

中国科学院希望高级电脑技术公司

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

H

HOPE

TP312

C42

354182

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

JS202/15

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。本书讨论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 的串库例程序的另一途径的基本串库。这些例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程序。随后有一个关于如何建立你自己的通用例程序的讨论。例子显示了通用外壳和插入排序函数; 合并排好序的数组, 双向排好序的数组; 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, 它们能执行带有多文件指定的扩展 DIR, 带有多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。

第十章引入一个变长记录 (OLR) 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入应用。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程序中的技巧和陷阱。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 2.0 的调试器和精选的各种流行的错误。

目 录

前言

| | |
|----------------------------------|------|
| 第一章 C 程序组成 | (1) |
| 1.1 预定义数据类型 | (1) |
| 1.2 用户定义类型 | (2) |
| 1.3 变量和常量说明 | (4) |
| 1.4 编译器指令 | (5) |
| 1.5 基本控制台 I/O | (8) |
| 1.5.1 格式化的 I/O | (8) |
| 1.5.2 非格式化的 I/O | (10) |
| 1.6 指针 | (11) |
| 1.7 操作符 | (15) |
| 1.8 表达式 | (18) |
| 1.9 决策结构 | (20) |
| 1.10 循环结构 | (22) |
| 第二章 函数 | (25) |
| 2.1 返回结果的函数 | (25) |
| 2.2 修改参数的函数 | (28) |
| 2.3 面向过程的函数 | (30) |
| 2.4 递归函数 | (31) |
| 2.5 函数指针 | (31) |
| 2.6 访问命令行参数 | (34) |
| 2.7 带有可变数目参数的函数 | (35) |
| 2.8 创建及使用库 | (37) |
| 第三章 基本键盘、鼠标器及屏幕 I/O | (38) |
| 3.1 键盘 | (38) |
| 3.2 基本文本输出 | (41) |
| 3.3 直接视频存取 | (42) |
| 3.4 TurboC 窗口 | (45) |
| 3.5 文本颜色 | (47) |
| 3.6 控制光标大小 | (48) |
| 3.7 使用鼠标器 | (51) |

| | |
|------------------------------|--------------|
| 3.8 基本鼠标器功能 | (51) |
| 3.9 检查鼠标器驱动程序 | (53) |
| 3.1.10 鼠标器工具箱 | (53) |
| 3.1.11 鼠标器光标 | (55) |
| 3.1.12 鼠标器突出显示例子 | (56) |
| 第四章 弹出窗口和错误报告 | (69) |
| 4.1 窗口结构 | (70) |
| 4.2 弹出窗口堆栈 | (71) |
| 4.3 操作窗口堆栈 | (72) |
| 4.4 隐藏和显现窗口 | (73) |
| 4.5 窗口 I/O | (73) |
| 4.6 一个简单菜单程序 | (74) |
| 4.7 移动窗口程序 | (75) |
| 4.8 弹出错误和信息包 | (76) |
| 第五章 文件 I/O | (98) |
| 5.1 文本与二进制文件 | (99) |
| 5.2 文件指针与文件把柄 | (99) |
| 5.3 DOS 文件信息 | (100) |
| 5.4 预定义的流和把柄 | (101) |
| 5.5 标准 I/O | (102) |
| 5.5.1 打开和关闭标准 I/O 文件 | (102) |
| 5.5.2 获取文件状态 | (104) |
| 5.5.3 控制文件缓冲 | (105) |
| 5.6 对文件的随机访问 | (106) |
| 5.6.1 读、写标准 I/O 文件 | (108) |
| 5.6.2 字符级和串级访问 | (109) |
| 5.6.3 记录级访问 | (109) |
| 5.6.4 结构压缩 | (110) |
| 5.7 系统级文件 I/O | (110) |
| 5.7.1 为随机访问打开文件 | (111) |
| 5.7.2 读和写系统级文件 | (112) |
| 5.8 文件 I/O 软件包例子 | (113) |
| 第六章 串函数库 | (123) |
| 6.1 库 strops1.c | (123) |
| 6.2 库 strops2.C | (126) |
| 6.3 一个应用: 基本文本文件翻译器 | (133) |
| 第七章 高级指针和内存分配技术 | (145) |
| 7.1 动态串 | (146) |
| 7.2 通用串 | (147) |

| | | |
|-------------|--------------------------|--------------|
| 7.3 | 指针分配 | (148) |
| 7.4 | VSTR 软件包 | (149) |
| 7.5 | 决定 VSTR 的大小 | (150) |
| 7.6 | 用动态串插入和删除 | (151) |
| 7.7 | 动态串与链接表 | (153) |
| 7.8 | 一个例子: 用动态串表示多边形 | (154) |
| 第八章 | TurboC 通用编程 | (161) |
| 8.1 | 通用例行程序 | (161) |
| 8.2 | 建立通用程序 | (164) |
| 8.3 | 补充的通用排序/查寻库 | (165) |
| 第九章 | 目录实用程序 | (176) |
| 9.1 | 扩展的目录函数和应用 | (176) |
| 9.2 | 扩展的文件拷贝函数和应用 | (178) |
| 9.3 | 多文件列表实用程序 | (178) |
| 9.4 | 目录跳转 | (179) |
| 第十章 | 高级文件 I/O | (193) |
| 10.1 | 变长记录文件 | (193) |
| 10.2 | 在文件中找 VLRS | (193) |
| 10.3 | 插入和删除 VLRS | (193) |
| 10.4 | 记录碎片 | (194) |
| 10.5 | VLR 文件格式 | (195) |
| 10.6 | VLR 文件头 | (195) |
| 10.7 | VLR 记录格式 | (195) |
| 10.8 | VLR 软件包 | (196) |
| 10.9 | 打开及创建 VLR 文件 | (196) |
| 10.10 | 访问头部 | (197) |
| 10.11 | 添加和删除记录 | (198) |
| 10.12 | 确定 VLRS 的类型 | (199) |
| 10.13 | 更新 VLRS | (199) |
| 10.14 | 建立一个内部 VLR 索引 | (201) |
| 10.15 | 例子: 一个简单的幻灯片程序 | (203) |
| 第十一章 | TurboC 图形 | (218) |
| 11.1 | 对 TurboC 图形的快速浏览 | (218) |
| 11.2 | 使用鼠标器 | (220) |
| 11.3 | 通过鼠标器驱动程序改变鼠标器光标 | (221) |
| 11.4 | 建自己的光标 | (224) |
| 11.5 | 一个样例图形弹出窗口软件包 | (228) |
| 11.6 | 窗口状态 | (228) |
| 11.7 | 初始化窗口软件包 | (229) |

| | | |
|-------------|----------------------------|--------------|
| 11.8 | 画窗口 | (230) |
| 11.9 | 交换图形 | (230) |
| 11.10 | 清除窗口 | (230) |
| 11.11 | 改变窗口 | (231) |
| 11.12 | 移动窗口例子 | (231) |
| 11.13 | 图形状态下的文本 | (234) |
| 11.14 | 格式化文本 | (235) |
| 11.15 | 覆盖文本 | (236) |
| 11.16 | 突出显示文本 | (238) |
| 11.17 | EGA 橡皮带式生成线 | (239) |
| 11.18 | 总结 | (242) |
| 第十二章 | 高级计划—多维文本系统 | (249) |
| 12.1 | 多维文本编译程序 | (249) |
| 12.2 | 多维文本浏览程序 | (250) |
| 12.3 | 动态串使用 | (253) |
| 12.4 | 多维文本浏览程序中的函数 | (254) |
| 12.4.1 | mainc) 函数 | (254) |
| 12.4.2 | paint~htxc) 函数 | (254) |
| 12.4.3 | get~next~cardl) 函数 | (254) |
| 12.4.4 | make~index~cardl) 函数 | (254) |
| 12.4.5 | 其它函数 | (255) |
| 12.5 | 多维文本系统的限制 | (255) |
| 第十三章 | 调试 | (267) |
| 13.1 | TurboC 调试器 | (267) |
| 13.2 | 精选的错误 | (268) |

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:

| <u>Data Type Identifier</u> | <u>Byte Size</u> | <u>Class</u> |
|-----------------------------|------------------|--------------|
| char | 1 | character |
| int | 2 | integer |
| float | 4 | floating |
| double | 8 | floating |
| void | 0 | typeless |

类型修正符是:

| <u>Type Modifier Identifier</u> | <u>Effect</u> |
|---------------------------------|-------------------------------------|
| 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

| <u>Simple Data Type</u> | <u>Byte Size</u> | <u>Value Range</u> | <u>Sample Constant(s)</u> |
|-------------------------|------------------|--------------------|---------------------------|
| 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数量的变量的例子是:

```

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 `string.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,fmt,var) printf(msg); scanf(fmt, &var)
#define read2var(msg,fmt,x,y) printf(msg); scanf(fmt, &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 __LINE__ 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 指令一起忽略。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. |
| 0 | Display a leading 0 for octals. |
| x | Display a leading 0X or 0x for hexadecimal. |
| . | 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:0000 |
| 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 | signed scientific format using [-]d.dddd |
| | e | 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。下面的例子表明如何使用 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 your name : ");
    scanf("%s", name);
    printf("\nHello %s, how are you?\n\n", name);
    printf("Enter an integer : ");
    scanf("%d", &i);
    j = i + 1;
    printf("\nid + 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("\nc 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 */
}

```

字符 I/O 用 `getch` (无回映地读一字符), `getche` (首屏幕回映地读一字符), `putchar`, 和 `putc` 执行。下面是简单字符 I/O 的一个例子:

```
#include <stdio.h>
#include "conio.h"
main()
{
    char c, d;
    printf("Enter a character : ");
    c = getche(); /* input with echo */
    printf("\nYou typed ");
    putchar(c);
    printf("\nEnter a character : ");
    d = getch(); /* input with no echo */
    printf("\nYou typed ");
    putchar(d);
}
```

1.6 指针

指针是 C 中有力的数据结构。它们含有其它数据项的地址。C 指针也与数据类型相关联。这使得指针能保持它指向的数据项的大小的记录。这一特性给指针以很强的数学运算能力: 例如, 当一个指针改变了 n , 则其相关的地址就改变 n (它指向的目标的大小)。这使指针能通过卷遍每一个数组元素来访问数组。星号字符用在指针说明中。例如:

```
int *ptr, i; /* only ptr is a pointer */
```

一旦被说明, 指针就必须在它们能正常工作之前用有效地址初始化。有两个主要例行和序来初始化一个指针:

1. 给指针赋一已存在的目标的地址。这种情况下, 指针存储另一变量 (数量的, 数组, 结构等等) 的内存地址。操作符 `&` 用于返回一个变量的地址。这类指针初始化的例子包括:

```
char c = 'a', *ptr_char;
int i = 11, *ptr_int = &i; /* ptr_int now points to i */
struct complex a = {12.2, 334.9}, *ptr_cplx;
ptr_char = &c; /* ptr_char now point to c */
ptr_cplx = &a; /* ptr_cplx now points to a */
```

