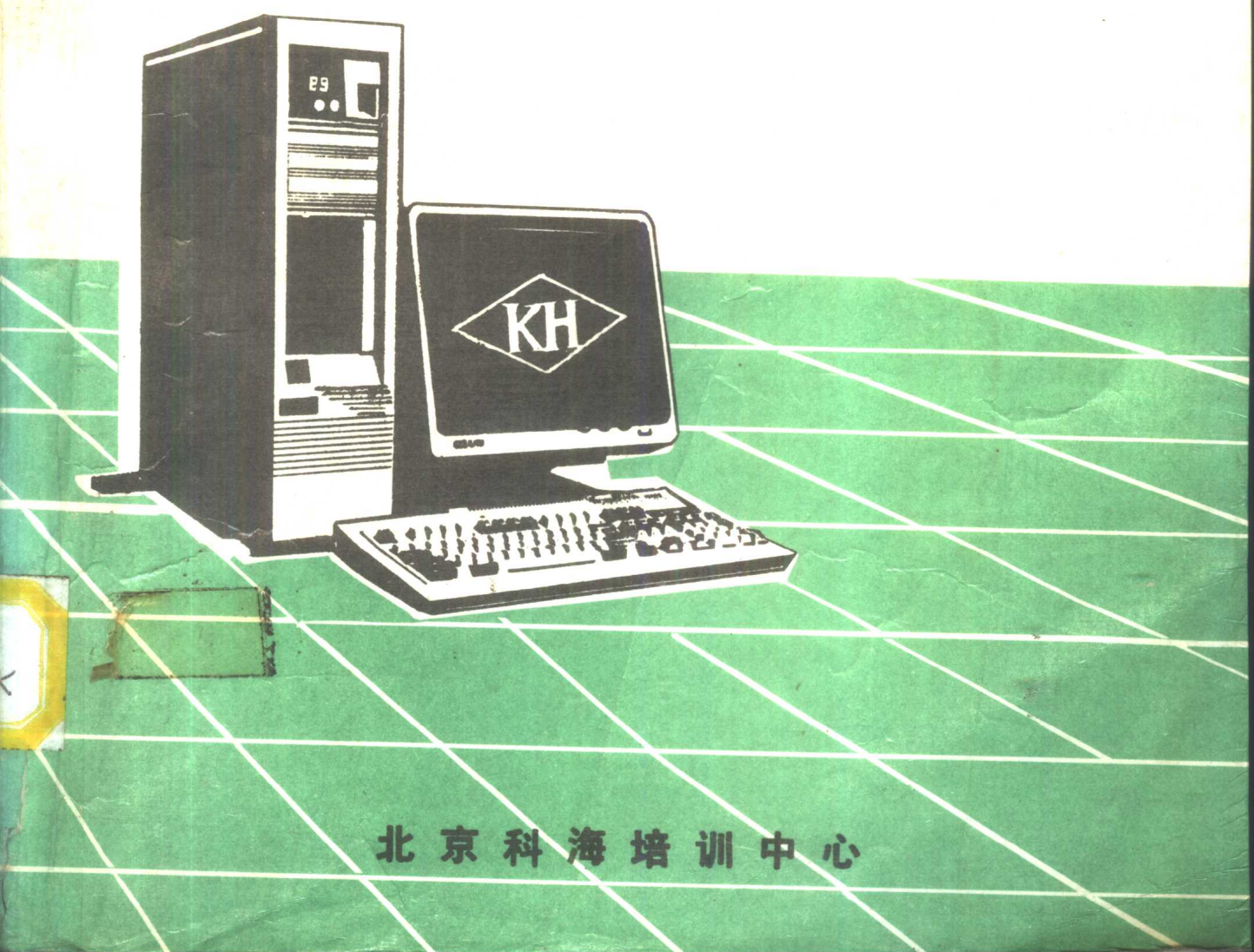
科海培训中心
系列教材

TURBO C 2.0
高级程序设计技术

作者 叶 新



北京科海培训中心

Turbo C 2.0 高级程序设计技术

林雪柏　唐克　蓝兰　编译

北京科海培训中心

一九九零年十一月

编译者序

Turbo C 的出现为一般程序员，特别是为 C 程序员提供了以其环境与编译速度而著称的激动人心的实现。Turbo C 一举成功并为专家以及新手所普遍接受。

随着 Turbo C 2.0 软件包在我国的不断应用和普及，广大计算机用户越来越感觉到其确实是软件开发必不可少的工具之一，掌握 Turbo C 的技术人员也越来越多。为了使这一软件包发挥更大的作用，我们特编译了本书，目的是使广大用户在原来的基础上提高一步，达到更高的应用境界。

