 I／O 由分别读和写串到标准输出的 gets 和 puts 执行.

```
   #include <stdio.h>
   main()
   {
      char name[81];
      int i;
      printf("Enter a string : ");
      gets(name); /* no & operator is needed */

      for (i = 0; name[i] != '\0'; i++)
         /* if lowercase convert to uppercase */
         if (name[i] >= 'a' && name[i] <= 'z')
            name[i] += 'A' - 'a';
      puts(name); /* display uppercase version */
   }
```