涉及到数组和串时, C 允许裸数组标识符用作指向数组第一元素的指针。这样一来, 例如, 若有数组如下被说明:

```
char myname[31];
```

则标识符 `myname` 等价于指向第一元素的表达式 `&myname[0]`。所以, 当指针与数组一起用时, 指针只被赋给数组名, 如下:

```
char myname[31];
char *ptr = myname;
```

2. 指针涉及到动态分配。运行时间系统分配一块新内存区并把它地址赋给指针。函数 malloc 和 calloc 用于新动态分配。函数 realloc 用于调整旧动态空间的大小。如果(重)分配失败, 则这些函数返回一个 NULL 指针。因为它们返回空一类型的指针, 所以为使这些指针的地址被正确访问必须要求有类型分配。动态分配的内存稍后可用函数 free 释放。为使用这些函数, C 程序必须包含 alloc.h 头文件。使用 malloc 动态分配内存的例子是:

```
struct complex {
    double real, imag;
};

struct complex *ptr;
/* use pointer type casting since malloc returns a void pointer */
ptr = (struct complex *) malloc(sizeof(struct complex));
```

指针被用于以星号和操作符来访问它们指向的数据。

```
#include <stdio.h>
#include "math.h"
#include "alloc.h"
main()
{

    struct complex {
        double real, imag;
    };

    char d, c = 'a', *ptr_char;
    int j, i = 11, *ptr_int = &i; /* ptr_int points to i */
    double x;
    struct complex a = {12.2, 334.9 }, *ptr_cplx;
    struct complex *ptr;
    ptr = (struct complex *) malloc(sizeof(struct complex));
    ptr_char = &c; /* ptr_char now point to c */
    ptr_cplx = &a; /* ptr_cplx now points to a */
    d = *ptr_char + 1; /* ASCII of d = ASCII of c + 1 */
    j = *ptr_int * *ptr_int; /* j = i * i */
    /* copy contents of 'a' into dynamic memory using the
       ptr_cplx and ptr pointers.
    */
    ptr->real = ptr_cplx->real;
    ptr->imag = ptr_cplx->imag;
    x = sqrt(a.real * ptr->real + a.imag * ptr->imag);
}
```

在访问一个数组时，依应用的情况，指针可使用下列方案之一：

1. 当一个数组中的元素按非顺序次序被访问时，可用一个偏移量整数来选择被寻找的元素。如果 $X[i]$ 代表数组的一个元素，它由指针 ptr 用 $*(ptr+i)$ 来访问。类似地，该元素的地址也可用 $\&X[i]$ 或 $(ptr+i)$ 的形式得到。在该访问方案中，指针被赋给数组的基地址并保留该地址。

2. 当数组元素将按完好的顺序被访问时，可用指针算术运算。该方法通常由把数组的基地址赋给该指针开始。为访问下一元素指针加 1。这样指针地址总是在变。按完好的顺序方式访问数组元素避免了使用数组标识符和偏移量索引。下例按降序值初始化一个数组而按升序排序之。两种数组存取方法都用到了。

```
#include <stdio.h>
#define SIZE 100
main()
{
    int numbers[SIZE];
    int i, j, tempo, *ptr;
    /* assign descending values to array members */
    ptr = numbers; /* assign base-address of array to pointer */
    /* scroll through the array */
    for (i = 0; i < SIZE; i++, ptr++)
        *ptr = SIZE - i;
    ptr = numbers; /* reassign base-address of array to pointer */
    /* employ bubble sort to arrange the array in ascending order */
    for (i = 0; i < (SIZE - 1); i++) {
        for (j = i+1; j < SIZE; j++) {
            /* access elements in an imperfect sequential manner */
            if (*(ptr+i) > *(ptr+j)) {
                tempo = *(ptr+i);
                *(ptr+i) = *(ptr+j);
                *(ptr+j) = tempo;
            }
        }
    }
}
```

```

    }
}
/* display array in descending order */
for (i = 0; i < SIZE; i++)
    printf("%d ", *(numbers+i));
printf("\n\n");

```

涉及到用指针访问多维数组时，就用下面的方案：

| <u>Access Using Indices</u> | <u>Access Using Pointers</u> |
|-----------------------------|---|
| <code>x[i][j]</code> | <code>*(*(ptr + i) + j)</code> |
| <code>x[i][j][k]</code> | <code>*(*(*(ptr + i) + j) + k)</code> |

下例说明了用指针访问二维数字数组的用法。程序提示用户输入行数、列数、数据矩阵，然后计算矩阵中每列的平均值（参见下列程序）。

```

/*
C program that demonstrates the use of pointers to access
two-dimension arrays. The average value of each matrix column
is calculated.
*/
#include <stdio.h>
#include "conio.h"
#define MAX_COL 10
#define MAX_ROW 30
main()
{
    double x[MAX_ROW][MAX_COL];
    double sum, sumx, mean;
    int i, j, rows, columns;
    clrscr();
    do {
        printf("Enter number of rows (2 to %d) : ", MAX_ROW);
        scanf("%d", &rows); printf("\n");
    } while (rows < 2 || rows > MAX_ROW);
}

```

```

do {
    printf("enter number of columns [1 to %d] : ", MAX_COL);
    scanf("%d", &columns); printf("\n");
} while (columns < 1 || columns > MAX_COL);
for (i = 0; i < rows; i++) {
    for (j = 0; j < columns; j++) {
        printf("X[%d][%d] : ", i, j);
        scanf("%lf", *(x+i)+j); printf("\n");
    }
    printf("\n");
}
for (j = 0; j < columns; j++) {
    sum = rows;
    sumx = 0.0;
    for (i = 0; i < rows; i++)
        sumx += *(x+i)+j;
    mean = sumx / sum;
    printf("\n\n");
    printf("Mean for column %d = %lf\n\n", j+1, mean);
} /* for j */
printf("\n\n");
}

```

1.7 操作符

C 中的操作符可以划分为下列几类 (见表 1.4)。

1. 算术运算符这包括 4 个算术运算函数和模块运算符。
2. 赋值运算符这允许更新一个变量的赋值写成较短的形式。每个赋值运算符由一个等号和一个其左的算术运算符或位运算符组成。赋值运算符的总的语法是:

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

它等价于长形式:

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

赋值操作符的例子有:

```

sum += x; /* same as sum = sum + x */
power *= term; /* same as power = power * term */
map |= 0x1F; /* same as map = map | 0x1F */

```

3. 加一 (+) 和减一 (-) 运算符 这些运算符使变量值增加或减少 1。它们被紧挨着放在变量名或前或后, 中间无空格。如果这些运算符被放在变量名之前, 则该变量值在它进入任何表达式之前先被改变。如果这些运算符放在变量名之后, 则在该变量被改变之前该变量的值可进入任何表达式。例如:

```

main ()
{
    int i, j, k;
    i = 10;
    j = 34;

```

```

    k = j * i++; /* i contributes a 10 to the expression
                  and then is incremented. k is assigned 340 */
    k = j * -i; /* i is first decremented from 11 to 10.
                  The value of 10 enters in evaluating k */
)

```

4. 关系运算符 这些运算符用于构成一个逻辑表达式。如果它们的值是零则这些表达式为假；否则为真。条件运算符可被看作是语句的袖珍形式。条件运算符的总的语法是：

```

(<expression>) ? <if-true-value> : <if-false-value>;

```

如果表达式为真（也即，非零），则<if-true-value>返回；否则<if-false-value>返回。值得注意的是没有关系运算符 XOR，它可被模拟。使用关系运算符的例子是：

```

#include <stdio.h>
main ()
{
    int i, j, k;
    i = 10;
    j = 34;
    k = (i > 5); /* assign a non-zero value to k */
    k = (! (i > 5)); /* assigns a 0 to k */
    k = (i >= 11) && (j == 30); /* assign 0 to k */
    k = (i != 8) || (j <= 0); /* assigns a 0 to k */
    k = (i > 100) ? 100 : -100; /* assigns -100 to k */
}

```

5. 位运算符 这些运算符改变一个变量的位模式。（包括 AND、OR、XOR 和 NOT 等位操作形式。

使用这些位运算符的例子如下：

```

main :
{
    int i, j, k;
    i = 50;
    j = 124;
    k = i | j; /* OR bits of i and j */
    k = i & j; /* AND bits of i and j */
    k = i >> 2; /* shift right bits of i by 2 places */
    k = i << 3; /* shift left bits of i by 3 places */
}

```

6. 存取运算符 这些运算符包括分别用于非指针和指针来存取结构和联合的域的点操作符，括弧用于存取数组的元素，而圆括号用于存取地址。

7. 大小运算符 这个象函数的运算符返回一个数据类型或一个变量的大小。该运算符的单个参量或是数据类型或是变量名。

8. 逗号运算符 这个运算符主要用于 for 循环中隔离表达式。

表 1.4 也显示了各种各样的 C 运算符的先后顺序。

Table 1.4 Operators in C

| Arithmetic Operators | | Arithmetic Assignment Operators | |
|----------------------|-------------|---------------------------------|----------------------|
| C Operator | Function | Assignment Operator | Equivalent Long Form |
| + | Unary Plus | $x += y$ | $x = x + y$ |
| - | Unary Minus | $x -= y$ | $x = x - y$ |
| + | Add | $x *= y$ | $x = x * y$ |
| - | Subtract | $x /= y$ | $x = x / y$ |
| * | Multiply | $x *= y$ | $x = x * y$ |
| / | Divide | | |
| % | Modulus | | |

Relational Operators

| Operator | Meaning |
|----------|---------|
| && | AND |
| | OR |
| ! | NOT |
| N/A | XOR |
| < | |
| <= | |
| > | |
| >= | |
| = | = |
| != | ≠ |
| ? | : |

Bit-manipulating Operator

| Operator | Meaning |
|----------|---------|
| & | AND |
| | OR |
| ^ | XOR |
| ~ | NOT |
| << | SHL |
| >> | SHR |

Bit-manipulating Assignment Operators

| C Operator | Long Form |
|------------|--------------|
| $x = y$ | $x = x y$ |
| $x ^= y$ | $x = x ^ y$ |
| $x <<= y$ | $x = x << y$ |
| $x >>= y$ | $x = x >> y$ |