本书主要介绍 Turbo C 2.0 的高级程序设计技术，内容涉及控制台 I/O、鼠标管理、弹出窗口、串、动态变量、图形学、文件 I/O 以及高级调试等等。书中配备了大量的程序实例，供广大读者选用。

书中给出了在 Turbo C 2.0 中应用鼠标的具体方法和实现，介绍了程序调试的高级技术；文件 I/O 的技巧；动态内存管理技术；一个正文编辑系统实例，为目前已有的 Turbo C 书中所不见。以下是本书章节安排：

头两章提供了对 C 和 Turbo C 的简单介绍。第一章讨论一个 C 程序的基本要素，第二章讨论函数。

第三章讨论了键盘、鼠标和屏幕 I/O 的实用基础。该章中所讨论的技术与代码将为其它章广为使用。第四章讨论了弹出窗口管理与错误报告窗口，给出了窗口的数据结构以及隐蔽、显示、栈化窗口和完成窗口 I/O 的技术。

第五章简单但全面地介绍了 Turbo C 支持的文件 I/O 系统，所论及的问题有处理正文文件和二进制文件、使用文件指针与文件句柄、DOS 文件信息、标准 I/O 和文件缓冲区控制等。

第六章给出了两个基本的串库以作为 Turbo C 串库例程的替代。这些例程在各种串函数中利用了索引。第七章讨论了操作指针及处理内存分配的高级技术。该章特别讨论了动态串结构的一种实现。

第八章处理通用排序与查找问题。首先考察了 Turbo C 的通用排序与查找例程，然后讨论了怎样建立自己的通用例程。

第九章讨论进行 DOS 目录操作的 C 函数，包括将 DIR 命令扩充成能处理多个文件说明的函数、文件拷贝的实用工具、多个文件的列举以及聪明的目录跳转。

第十章介绍了一个变长记录(VLR)包。该章说明了许多文件 I/O 函数的实际应用，还使用了弹出窗口包及动态串包，最后以一个非常简单的“幻灯显示”程序而结束。

第十一章给出了使用图形例程的技术与陷井。该章给出了一个典型的图形弹出窗口包，还说明了克服 Turbo C 图形学局限的方法。

第十二章利用前面章中开发的不同工具与技术给出了一个超级正文应用系统，它利用了弹出窗口、鼠标和键盘 I/O、变长动态串以及变长记录的文件 I/O。

第十三章讨论了 Turbo C 调试器及各种典型错误。

附录 A 详细介绍 Turbo C 2.0 的编辑命令。

附录 B 列出并解释了 Turbo C 2.0 的编译错误信息，简要地说明了产生信息的可能原因。

附录 C 列出了与用户有关的 TCC（命令行编译器）任选项入口。

附录 D 介绍了 Turbo C 的实用工具，讨论了包含在 Turbo C 软件包里的实用程序：预处理程序 TCC、MAKE 管理程序、Turbo 连接程序 TLINK、Turbo 库管理程序 TLIB、文件搜索实用程序 GREP、图形驱动程序和字体转换实用程序 BGIOBJ 和目标模块交叉参考程序 OBJXREF。

附录 E 说明如何使用定做程序 TCINST。通过 TCINST 设置自己的键盘配置、修改缺省值、改变屏幕颜色等。

本书由译者结合国外最新 Turbo C 资料，并在多年编程的基础上，经过认真整理而成，但由于时间仓促，加上我们水平有限，书上错误缺点仍在所难免，欢迎广大读者指正，以备再版时修订。