Operators in C with their precedence and evaluation direction

| Category | Name | Symbol | Eval. Direction | Precedence |
|-----------|-----------------|--------|-----------------|------------|
| Selection | Parentheses | () | left to right | 1 |
| | Array indexing | [] | left to right | 1 |
| | Field reference | . | left to right | 1 |
| | | -> | left to right | 1 |
| Monadic | Post-increment | ++ | left to right | 2 |
| | Post-decrement | -- | left to right | 2 |
| | Address | & | right to left | 2 |
| | Bitwise NOT | ~ | right to left | 2 |
| | Type cast | (type) | right to left | 2 |
| | Logical NOT | ! | right to left | 2 |
| | Negation | - | right to left | 2 |
| | | | | |

| | Plus sign | + | right to left | 2 |
|----------------|------------------|--------|-----------------|------------|
| | Pre-increment | ++ | right to left | 2 |
| | Pre-decrement | -- | right to left | 2 |
| | Type cast | (type) | right to left | 2 |
| | Size of data | sizeof | right to left | 2 |
| Multiplicative | Modulus | % | left to right | 3 |
| | Multiply | * | left to right | 3 |
| | Divide | / | left to right | 3 |
| Category | Name | Symbol | Eval. Direction | Precedence |
| Additive | Add | + | left to right | 4 |
| | Subtract | - | left to right | 4 |
| Bitwise Shift | Shift left | << | left to right | 5 |
| | Shift right | >> | left to right | 5 |
| Relational | Less than | < | left to right | 6 |
| | Less or equal | <= | left to right | 6 |
| | Greater than | > | left to right | 6 |
| | Greater or equal | >= | left to right | 6 |
| | Equal to | == | left to right | 7 |
| | Not equal to | != | left to right | 7 |
| Bitwise | AND | & | left to right | 8 |
| | XOR | ^ | left to right | 9 |
| | OR | | left to right | 10 |
| Logical | AND | && | left to right | 11 |
| | OR | | left to right | 12 |
| Ternary | Cond. Express. | ? : | right to left | 13 |
| Assignment | Arithmetic | = | right to left | 14 |
| | | += | | |
| | | -= | | |
| | | *= | | |
| | | /= | | |
| | | %= | | |
| | Shift | >>= | right to left | 14 |
| | | <<= | right to left | 14 |
| | Bitwise | &= | right to left | 14 |
| | | = | | |
| | | ^= | | |
| | Comma | | left to right | 15 |

1.8 表达式

C 支持下列表达式类型。

1、使用运算符的表达式 这些是涉及到上一节讨论过的不同运算符的最常见的表达式。

2、数组下标表达式 这些表达式专用于找回具体的数组元素。它们有两种总的形式。

- 使用一个显式的索引: <数组>[<索引>]
- 使用一个指针和偏移量: *(<数组名>+<索引>)

例如:

```
#include <stdio.h>
main()
{
    int numbers[100], i, j, k, *ptr1;
    double table[10][10], x;
    double* *ptr2;
    ptr1 = numbers;
    i = 10;
    k = numbers[i];
    /* or */
    k = *(ptr1 + i);
    ptr2 = table;
    i = 5;
    j = 3;
    x = table[i][j];
    /* or */
    x = *( *(ptr2 + i) + j);
}
```

3、结构/联合元素选择表达式 是结构或联合的变量可用点运算符来访问它们的任一域而指向结构和联合的指针用→运算符代替之。下面是两种存取类型的例子。

```
#include <string.h>
struct pc_info_rec {
    char brand[31];
    unsigned int num_drive;
    unsigned int kram;
    double cost;
};

typedef struct pc_info_rec pc_info;
main()
{
    pc_info my_pc, *ptr = &my_pc;
    /* assign values of first two fields using my_pc structure */
    strcpy("Hindi PC", &my_pc.brand[0]);
    my_pc.num_drive = 3;
    /* assign last two fields using pointer access */
    ptr->kram = 2000;
    ptr->cost = 5000.0;
}
```

4. 类型分配表达式 这种表达式使你能够从一种类型向另一种兼容的类型传输数据, 当向类型赋较大的容量时, 如果可能的话, 表达式将保留其信息和符号。在下例中, 信息由一种类型被传往另一向上兼容的类型。字母 A 的 ASCII 码被存储在整数和实数变量中。

```
main ()
{
    char c = 'A';
    int i;
    long j;
    double x;
    i = (int) c; /* i is assign 65 */
    j = (long) i; /* j is assigned 65, stored as a long integer */
    x = (double) j; /* x is assigned 65. */
}
```

1.9 决策结构

C 支持 if 和 if-else 语句。if 语句总的语法是:

```
if (tested expression is not zero)
    single statement | { sequence of C statements }
```

if-else 语句总的语法是:

```
if (tested expression is not zero)
    single statement | { sequence of C statements }
else
    single statement | { sequence of C statements }
```

C 中允许嵌套的 if-else 并有如下总的语法:

```
if (tested expression #1 is not zero)
    single statement | { sequence of C statements }
else if (tested expression #2 is not zero)
    single statement | { sequence of C statements }
else if (tested expression #3 is not zero)
    single statement | { sequence of C statements }
.....
else if (tested expression #n is not zero)
    single statement | { sequence of C statements }
else
    single statement | { sequence of C statements }
```

嵌套的 if-else 语句的被测试表达式在它们之一返回一个非零值, 或遇到 else 语句时按顺序计算。

使用 if 语句的别的规则是:

1. 被测试的表达式必须括在圆括号内。
2. C 语言无关键字 then。
3. C 中的块语句括在一个开始和一个结束大括号中 每个语句, 包括块中最后一个,

必须以分号结束。

结束大括号后面必须不接分号。下面给出了一个使用 `if` 语句的例子。程序触发一个串中字母的上下档形式并把所有非字母字符转化为点字符。

```
#include <stdio.h>
main()
{
    char string[81];
    char *ptr;
    printf("Enter a string : ");
    gets(string);
    for (i = 0, ptr = string; *ptr != '\0'; ptr++)
        /* if lowercase convert to uppercase */
        if (*ptr >= 'a' && *ptr <= 'z')
            *ptr += 'A' - 'a';
        /* if uppercase convert to lowercase */
        else if (*ptr >= 'A' && *ptr <= 'Z')
            *ptr -= 'A' - 'a';
        else
            *ptr = '.'; /* assign a dot to non-letters */
    puts(name); /* display altered case version */
}
```

C 语言支持 `switch` 语句，它与嵌套的 `if-else` 语句很相象。总的语法是：

```
switch (variable) {
    case constant1:
        [case constant2: ...]
        one or more statement;
        break;
    case constant3:
        [case constant4: ...]
        one or more statement;
        break;
    ....
    default:
        one or more statement;
        break;
}
```

使用 `switch` 语句时必须遵守下列规则：

1. `switch` 值必须是个与整数兼容的值。你可用变量、函数、或表达式提供 `switch` 值。
2. `case` 标号必须只能含有常量（括在可选择括号内）。
3. 一个 `case` 标号由每个常量值允许。C 不支持值范围与单个 `case` 标号相联系。
4. 在 `case` 语句尾部必须用 `break` 语句强制程序流在 `switch` 语句结束后恢复。
5. 在每个 `case` 语句中不必用括号把语句块括起来。

下面的例子要求输入一个无数号和对所输入的值注释。

```

#include <stdio.h>
main()
{
    enum days { Sunday = 1, Monday, Tuesday,
                Wednesday, Thursday, Friday, Saturday };
    enum days week_day;
    int day_num;
    do {
        printf("Enter a week day number (Sunday = 1) : ");
        scanf("%d", &day_num);
    } while (day_num < 1 || day_num > 7);
    week_day = day_num;
    switch (week_day) {
        case Sunday:
        case Saturday:
            printf("Enjoy the weekend!\n");
            break;
        case Monday:
        case Tuesday:
        case Wednesday:
        case Thursday:
            printf("Working 9 to 5:30!\n");
            break;
        case Friday:
            printf("T.G.I.F.!\n");
            break;
    }
}

```

1.10 循环结构

C 支持许多循环结构

1. for 循环有三部分：初始化语句，循环连续测试，和增 1 减 1 语句。这由下面的总语法说明。

```

for(list of value initialization statement;
    loop continuation test;
    list of increment statement)

```

C 语言中的 for 循环以下面两点著名：

a. for 循环的每一部分都是可选择的。删除所有三部分将导致一个永无终止的执行的敞开的 for 循环。

b. 初始化和增 1 / 减 1 部分可以含有多个由逗号分开的语句。这样，多循环控制变量可用于 for 循环。

下例用降序开初始化一个数组而按升序排序之。两种数组存取方法都用到了。

```

#include <stdio.h>
#define SIZE 100

```

```

main()
{
    int numbers[SIZE];
    int i, j, *ptr;
    /* assign descending values to array members */
    /* scroll through the array */
    for (i = 0, ptr = numbers; i < SIZE; i++, ptr++)
        *ptr = SIZE - i;
    ptr = numbers; /* reassign base-address of array to pointer */
    /* employ bubble sort to arrange the array in ascending order */
    for (i = 0; i < (SIZE - 1); i++) {
        for (j = i+1; j < SIZE; j++) {
            /* access elements in an imperfect sequential manner */
            if (*(ptr+i) > *(ptr+j)) {
                tempo = *(ptr+i);
                *(ptr+i) = *(ptr+j);
                *(ptr+j) = tempo;
            }
        }
    }
    /* display array in ascending order */
    for (i = 0; i < SIZE; i++, ptr++)
        printf("%d ", numbers[i]);
    printf("\n\n\n");
}

```

2. 只要被测试的表达式为真（即非零），do-while 就重复循环体。do-while 循环的总的语法是：

```

do {
    sequence of statements
} while (logical expression);

```

下面提示输入 Yes / No 回答的程序段是 do-while 循环的例子：

```

do {
    printf("More calculations? Y/N ");
    answer = getche();
    printf("\n");
} while (answer != 'Y' || answer != 'y' ||
        answer != 'N' || answer != 'n');

```

3. 只要被测试条件是真（即非零），while 循环就执行其语句。while 循环总的语法是：

```

while (logical expression)
    single statement | { sequence of statements }

```

循环用 break 语句退出。程序在 break 放置处的循环尾后继续执行。exit 语句用于退出一个循环并使程序一起停止。

使用 while 循环的例子显示如下。程序触发一个串的字母的上下档形式并把所有非字母字符转化为点字符。while 循环用于检查输入串中的每个字符：

```

#include <stdio.h>
main()
{
    char string[81];
    char *ptr string;
    int i = 0;
    printf("Enter a string : ");
    gets(string);
    while (*ptr != '\0') {
        /* if lowercase convert to uppercase */
        if (*ptr >= 'a' && *ptr <= 'z')
            *ptr += 'A' - 'a';
        /* if uppercase convert to lowercase */
        else if (*ptr >= 'A' && *ptr <= 'Z')
            *ptr -= 'A' - 'a';
        else
            *ptr = '.'; /* assign a dot to non-letters */
        ptr++;
    }
    puts(name); /* display altered case version */
}

```

循环执行流可用 continue 或 exit 语句控制。break 语句用于从一个循环退出并在其结束后马上恢复。continue 语句，正如名字所暗示的，绕过循环的剩余部分并从其开头重新执行。它试图执行下一重复。

第二章 函 数

本章简短地看一下 C 中用户可定义的函数的不同类型。下面是这些函数的清单:

- 1、返回结果的函数
 - a、返回一个预定义的数据类型
 - b、返回一个用户定义的数据类型
- 2、修改其参量的函数 (也许或也许不返回结果)。
- 3、既不返回结果也不修改参量的函数。

说明一个符合 ANSI 标准的函数的总的语法是:

```
[data.type] function name (list of typed arguments | void)
{
    <declarations of data objects>
    <statements>
    return <expression>; /* optional */
}
```

当函数的数据类型被略掉时, 函数被认为是要返回一个 int 类型的结果。有类型说明的参量表含有参数的详细说明; 每个参数前都有其数据类型。如果未用参量表, 则必须把关键字 void 放在函数名之后的括号内。每个函数都可说明它自己的常量、类型、和变量。所有这些数据目标都只是局部于该函数的。返回语句通常用于把结果发回到调用例行程序。

在新的 ANSI 标准下, 所有其数据类型不是 int 的函数必须说明。这叫做定原型, 它使编译器能够判别出一个函数的任何误用。函数定原型通常发生在任何完成一个全程影响的函数之外。

2.1 返回结果的函数

此类函数遵循一个函数的典型特征: 一个按值取参量并返回一个单个结果的例行程序。在此类函数中不执行参量修改, 其中有两类函数:

- 1、返回一个预定义数据类型函数

这类函数代表了 C 中最简单的函数。下面的例子说明了怎样定模型, 使用, 和说明一组函数。这些函数也说明了此类函数中源于不同参数类型的变化:

```
#include <stdio.h>
#include "math.h"
struct point {
    double xcoord;
    double ycoord;
};
#define ARRAY_SIZE 100
```



```

/* declare function prototype */
double get_pi(void);
double dbl_square(double);
long int_power(int, int);
double distance(struct point, struct point);
double mean(double*, int);
main()
{
    double y;
    long base, expon;
    struct point a = { 1.0L, 2.0L };
    struct point b = { 2.0L, 3.0L };
    double x[ARRAY_SIZE];
    int i, ndata = ARRAY_SIZE;
    char string[31] = "Hello World";
    /* test function get_pi() */
    printf("Pi is aprox. = %lf\n", get_pi());
    /* test function dbl_square */
    y = 5.0;
    printf("%lf squared = %lf\n", y, dbl_square(y));
    /* test function int_power */
    base = 2;
    expon = 4;
    printf("%ld^%ld = %ld\n", base, expon, int_power(base, expon));
    /* test function distance */
    printf("Distance between points a and b = %lf\n", distance(a, b));
    /* test function mean */
    for (i = 0; i < ndata; i++)
        x[i] = i * i - 1;
    printf("Mean value of array = %lf\n", mean(x, ndata));
    /* test function strlen */
    printf("There are %d characters in '%s'\n",
        my_strlen(string), string);
}
/* function with no arguments. Works as a pseudo-constant */
double get_pi(void)
{
    return 355.0L / 113.0L;
}
/* function with a single argument */
double dbl_square(double x)
{
    return x * x;
}
/* function with two simple arguments */
long int_power(int base, int exponent)
{
    long power = 1;

```

```

        while (exponent- > 0)
            power *= base;
        return power;
    }
    /* function that takes structured arguments */
    double distance(struct point a, struct point b)
    /* this function needs math.h to use the sqrt function */
    {
        double x, y;
        x = b.xcoord - a.xcoord;
        y = b.ycoord - a.ycoord;
        return sqrt(x * x + y * y);
    }
    /* function that takes an array */
    double mean(double x[], int ndata)
    {
        double sumx, sum = ndata;
        while (ndata- > 0)
            sumx += x[ndata];
        return sumx / sum;
    }
    /* function that takes a string-typed argument */
    int my_strlen(char* string)
    {
        int count = 0;
        char* ptr = string;
        while (*ptr != '\0') {
            ptr++;
            count++;
        }
        return count;
    }
}

```

2. 返回一个用户定义的数据类型的函数

枚举类型、结构、和联合都可用作函数类型。下例说明函数怎样返回结构。

```

/* function that calculates the centroid of a shape */
struct point centroid(struct point vertex[], int num_vertex)
{
    struct point result = { 0.0, 0.0 };
    int i;
    for (i = 0; i < num_vertex; i++) {
        result.xcoord += vertex[i].xcoord;
        result.ycoord += vertex[i].ycoord;
    }
    result.xcoord /= num_vertex;
}

```

```

        result.ycoord /= num_vertex;
        return result; /* return structure result */
    }
    struct string {
        char str[256];
        int strlen;
    };
    /* return the concatenation of two structured strings */
    struct string concat(struct string str1, struct string str2)
    {
        struct string result;
        int i, j;
        /* initialize result */
        result.str[0] = '\0';
        result.strlen = 0;
        if (str1.strlen > 0) {
            /* copy characters of str1 to intermediate result */
            for (i = 0; i < str1.strlen; i++)
                result.str[i] = str1.str[i];
            result.strlen = str1.strlen; /* copy string length field */
        }
        if (str2.strlen > 0) {
            j = str1.strlen;
            /* concat characters of str2 to intermediate result */
            for (i = 0; i < str2.strlen; i++)
                result.str[i+j] = str2.str[i];
            result.strlen += str1.strlen;
        }
        return result;
    }
}

```

返回结构可使 C 函数返回多个逻辑相关的结果。

2.2 修改参数的函数

此类函数修改其参数并且可能或可能不返回结果。通常，这些函数返回结果主要用来反映它们的任务的成功或失败状态。由引用传递的参量或接收数据或被修改。

函数可修改它们的数量类型的参量：它们或是预定义的或是用户定义的类型。函数 swap-int 是其参数让它们的值被修改的函数的例子。它取两个参数并交换其值：

```

/* function to swap two integers */
void swap_int(int *i, int *j)
{
    int tempo;
    tempo = *i;
    *i = *j;
    *j = tempo;
}

```

下面的 power 函数是一个带有一既接收又返回值的参数的函数例子：

```

int power(double base, double exponent, double *result);
{
    if (base == 0.0) {
        *result = 0.0;
        return 0; /* no-error code */
    }
    else if (base > 0.0) {
        /* function exp and log in math.h */
        *result = exp(exponent * log(base));
        return 0; /* no-error code */
    }
    else {
        *result = -1.0; /* dummy result */
        return -1; /* error code */
    }
}

```

如果指数运算未发生任何错误，则函数返回 0；否则返回 -1。

函数可以修改其数组类型的参量。下面的程序创建、排序并显示一个数字数组。void 类型的函数 initializearray 和 shellsort 改变它们的参数值。initializearray 函数只是用新值填充其数组类型的参数，覆盖原来的值（很象杂乱的废品），shellsort 函数把数组元素按升序排列以改变其数组类型的参数值。

```

/*
   Program will test the speed of sorting an integer array.
   It will create a sorted array (in descending order) and
   then sort it in the reverse order.
*/
#include <stdio.h>
#include "conio.h"
enum booleans { TRUE, FALSE };
typedef enum booleans boolean;
typedef unsigned char byte;
typedef unsigned int word;
/* define 'numbers' as an array-type identifier */
typedef word numbers[ARRAY_SIZE];
#define ARRAY_SIZE 1000
/* declare prototype of void functions used */
void initializearray(int*);
void swapthem(int* int*);
void shellsort(int*);
void displayarray(int*);
main()
{
    char ch;
    numbers A;
    initializearray(A);
    printf("Beginning to sort press <cr>");
}

```

```

        ch = getche(); printf("\n");
        shellsort(A);
        puts("Finished sorting!");
        displayarray(A);
    }
void initializearray(int *A)
/* routine to initialize array */
{
    int i;
    puts("Initializing integer array");
    for (i = 0; i < ARRAY_SIZE; i++)
        *(A++) = ARRAY_SIZE - i;
}
void swapthem(int *x, int *y)
/* routine that swaps elements x and y */
{
    int temporary = *x;
    *x = *y;
    *y = temporary;
}
void shellsort(int *A)
/* routine to perform a Shell-Metzner sorting */
{
    int offset, i, j;
    boolean sorted;
    offset = ARRAY_SIZE;
    while (offset > 1) {
        offset /= 2;
        do {
            sorted = TRUE;
            for (j = 0; j < (ARRAY_SIZE - offset); j++) {
                i = j + offset;
                if (*(A+i) < *(A+j)) {
                    swapthem((A+i), (A+j));
                    sorted = FALSE;
                }
            }
        } while (sorted == FALSE);
    }
}
void displayarray(int *A)
/* Display array members */
{
    int i;
    for (i = 0; i < ARRAY_SIZE; i++)
        printf("%4d", *(A+i));
    printf("\n");
}

```

2.3 面向过程的函数)

这些函数是与其它结构语言如 pascal 中的过程相似 void 类型的函数。它们既不返

回结果也不修改参数；相反，它们执行某一特殊任务。下面执行基本屏幕和光标控制的例行程序是此类函数的例子：

```
/* set of void functions (i.e. procedures) to perform screen
   management and waiting for a key to be pressed.
   The screen and cursor routines work with the ANSI.SYS driver.
*/
void clrscr(void)
{
    printf("\x1b[2J");
}
void gotoxy(int col,int row)
{
    printf("\x1b[%d;%dH", row, col);
}
void clreol(void)
{
    printf("\x1b[K");
}
void wait_key(void)
{
    char akey;
    printf("\n\npress any key to continue");
    akey = getch();
}
```

在上面的 void 函数当中，只有 gotoxy 使用了一个参数表。别的例行程序不需要参数。

2.4 递归函数

C 支持递归函数。这使 C 应用能用递归算法处理数据结构，如表和树。下面的递归函数计算阶乘：

```
double factorial(int n)
/* recursive function that calculates factorials */
{
    if (n > 1)
        /* issue a recursive function call */
        return factorial(n-1) * (double) n;
    else
        return 1.0L;
}
```

2.5 函数指针

一个后随参数表的函数名导致对它的计算，但一个裸函数名是一个到该函数的指针。

用函数指针，你能够完成两类基本任务：

1、把函数赋给一个指针并把该指针用做该函数的间接引用。这听起来似乎是产生了更多的工作，但它可用于减少程序长度。考虑这样一种情况：一个程序具有一个类似函数（即它们有相同参数表并返回相同的数据类型）的库，其中每次只需要一个。如果不用函数指针，则每次这些函数之一被激活时就必须用一个 switch 语句。随着调用的次数的增加，程序长度也迅速增加。相反，使用函数指针，程序长度就能更有效地被控制。只使用一次 switch 语句来连接指针到被寻找的函数。任何到库函数的引用都由使用指针来执行。