编 译 者

一九九零年十一月

目　录

第一章　C 程序要素

本章讨论 C 程序的基本组成成分以用作对如下课题的简单回顾:

- 数据类型
- 变量和常量声明
- 编译指令
- 基本控制台 I/O
- 指针
- 操作符
- 表达式
- 判断语句
- 循环语句

§ 1.1　预定义数据类型

Turbo C 支持预定义的简单数据类型的多种组合。这可通过组合基本类型标识符和类型修饰符来完成。数据类型标识符是:

数据类型标识符	字节大小	类别
char	1	字符
int	2	整数
float	4	浮点数
double	8	浮点数
void	0	无类型

类型修饰符有:

类型修饰符	结果
short	减少值的有效范围
long	增大值的有效范围
signed	高位被用作符号位
unsigned	高位不被用作符号位

表 1.1 列出了简单数据类型的组合。

typedef 可用单字长的数据类型标识符来替代多字长的标识符，如下例所示：

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

表 1.1 Turbo C 中的预定义数据类型

简单数据类型	字节大小	值域	典型常量
char	1	-128至127	-5,'a'
signed char	1	-128至127	5,'b'
unsigned char	1	0至255	5,'x'
int	2	-32768至32767	-234
signed int	2	-32768至32767	-344
unsigned int	2	0至65535	65000
short int	2	-32768至32767	1230
signed short int	2	-32768至32767	345
unsigned short int	2	0至65535	40000
long int	4	-2147483648至2147483647	1000000
signed long int	4	-2147483648至2147483647	-2000000
unsigned long int	4	0至4294967295	300000000
float	4	3.4E-38至3.4E+38 和-3.4E-38至-3.4E+38	-1.23e-02
long float	8	1.7E-308至1.7E+308 和-1.7-308至-1.7E+308	2.3e+100
double	8	1.7E-308至1.7E+308 和-1.7-308至-1.7E+308	-4.32e-100
long double	8	1.7E-308至1.7E+308 和-1.7E-308至-1.7E+308	12.34e+100

§1.2 用户定义类型

Turbo C 支持三种用户定义类型：枚举，结构和联合。

1. 枚举类型定义一种标识符——它具有特殊含意的隐含值——的表。一枚举表中的元素不能出现在任何其它的枚举表中。声明一枚举类型的一般语法是：

enum <枚举类型名> {<用逗号隔开的元素表>};

声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_disk, no_printer };
enum days { Sun, Mon, Tue, Wed, Thu, Fri, Sat };
```

在缺省情况(如上面所有给出的例子)下，枚举表中的第 1 个元素被赋为 0，第 2 个元素被赋为 1，等等。可以对所有或部分枚举元素赋值来改变数的递增性。考虑枚举类型 days 的例子，可以将 1 而非 0 赋给元素 Sun(Sunday 的缩写)，因为它是一星期中的第一天而不是“第 0 天”。

这样的枚举类型声明可重写成:

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

这同样将 Mon 置为 2，将 Tue 置为 3，等等。在此例中，只明确地对一个枚举元素赋了值。一种极端情况包括对枚举表中的每个元素都置一特殊值，如下例所示:

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

2. 结构定义包含逻辑上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。所以就可在域中包含数组、枚举型、其它结构和联合。结构是用下面的一般语法来声明的:

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

声明结构的例子有:

```
struct complex_math {
                      double real;
                      double image;
                    };
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还支持位域，它是可存取特殊位的特殊结构，其语法为:

```
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>;
                      };
```

位域将内存地址从最低位映射到最高位。例如:

```
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可被覆盖的结构。每个域提供了存取数据的一种可选形式，并

且不补充说明其它域。声明一个联合的一般语法是:

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

下例中声明了一个联合，其中每个域占据 8 个字节:

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

§ 1.3 声明变量及常量

C 中用下列语法来声明变量:

<数据类型> <变量名表>;

表中的变量由逗号隔开并可被初始化。利用预定义或用户定义的类型来声明标量变量的例子有:

```
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 中是通过说明每个维的大小来声明数组的。每个数组维的下界定为 0，各数组维是用单独的花括号来表示的。声明并同时初始化数组的例子有:

```
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][3] = { {1.0,2.0}, /* values for x[0][0..1] */
```

```
                {3.0,4.0} }; /* values for x[1][0..1] */
            {3.0,4.0}}; /* values for x[1][0..1] */
```

这些例子还说明了多维数组是怎样存储值以使右边的下标比它左边的下标变化快的。初始化数组时，如果数据对象的数目正好是所建立数组的大小，那么就可省略此大小。所以上面的例子可重写成：

```
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] */
```

C 将串看成是用空字符作为串的定界符的字符数组。在给出串的维数大小时，必须用一元素表示空字符。Turbo C 提供了一个含大量串操作函数的库。文件 string.h 包含了这些串函数的原型。

Turbo C 支持两种存取修饰：const 和 volatile。常量可用#define 指令声明成宏(详见下节)，也可用新的 ANSI const 声明成宏。后者将一标识符声明成有一不可改变的固定值。声明常量的一般语法是：

const <类型> <常量标识符> = <值>;

如果省略了常量类型，则缺省假定为 int 类型。声明常量的例子有：

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

Turbo C 有另一种变量存取修饰，即 volatile 修饰。它告诉编译器变量的内容也可为其内部系统所改变。

Turbo C 支持下列四种存储类：

1. auto 存储说明符将局部变量明确地声明成自动的。函数中的局部变量在该函数终止运行并返回到调用它的函数中时立即消失。

2. static 存储说明符说明局部变量，将其存储在内存中一永久的位置中并在函数调用间保留其值。

3. extern 说明符在链接时将指引变成同一对象。它通常用在多文件的工程中。此外，还可用 extern 说明符来存取全局，编译和被定义变量，以及对运行时的函数进行原型说明。

4. register 说明符告诉编译器使用硬件 CPU 寄存器来存储关键数据以提高速度。

§1.4 编译指令

Turbo C有下列指令:

1. #define指令定义有可选变元的宏。其语法为:

#define <宏> <宏正文或值>

#define <宏(<变元表>)> <宏表达式>

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[KJ")

/* 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')
```

下面是一个变元的宏怎样为预处理器所扩展的简单例子:

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

它为预处理器转换成:

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

在声明含变元的一个宏时，将宏中每个变元及整个宏表达式都用括号括起来。这样就能保证基于宏的伪函数既能和表达式类型的变元一起也能在其它表达式中正确地被执行。

2. 由#define 所创建的宏可由#undef 指令来消除。一般语法是:

```
#undef <宏名>
```

3. #include 指令被用来将源代码从另一文件中包括进来。

```
#include <文件名>
```

或

```
#include "文件名"
```

两种形式在寻找所要包括的文件上是不同的。使用尖括号就只搜索特殊目录而不检查当前目录。

4. #error 出错信息指令在遇到错误时终止程序编译并显示相应的出错信息。一般语法是:

#error <出错信息正文>

5. #if, #else, #elif 和#endif 条件编译指令的工作方式类似于 if 语句。唯一的差别是涉及判断是否编译特定代码段。下面是利用这组指令的一个例子:

```
#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("\"\"You entered");
  puts(string);
  }
}
```

6. #line 指令给预定义的宏_LINE_赋一值。并将一文件名赋给宏_FILE_。使用该指令的一般语法是:

#line <行号> ["文件名"]

7. #pragma 指令是作为下列形式的一通用指令:

#pragma<指令名>

#pragma 指令可能在支持它们的不同 C 实现中各不相同。如果没有识别出<指令名>中的指令，就完全忽略掉#pragrma 指令。

Turbo C 支持两种#pragma 指令:

(1) #pragma inline 指令告诉编译程序你的源代码含汇编语言代码。

(2) #pragma warn 指令使 Turbo C 让警告信息有/无效。可用此指令在开始编译文件时打开、关闭或恢复警告值。指令的警告有:

指令	含意
#pragma warn +<wrn1>	打开警告<wrn1>
#pragma warn -<wrn2>	关闭警告<wrn2>
#pragma warn <wrn3>	恢复警告<wrn3>