2、把函数做为参量传给其它函数。这使你能够创建一个高级函数系统，其中函数的一部分是从一个例行程序库中选出的另一函数。

下面是一个使用函数指针激活该函数的例子。它显示 1 到 10 的范围内，由指针存取的函数值。

```
#include <stdio.h>
main()
{
    double x;
    const double delta = 1.0;
    const double first = 0.0;
    const double last = 10.0;
    /* declare function pointer */
    double (*fx)();
    /* declare prototype of sample function */
    double quad_poly(double);
    fx = quad_poly; /* assign function name to pointer */
    x = first;
    while (x <= last) {
        printf("f(%lf) = %lf\n", x, *(fx)(x));
        x += delta;
    }
}

double quad_poly(double x)
{
    double a = 1.0, b = -3.0, c = 5.0;
    return ((x * a) * x + b) * x + c;
}
```

下例说明怎样说明并使用一个函数指针数组。说明了三个函数指针。第一个访问用户定义的函数，另两个指针由文件 math.h 里的函数联结起来。该程序显示在 1 到 10 的范围内由它的指针存取的每个函数的值：

```
#include <stdio.h>
#include <conio.h>
#include <math.h>
#define MAX 3
main()
{
```

```

double x;
const double delta = 1.0;
const double first = 0.0;
const double last = 10.0;
/* declare array of function pointers */
double (*fx[MAX]) ();
int i;
char ch;
/* declare prototype of sample function */
double quad_poly(double);
fx[1] = quad_poly; /* assign function name to pointer */
fx[2] = sqrt;
fx[3] = log;
for (i = 0; i < MAX; i++) {
    x = first;
    while (x <= last) {
        printf("f(%lf) = %lf\n", x, *(fx)(x));
        x += delta;
    }
    printf("press any key to continue ");
    ch = getch();
    clrscr();
}
double quad_poly(double x)
{
    double a = 1.0, b = -3.0, c = 5.0;
    return ((x * a) * x + b) * x + c;
}

```

下例说明函数指针怎样使你能把函数做为参数传给另一函数。它含有一个寻找该函数在给定范围内最大值的 find_largest 函数。被扫描的函数通过使用函数指针 fx 的办法被做为参数传给 find_largest。

```

#include <stdio.h>
main()
{
    double x;
    const double delta = 0.01;
    const double first = 0.0;
    const double last = 10.0;
    /* declare function pointer */
    double (*fx) ();
    /* declare prototype of sample function */
    double quad_poly(double);
    double find_largest(double, double, double, double (*fx)());
    fx = quad_poly; /* assign function name to pointer */
    printf("The largest value in the range %lf -> %lf ", first, last);
    printf("is %lf\n", find_largest(first, last, delta, fx));
}

```



```

double quad_poly(double x)
{
    double a = 1.0, b = -3.0, c = 5.0;
    return ((x * a) * x + b) * x + c;
}

double find_largest(double a, double b, double step, double (*fx)())
/* find the largest function value in the range [a,b], scanning in
   the step size.
*/
{
    double x = a, big = (*fx)(a);
    while (x <= b) {
        if (big < (*fx)(x))
            big = (*fx)(x);
        x += step;
    }
    return big;
}

```

2.6 访问命令行参数

象 C 这样用于开发操作系统的语言应提供一种访问命令行参数的机制。这使你能够调整程序工作的方式。总的语法是：

A> <program name> <argument 1> <argument 2> ... <argument n>

命令行参数的个数依程序所做而不同。

```
main(int argc, char* argv[])
```

在 C 中,你可借助于使用 main 函数的特殊参数表。该参数表是：

第一个参数 argc 返回一个命令行参数的个数的计数。argv 是存取这些参数的字符指针的数组。

下表列出了 argv 的每个元素指向的项的图。

| i | Contents of argv[i] |
|--------|-------------------------------------|
| 0 | program name (in DOS 3.0 and later) |
| 1 | argument #1 |
| 2 | argument #2 |
| ... | ... |
| argc-1 | argument #(argc-1) |

下面的程序显示了对命令行参数的访问及它们的影响程序行为的使用。这个程序与一个从 DOS 命令行取其输入的简单 4 功能计算器相似。它按下列总的语法使用：

```

    CALC <operand 1> <operator> <operand 2>
/* C program that uses command line arguments to
   perform one-line calculations. Only the four basic
   operations are supported.
   For practical use make filename CALC.EXE, so that when
   you invoke it from DOS you type, for example:

```

```

A> CALC 355 / 113
*/
#include <stdio.h>
#include "stdlib.h"
main(int argc, char* argv[])
{
    char opr;
    int error1, error2, error3;
    char string[81];
    double result, first, second;
    if (argc < 4) {
        printf("Proper arguments : <number> <operation> <number>");
        printf("\n\n");
        exit(0);
    }
    /* convert operands to double */
    first = atof(argv[1]);
    second = atof(argv[3]);
    strcpy(string, argv[2]);
    opr = string[0];
    if (opr != '+' && opr != '-' && opr != '*' && opr != '/')
        error2 = 1;
    else
        error2 = 0;
    if (first == 0.0 || error2 == 1 || second == 0.0) {
        printf("bad number(s) or operator\n\n");
        exit(0);
    }
    switch (opr) {
        case '+':
            result = first + second; break;
        case '-':
            result = first - second; break;
        case '*':
            result = first * second; break;
        case '/':
            result = first / second; break;
    }
    printf("result = %lf\n\n", result);
}

```

2.7 带有可变数目参数的函数

C 允许你写一些接收可变数目的参数的函数。你用来写带有可变数目的参数的函数所需的组成成分有数据目标和从库 `stdarg.h` 中引入的例行程序。必须遵循下列总的步骤：

1. 把文件 `stdarg.h` 包含进你的 C 程序。
2. 如果变参函数不是 `int` 类型的，你必须给它定类型，用：

```
<type> <function>(<list for the type of fixed arguments>,...);
```

以上表明该函数中必须至少有一个固定参数。

3、函数被说明:

```
<type> <function>(<list of fixed parameters>,...);
```

4、从 `stdarg.h` 中引入的指针类型的 `va-list` 用于说明一个指向变参表的指针:

```
va_list <list pointer>;
```

5、变参表指针由调用 `va-start` (也由 `stdarg.h` 引入) 初始化, 使用下面的语法:

```
va_start(<list pointer>, <name of last fixed parameter>);
```

这使表指针能指向第一个变参。

6、访问变参由使用取两个参数的 `va-arg` 函数执行。第一个是表指针, 前面已初始化了。第二个是变参数据类型的名子。由 `va-arg` 返回的结果与后面的数据类型参数是相同的:

```
<variable> = va_arg(<list pointer>, <type casted>);
```

为停止读变参表, 可在表尾放一个表尾元素并检查它的值。正如下例所示, `while` 循环很适宜此目的。

下面的 C 程序用一个带有可变数目参数的函数来判别一个数字表中的最大值。该表是可变大小的参数表。程序如下:

```
/* C program that illustrates functions with a variable
   number of arguments
*/
#include <stdio.h>
#include <stdarg.h>
#define EOL -1
main()
{
    int big;
    void vmax(int*, char*, ...);
    vmax(&big, "The largest of 55, 67, 41 and 28 is ",
        55, 67, 41, 28, EOL);
    printf("%d\n", big);
}
void vmax(int* large, char* message, ...)
{
    int num;
    va_list num_ptr;
    va_start(num_ptr, message);
    printf("%s", message);
    *large = -1;
    while ((num = va_arg(num_ptr, int)) != EOL)
        if (num > *(large))
            *(large) = num;
    va_end(num_ptr);
}
```

2.8 创建及使用库

C 是一种广泛依赖于函数库的小内核语言。每个 C 编译器都带有一套已编译的库和用于定类库函数、数据类型、常量、和任何别的说明的相随的头文件（通过带有 `.h` 文件扩展名）。象 `stdio.h`、`stdlib.h`、`string.h`、和 `alloc.h` 这样少数几个头文件都普遍应用于程序中。

涉及到用户自定义库的创建和使用有一些问题。这个过程涉及以下步骤：

- 1、创建库的源代码的主要步骤可以下列任一方式产生：
 - 程序从零开始写出并作为主程序的一部分被测试。
 - 程序从一个旧程序修改或接收而来并被测试。
 - 程序由旧程序拷贝或抽取而来。
- 2、库的程序被收集到一个单独的源文件，称为 `source.c`。这个文件也应含有从别的库（标准的和 / 或用户定义的引入的例程序的头文件）。
- 3、头文件 `source.h` 被创建为含有函数模型、数据类型，和常量。如果头文件中有数据类型和 / 或常量，则 `source.h` 文件也应插入到文件 `source.c` 中。另外，必须从 `source.c` 文件中删除数据类型和常量的说明。
- 4、投影文件是为每个使用用户定义库的应用创建的。投影文件（带有 `.PRT` 扩展名）是一个文本文件，它列出所涉及的源文件：应用文件名和用户定义的库。每个文件名占一行，没有附加的文件名。假如一个取于文件 `calc.c` 中的应用使用库 `source.c`，则投影文件，称之为 `calc.prj` 是这样的：

```
calc
source
```

这个投影文件告诉编译器所涉及的文件不是标准库的一部分。

如果 `source.c` 将使用另一用户自定义库的函数，如 `smart.c` 的，则 `calc.prj` 投影文件将含有如下的行：

```
calc
source
smart
```

Project（投影）选择必须在 Turbo C 环境或命令行形式中被激活，而且投影文件名必须指定。有了这些信息的，Turbo C 编译器就能够使用用户定义的库了。

为了使用下一节里的库，你须创建并使用投影文件。

第三章 基本键盘、鼠标器和屏幕 I/O

在本章, 你将学到键盘、鼠标器和屏幕 I/O 的基本知识。这里开发的技术将在本书后面得到广泛应用, 所以认真研究它们是很重要的。每种 I/O 设备所覆盖的内容并非是透彻的, 而是向你说明开发常用应用所需知道的东西。具体地说, 你将学到怎样开发基于鼠标器和键盘的弹出窗口应用。

3.1 键盘

我们要学的第一件事是如何用偏离标准 C 函数 `getch()` `getche()` 和 `getchar` 的方法处理击键。由这些函数得到的键发自 DOS。在别的地方你可以以为以更直接的方式获取键而击键。这些别的方法通常更方便, 尤其对窗口化的应用。作为开始, 你将学到击键是怎样被转化为你的程序使用的键码的。

基本上有两类处理键的键盘中断: 一类是中断 0×09 , 另一类是中断 0×16 。中断 0×09 是无论何时你按下或放开键都会被激活的由硬件产生的中断。这个中断处理所有用键盘的低级通信, 并把击键翻译成两字节的码, 我们称之为扫描 ASCII 码。这些码随后被放入键盘缓冲区, 就绪于发往你的应用。

表 3-1 显示了按键的所有可能组合的扫描 ASCII 码对应关系。你会注意到该表并未显示所有可想象的按键组合。这是因为 BIOS 并未把它们全部翻译。该表只显示那些被翻译的。

扫描 ASCII 码的高位字节叫扫描码, 并且代表了实际按下的键的映像。扫描码不反映 shift、control 或 alt 键的状态, 且不是唯一的。低位字节是 ASCII 码, 并且如果被打印, 将产生与该键联系的适当的可打印字符。ASCII 码也不是唯一的: 如: tab (制表) 键和 Ctrl-I 键两者都产生相同的 ASCII 码 0×09 , 尽管它们的扫描码不同。值得庆幸的是扫描码和 ASCII 码的完整组合是唯一的, 稍后我们将看到如何利用它。

任何时候从 DOS 得到一个键, 它都是从键盘缓冲区得到, 剥去扫描码, 并仅仅返回 ASCII 码。然而, 某些键没有 ASCII 码, 所以 DOS 将返回一个空码, 并把扫描码放回键盘缓冲区为下一次读做准备。这样, 如果在接收到一个空码后你请求另一键, 则 DOS 将给出相关联的扫描码。用这种方式, 首先寻找一个空码, 之后做另一读操作取实际功能码就能检查某些控制和功能键。但这种方法可能不方便, 因为它迫使你对一个序列中的每个键做不同数目的读操作。一种较容易的办法是干脆把两个码做为一个整数读取, 而如果你想把该键存入一个字符串则只须屏蔽掉扫描码。

怎么实现呢? 原来, BIOS 中只有一中断处理程序用于从键盘缓冲区中取键: 中断 0×16 。发送适当的功能码, 你就能调用中断 0×16 来查看一个键是否在等待被读, 或实际获取该键。有幸的是你不必直接处理任何中断, 因为 Turbo C 提供了一个叫做 `bioskey()` 的内部函数, 它为我们做低级接口。它有如下参数:

- 0 - get the next key from the buffer, wait if necessary
- 1 - just look to see if a key is ready in the buffer
- 2 - get modifier status

前两个功能把扫描码做为一个整数返回。注意，它们不回映输入的键。这给你以比在有效输入和文本编辑中所做的把字符回映到屏幕的更好的控制。

最后一个功能，当前的 shift 键状态返回如下：

```
0x80 Insert Key toggled on
0x40 Caps Lock on
0x20 Num Lock on
0x10 Scroll Lock on
0x08 Alt pressed
0x04 Ctrl pressed
0x02 Left Shift down
0x01 Right Shift down
```

这些码被 OR 连接在一起以给出一个返回值。在下面的情形中恢复变换状态是有用的。某些按键组合不被 BIOS 翻译，因此在缓冲区中没有为它们放置键；例如 ALT-ENTER。许多弹出的驻留内存的程序使用这些“鬼”组合提供与通常键盘使用不相冲突的热键。

变换状态对于通过 NUMLOCK 状态把光标小键盘转化为数字小键盘也是有用的。每次从光标小盘上读取一个键时，你都可检查 NUMLOCK 的状态以决定是作为光标还是数字来解释该键。例如 PGUP 和 9 在数字小键盘上共享同一扫描 ASCII 码。对该码的解释由你决定。

做为一个做直接 BIOS 键存取的例子，下面的程序将等待每一个键，随后打印出它的扫描码，ASCII 码并试图打印该字符。它在遇到转换键 (ESC) 时终止。注意被提供用来从扫描 ASCII 码中取单个字节的有用的宏。程序如下：

```
/* Keyboard I/O example (ioex1.c) */
#include <stdio.h>
#include <bios.h> /* Turbo C's BIOS routines header */
/* macros to retrieve low and high byte of an integer */
#define lo(f) ((f) & 0xff)
#define hi(f) (lo(f) >> 8)
void main() {
    int key;
    do {
        key = bioskey(0); /* wait for key */
        printf("0x%02x 0x%02x %c\n", hi(key), lo(key), lo(key));
    } while( key != 0x011b );
}
```

别试图记住每个扫描 ASCII 码，要做的最好的事情是建一个这些码的有用的助记名头文件。例子作为清单 3.5 的一部分给出，它在稍后一个应用程序中将用到。它也有助于使表 3.1 方便可用。

总之，有三种基本方法来获取一个键：捕捉中断 0x09，调用中断 0x16，和通过标

准 I/O 例行程序由 DOS 取得。哪个更好些？除非你要写键盘宏程序或弹出的驻留内存的程序，捕捉中断 0×09 决非好主意。这主要因为你将必须自己把硬件键码译为扫描 ASCII 码。你还得处理当键被按下和放开时中断 0×09 都被调用这一事实。

使用中断 0×16 原是一种取键的方便方法，因为你在一次读操作中可取整个键码。但是，它确实有一个缺点：由于你不通过 DOS 取这些键，所以使用 CTRL-BREAK 或 CTRL-C 将不能使某些程序停止(例如 Turbo C (环境)，所以在最初调试阶段，只在你的程序一进入无限循环，这就很容易使你的计算机挂起。另外，使用标准 I/O 例行程序有允许你使用 I/O 重定向的好处。恰当的选择依赖于应用。以后的章节既使用 BIOS 又使用标准 I/O 调用。你也可写自己的有用于弹出窗口应用程序的 CTRL-C 捕捉程序。

Table 3.1 Scan-ASCII Key Code Mappings

| Key | Code | Name | Key | Code | Name | Key | Code | Name |
|-----|--------|----------|-----|--------|--------|-----|--------|------------|
| 1 | 0x0000 | ctrl brk | 41 | 0x1100 | alt w | 81 | 0x1c0a | ctrl enter |
| 2 | 0x011b | esc | 42 | 0x1265 | e | 82 | 0x1e61 | a |
| 3 | 0x0231 | 1 | 43 | 0x1245 | E | 83 | 0x1e41 | A |
| 4 | 0x0221 | ! | 44 | 0x1205 | ctrl e | 84 | 0x1e01 | ctrl a |
| 5 | 0x0332 | 2 | 45 | 0x1200 | alt e | 85 | 0x1e00 | alt a |
| 6 | 0x0340 | @ | 46 | 0x1372 | r | 86 | 0x1f73 | s |
| 7 | 0x0300 | ctrl @ | 47 | 0x1352 | R | 87 | 0x1f53 | S |
| 8 | 0x0433 | 3 | 48 | 0x1312 | ctrl r | 88 | 0x1f13 | ctrl s |
| 9 | 0x0423 | # | 49 | 0x1300 | alt r | 89 | 0x1f00 | alt s |
| 10 | 0x0534 | 4 | 50 | 0x1474 | t | 90 | 0x2054 | d |
| 11 | 0x0524 | \$ | 51 | 0x1454 | T | 91 | 0x2044 | D |
| 12 | 0x0635 | 5 | 52 | 0x1414 | ctrl t | 92 | 0x2004 | ctrl d |
| 13 | 0x0625 | % | 53 | 0x1400 | alt t | 93 | 0x2000 | alt d |
| 14 | 0x0736 | 6 | 54 | 0x1579 | y | 94 | 0x2166 | f |
| 15 | 0x075e | ^ | 55 | 0x1559 | Y | 95 | 0x2146 | F |
| 16 | 0x071e | ctrl ^ | 56 | 0x1519 | ctrl y | 96 | 0x2106 | ctrl f |
| 17 | 0x0837 | 7 | 57 | 0x1500 | alt y | 97 | 0x2100 | alt f |
| 18 | 0x0826 | & | 58 | 0x1675 | u | 98 | 0x2267 | g |
| 19 | 0x0938 | 8 | 59 | 0x1655 | U | 99 | 0x2247 | G |
| 20 | 0x092a | * | 60 | 0x1615 | ctrl u | 100 | 0x2207 | ctrl g |
| 21 | 0x0a39 | 9 | 61 | 0x1600 | alt u | 101 | 0x2200 | alt g |
| 22 | 0x0a28 | (| 62 | 0x1769 | i | 102 | 0x2368 | h |
| 23 | 0x0b30 | 0 | 63 | 0x1749 | I | 103 | 0x2348 | H |
| 24 | 0x0b29 |) | 64 | 0x1709 | ctrl i | 104 | 0x2308 | ctrl h |
| 25 | 0x0c2d | - | 65 | 0x1700 | alt i | 105 | 0x2300 | alt h |
| 26 | 0x0c5f | _ | 66 | 0x186f | o | 106 | 0x246a | j |
| 27 | 0x0c1f | ctrl - | 67 | 0x184f | O | 107 | 0x244a | J |
| 28 | 0x0d3d | = | 68 | 0x180f | ctrl o | 108 | 0x240a | ctrl j |
| 29 | 0x0d2b | + | 69 | 0x1800 | alt o | 109 | 0x2400 | alt j |
| 30 | 0x0e08 | bs | 70 | 0x1970 | p | 110 | 0x256b | k |
| 31 | 0x0e7f | ctrl bs | 71 | 0x1950 | P | 111 | 0x254b | K |
| 32 | 0x0f09 | tab | 72 | 0x1910 | ctrl p | 112 | 0x250b | ctrl k |
| 33 | 0x0f00 | bk tab | 73 | 0x1900 | alt p | 113 | 0x2500 | alt k |
| 34 | 0x1071 | q | 74 | 0x1a5b | [| 114 | 0x266c | l |
| 35 | 0x1051 | Q | 75 | 0x1a7b | { | 115 | 0x264c | L |

| | | | | | | | | |
|----|--------|--------|----|--------|--------|-----|--------|--------|
| 36 | 0x1011 | ctrl q | 76 | 0x1a1b | ctrl [| 116 | 0x260c | ctrl l |
| 37 | 0x1000 | alt q | 77 | 0x1b5d |] | 117 | 0x2600 | alt 1 |
| 38 | 0x1177 | w | 78 | 0x1b7d | } | 118 | 0x273b | ; |
| 39 | 0x1157 | W | 79 | 0x1b1d | ctrl] | 119 | 0x273a | : |
| 40 | 0x1117 | ctrl w | 80 | 0x1c0d | enter | 120 | 0x2827 | ' |

| Key | Code | Name | Key | Code | Name | Key | Code | Name |
|-----|--------|--------|-----|--------|-------------------|-----|--------|------------------|
| 121 | 0x2822 | " | 162 | 0x3920 | space | 203 | 0x5b00 | shift f8 |
| 122 | 0x2960 | ' | 163 | 0x3b00 | f1 | 204 | 0x5c00 | shift f9 |
| 123 | 0x297e | ~ | 164 | 0x3c00 | f2 | 205 | 0x5d00 | shift f10 |
| 124 | 0x2b5c | \ | 165 | 0x3d00 | f3 | 206 | 0x5e00 | ctrl f1 |
| 125 | 0x2b7c | | 166 | 0x3e00 | f4 | 207 | 0x5f00 | ctrl f2 |
| 126 | 0x2b1c | ctrl \ | 167 | 0x3f00 | f5 | 208 | 0x6000 | ctrl f3 |
| 127 | 0x2c7a | z | 168 | 0x4000 | f6 | 209 | 0x6100 | ctrl f4 |
| 128 | 0x2c5a | Z | 169 | 0x4100 | f7 | 210 | 0x6200 | ctrl f5 |
| 129 | 0x2c1a | ctrl z | 170 | 0x4200 | f8 | 211 | 0x6300 | ctrl f6 |
| 130 | 0x2c00 | alt z | 171 | 0x4300 | f9 | 212 | 0x6400 | ctrl f7 |
| 131 | 0x2d78 | x | 172 | 0x4400 | f10 | 213 | 0x6500 | ctrl f8 |
| 132 | 0x2d58 | X | 173 | 0x4700 | home | 214 | 0x6600 | ctrl f9 |
| 133 | 0x2d18 | ctrl x | 174 | 0x4737 | shift home | 215 | 0x6700 | ctrl f10 |
| 134 | 0x2d00 | alt x | 175 | 0x4800 | up arrow | 216 | 0x6800 | alt f1 |
| 135 | 0x2e63 | c | 176 | 0x4838 | shift up | 217 | 0x6900 | alt f2 |
| 136 | 0x2e43 | C | 177 | 0x4900 | pgup | 218 | 0x6a00 | alt f3 |
| 137 | 0x2e03 | ctrl c | 178 | 0x4939 | shift pgup | 219 | 0x6b00 | alt f4 |
| 138 | 0x2e00 | alt c | 179 | 0x4a2d | grey - | 220 | 0x6c00 | alt f5 |
| 139 | 0x2f76 | v | 180 | 0x4b00 | left arrow | 221 | 0x6d00 | alt f6 |
| 140 | 0x2f56 | V | 181 | 0x4b34 | shift left arrow | 222 | 0x6e00 | alt f7 |
| 141 | 0x2f16 | ctrl v | 182 | 0x4c35 | num 5 | 223 | 0x6f00 | alt f8 |
| 142 | 0x2f00 | alt v | 183 | 0x4d00 | right arrow | 224 | 0x7000 | alt f9 |
| 143 | 0x3062 | b | 184 | 0x4d36 | shift right arrow | 225 | 0x7100 | alt f10 |
| 144 | 0x3042 | B | 185 | 0x4e2b | grey + | 226 | 0x7200 | ctrl prt scrn |
| 145 | 0x3002 | ctrl b | 186 | 0x4f00 | end | 227 | 0x7300 | ctrl left arrow |
| 146 | 0x3000 | alt b | 187 | 0x4f31 | shift end | 228 | 0x7400 | ctrl right arrow |
| 147 | 0x316e | n | 188 | 0x5000 | down arrow | 229 | 0x7500 | ctrl end |
| 148 | 0x314e | N | 189 | 0x5032 | shift down arrow | 230 | 0x7600 | ctrl pgdn |
| 149 | 0x310e | ctrl n | 190 | 0x5100 | pgdn | 231 | 0x7700 | ctrl home |
| 150 | 0x310 | alt n | 191 | 0x5133 | shift pgdn | 232 | 0x7800 | alt 1 |
| 151 | 0x326d | m | 192 | 0x5200 | ins | 233 | 0x7900 | alt 2 |
| 152 | 0x324d | M | 193 | 0x5230 | o | 234 | 0x7a00 | alt 3 |
| 153 | 0x320d | ctrl m | 194 | 0x5300 | del | 235 | 0x7b00 | alt 4 |
| 154 | 0x3200 | alt m | 195 | 0x532e | shift del | 236 | 0x7c00 | alt 5 |
| 155 | 0x332c | , | 196 | 0x5400 | shift f1 | 237 | 0x7d00 | alt 6 |
| 156 | 0x333c | < | 197 | 0x5500 | shift f2 | 238 | 0x7e00 | alt 7 |
| 157 | 0x342e | . | 198 | 0x5600 | shift f3 | 239 | 0x7f00 | alt 8 |
| 158 | 0x343e | > | 199 | 0x5700 | shift f4 | 240 | 0x8000 | alt 9 |
| 159 | 0x352f | / | 200 | 0x5800 | shift f5 | 241 | 0x8100 | alt 0 |
| 160 | 0x353f | ? | 201 | 0x5900 | shift f6 | 242 | 0x8200 | alt - |
| 161 | 0x372a | grey * | 202 | 0x5a00 | shift f7 | 243 | 0x8300 | alt = |
| | | | | | | 244 | 0x8400 | ctrl pgup |

3.2 基本文本输出

除了打印屏幕的标准 C 例程序外 (名为 `printf()`, `putchar ()`, 和 `puts()`) , Turbo C 还提供对窗口化应用有用的别的函数。例程序 `cprintf()`, `cputs()`, 和 `putch()` 模拟它们的标准 I/O 对等程序; 另外, 它们也知道当前使用的窗口和把所有的输出保持在边界以内。在解释怎样使用这些函数之前看看在低级层次怎样访问屏幕是有用的。

3.3 直接视频存取

屏幕被组织为 80×25 的称为文本单元的两字节单元网格, 如图 3.1 所示。每个文本单元由一个字符码和一个属性组成。该字符码是含有各种各样的图形字符的扩展 ASCII 码。属性码控制文本前景和背景颜色, 也控制下划线和闪烁。图 3.2 表明单色显示的属性字节的前景和背景映像关系, 图 3.3 表明彩色显示的映像。给定的值是由系统设置的。较新的显示卡 (即 EGA, VGA) 有更多的可用的颜色, 你可通过改变调色板来使用它们。

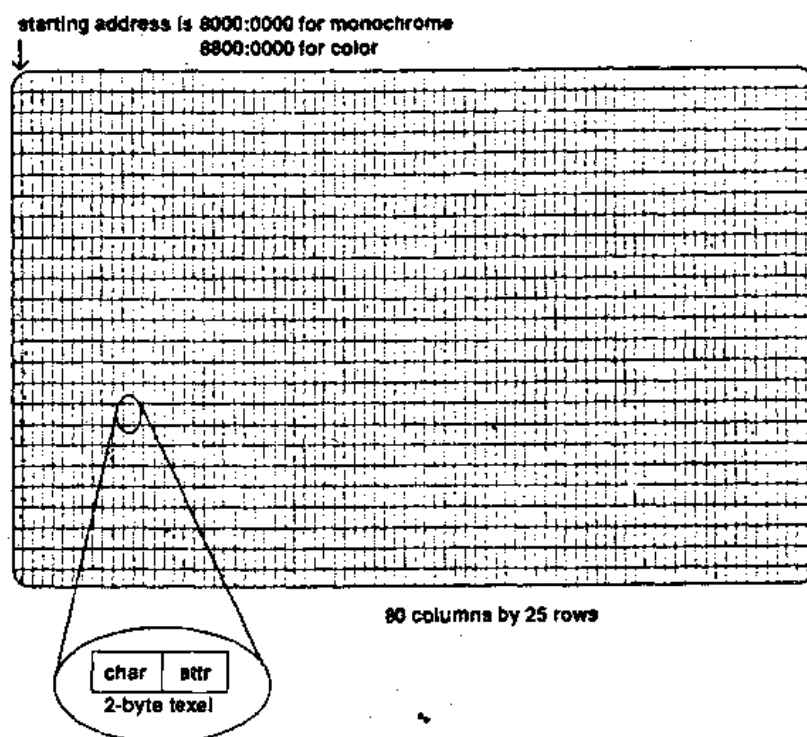
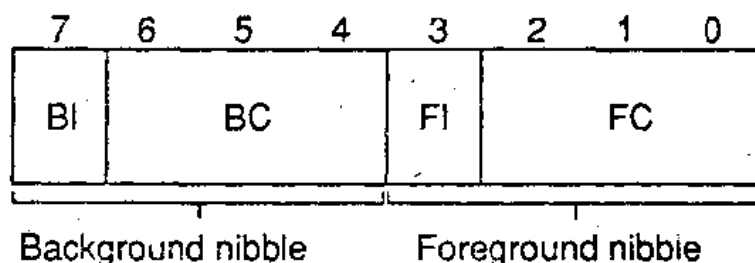


Figure 3.1 Screen organization in text mode



BI = 0 No blink / low intensity *
1 blink / high intensity

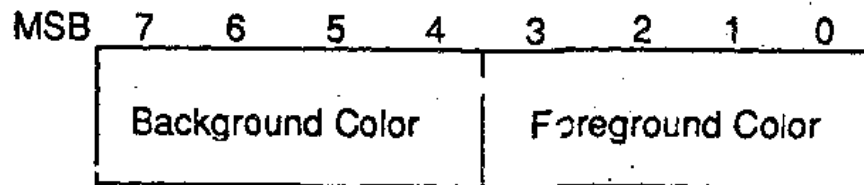
BC = 000 Black
111 White

FI = 0 low intensity
1 high intensity

FC = 000 black
001 white/underline
111 white

* Depends on hardware Blink Enable Bit (default is blink)

Figure 3.2 Attribute mapping for monochrome mode



low intensity

high intensity *

| | | | |
|------|---------|------|---------------|
| 0000 | Black | 1000 | Grey |
| 0001 | Blue | 1001 | Light Blue |
| 0010 | Green | 1010 | Light Green |
| 0011 | Cyan | 1011 | Light Cyan |
| 0100 | Red | 1100 | Light Red |
| 0101 | Magenta | 1101 | Light Magenta |
| 0110 | Brown | 1110 | Light Brown |
| 0111 | White | 1111 | Bright White |

* Only available for background color if Blink Enable Bit is off. Otherwise, foreground blinks and background takes low intensity color.

Figure 3.3 Attribute mapping for color modes

属性字节的第7位服务于双重目的：它或是控制前景字符闪烁，或是作为背景颜色的高亮度位。所用的状态由你的显示卡的闪烁的当前状态决定。当选择一个新的文本模式时，BIOS就把闪烁使能位置为1以便属性字节的第7位引起闪烁。如果你希望得到高亮

度背景颜色则必须复位闪烁位。进一步的详情参见你的显示卡技术参考手册。

文本单元网格驻留在你的显示卡上的 RAM 中并且可以象普通内存那样存取。地址决定于你使用的卡。对单色图形适配器，它从地址 $0 \times 6000: 0000$ 开始，第一个数是段地址，第二个数是偏移量。对 CGA、EGA 和 VGA 卡，通常起始地址是 $0 \times 6800: 0000$ 。这些卡中某些有具有别的地址的多个页面，但这里不处理之，因为多数时间使用的是高层次 Turbo-c 调用。

作为一个例子，你将看到用直接存取怎样写一个在屏幕上打印“Hello,World”信息的程序。为做到这一点，先创建一个代表一个文本单元的结构是很有用的：

```
typedef struct texel_struct {
    unsigned char ch;
    unsigned char attr;
} texel;
```

用这个结构，你可以定义网格本身，如下所示。注意：文本单元按行存储。

```
typedef texel screen_array[25][80];
```

为访问屏幕，你可以定义一个指针并把它初始化为适当的地址。由于屏幕将总是驻留在任何程序段之外，所以你应该把它说明为长指针 (far)。这样，同一程序可用于任何内存模式。

```
/* for monochrome cards */
screen_array far *screen_ptr = (screen_array far *) 0xb0000000L;
/* for color cards */
screen_array far *screen_ptr = (screen_array far *) 0xb8000000L;
```

随后说明下面的宏是很有用的，说明之后你就可把屏幕看成是数组：

```
#define screen (*screen_ptr)
```

来访问。例如，要在行 13.15 处写字符 A，你可用 `screen[1][5].ch = 'A';`

你可用：`screen[1][5].attr = 112;`

把它的颜色设置为反相显示。注意：与所有 C 数组一样，下标从 0 开始；这样屏幕顶角为 `screen[0][0]`。

作为此例的结束，下面是为彩色卡写的著名的“Hello World”程序：

```
/* Low level screen access example (ioex2.c) */
#include <string.h>
#include <conio.h>
#define screen (*screen_ptr)
typedef struct texel_struct {
    unsigned char ch;
    unsigned char attr;
} texel;
typedef texel screen_array[25][80];
screen_array far *screen_ptr = (screen_array far *) 0xb8000000L;
char hello[] = "Hello world";
void main() {
    int i;
    /* write out the characters to row 10, col 0 */
```

```

    for (i = 0; i < strlen(hello); i++) {
        screen[10][i].ch = hello[i];
    }
    getch(); /* wait for key */
    /* Now, for fun, lets change its color to reverse video */
    for (i = 0; i < strlen(hello); i++) {
        screen[10][i].attr = 112;
    }
}

```

运行此程序时你注意到了雪状物吗？如果你用的是旧图形卡之一，则向视频存储直接写入会引起视频跟踪干扰。多数商业程序绕过此问题，其方法是捕捉由该图形卡产生的某种水平折回中断，随后等待适当的时间来写向屏幕。你不必处理之，因为可用 TurboC 的屏幕例行程序未实现。如果你用函数 `cprintf()`、`cputs()` 和 `putch()`，它们将直接写向 RAM，但只是在适当的时候。

虽然 `cPrintf()`、`cputs()` 和 `putch()` 例行程序通常直接到 RAM，但你仍可以把全程变量 `directvideo` 置为 0 以通过 BIOS。这是为 BIOS 兼容但视频硬件不兼容的计算机提供的。然而，通过 BIOS 确实是慢，做一个试验，当你敲入稍后将给出的弹出窗口程序时，试将直接视频位在程序之初置为 0。随后你将看到我指的是什么。

甚至在直接视频模式使用 Turbo C 函数也比直接写向 RAM 慢，主要因为你通过函数调用访问屏幕而非简单地写向一个数组。除非你用的是低速计算机，这种交易是值得的，因为它为你考虑了雪状物，而且通过使用高层次调用，你的程序可更容易地移植到将来的操作系统上。

3.4 Turbo C 窗口

Turbo C 提供了一个允许你在屏幕上定义一些能独立于屏幕其它部分卷动的长方形区域的内部函数。这就是 `window()` 函数。你可用这个函数使屏幕上的一个区域成为活动窗口。一旦完成了这一点，例行程序 `cprintf()`、`putch()` 和 `puts` 将自动在此区域内打印及卷动。还有一些可用于做自己的卷动和编辑函数的支持清除窗口(`clrscv(1)`)插入和删除行(`insline()`，`define()`) 例行程序。

Turbo C 还提供控制与当前窗口相关的光标及返回其状态的函数，它们是 `gotoxy()`，`wherex()`，和 `wherey()`。还有个叫做 `gettextinfo()` 的函数，它返回当前窗口坐标，当前使用的颜色，和光标当前位置。

从一个简单的例子开始，下面的程序将建一个 20 列 4 行的窗口在屏幕中心，把光标移到行 4 列 2，随后调用 `putch()` 回映敲入的键。按下 ESC 键程序清屏并退出。

```

/* Turbo C window example (ipex3.c) */
#include <conio.h> /* include this whenever you use windows */
void main() {
    int c;
    window(30,10,49,13); /* window at upper left (30,10), */
                          /* lower right (49,13) */
    gotoxy(0,0);          /* x is column, y is row */
    do {

```

```

    c = getch();
    putchar(c);
} while(c != 0x1b);
window(1,1,80,25);
clrscr();
}

```

这个程序遗漏的是窗口周围的框。Turbo C 不会为你画一个。清单 3.1 (Listing 3.1) 中的例行程序 my_box() 将为你画一个框。实际上它甚至允许你选择边界的类型。注意窗口坐标是怎样在框坐标之内的。否则，框轮廓也将被覆盖！你必须在设置窗口之前画框，否则字符将写在与窗口相关的位置上，而且将产生一个被窜改的屏幕。

Listing 3.1 Drawing Boxed Windows (IOEX4.C)

```

/* Drawing text boxes example (ioex4.c) */
#include <conio.h>
void my_box(int xul, int yul, int xlr, int ylr, int btype);
void main() {
    int c;
    my_box(29,9,50,14,2); /* double line with reverse video */
    window(30,10,49,13); /* window at upper left (30,10), */
                          /* lower right (49,13) */
    gotoxy(0,0); /* x is column, y is row */
    do {
        c = getch();
        putchar(c);
    } while(c != 0x1b);
    window(1,1,80,25);
    clrscr();
}

void my_box(int xul, int yul, int xlr, int ylr, int btype)
/* Draws a box at the upper left (xul,yul) and lower right
(xlr,ylr) coordinates.
If btype = 0, no box is drawn.
If btype = 1, a single line box is drawn.
If btype = 2, a double line box is drawn.
*/
{
    static int boxcar[2][6] = { /* graphics characters for a box */
        {218,196,191,179,192,217}, /* single line box */
        {201,205,187,186,200,188} /* double line box */
    };
    int i, hzchar, vtchar;
    if (btype) {
        hzchar = boxcar[btype-1][1];
        vtchar = boxcar[btype-1][3];
        /* draw top and bottom sides */
        gotoxy(xul,yul);
        for (i=xul; i<=xlr; i++) putchar(hzchar);
    }
}

```

```

gotoxy(xul,ylr);
for (i=xul; i<=xlr; i++) putch(hzchar);
/* draw vertical sides */
for (i=yul; i<=ylr; i++) {
    gotoxy(xul,i);
    putch(vtchar);
    gotoxy(xlr,i);
    putch(vtchar);
}
/* draw corners */
gotoxy(xul,yul); putch(boxcar[btype-1][0]); /* upper left */
gotoxy(xlr,yul); putch(boxcar[btype-1][2]); /* upper right */
gotoxy(xlr,ylr); putch(boxcar[btype-1][5]); /* lower right */
gotoxy(xul,ylr); putch(boxcar[btype-1][4]); /* lower left */
}
}

```

3.5 文本颜色

Turbo C 在下列函数中提供对改变写到屏幕上的文本的颜色的支持:

| | |
|----------------|--|
| textattr | - sets both background and foreground colors in attribute byte |
| textbackground | - sets only the background color |
| textcolor | - sets only the foreground color |
| highvideo | - sets high-intensity bit of foreground color |
| normvideo | - sets both foreground and background colors back to what they were on program startup |
| lowvideo | - resets the high-intensity bit of foreground color |

另外, 如果你有 EGA 或 VGA 适配器, 则还有设置和恢复颜色调色板信息的例行程序。关于这的讨论见 Turbo C 使用手册。

Turbo C 提供的这套函数遗漏的一件事是一种直接只设置屏幕文本颜色而不重写字符的方法。换句话说, 是与使用: `screen[0][0].attr = 112;`

等价的某件事, 它设置行 0 列 0 的属性字节以反相显示而不改变那儿的字符。这种功能特性很重要, 例如当突出显示菜单条时, 你不必记住当前在菜单中被突出显示项目的字符是什么。用 Turbo C 提供的两个别的函数 `gettext()` 和 `puttext()` 可绕过此问题 (而不直接写屏幕)。

`gettext()` 从屏幕上的一个区域拷贝存贮映像到一个用户提供的数组里。`Puttext` 做相反的事。按下面的方法你可以模拟只改变属性字节的效果: 假设你想把列 5 行 4 设置为蓝色 (`attr=1`)。下面的程序段表明怎样做:

```

texel t;
gettext(5,4,5,4, &t); /* get one texel */
t.attr = 1;           /* set color */
puttext(5,4,5,4, &t); /* put back text */

```

在扩展了该窗口例子以包括进文本突出显示后, 清单 3.2 (Listing 3.2) 中给出的程序允许你用箭头移动光标, 和用 INS 键触发突出显示开或关。这个程序象上节讨论过的那样使用键盘 I/O, 并且包含早先给出的框例行程序。

你将注意到的一件事是对正在使用的是哪种坐标系统保持记录 and 了解窗口边界是多么困难, 下一章将开发一套弹出窗口实用程序以使对所有这些详情保持记录的问题简化。

3.6 控制光标大小

BIOS 含有一种改变光标形状的方法, 从一个单个扫描行到一个 14 行的箱的任一地方。你甚至可以完全关掉光标。这对象菜单光条这样的东西是有用的, 菜单中有光标会把注意力从高光条本身分开。无法关闭光标, 因为它是硬件固有的。系统设置的光标是一个闪烁的连字号, 它使用字符箱底的两行。

为改变光标形状, 你可调用 BIOS 中断 0×10 , 用功能号 0×01 。随后在 CH 和 CL 中传递两个参数, 分别代表光标的起始和终止行, 如下列程序段所示:

```
regs.h.ah = 0x10;
regs.h.ch = startline;
regs.h.cl = endline;
int86(0x10, &reg, &regs);
```

每个字符在文本状态下占 14 扫描行, 并且这些行从顶部的行 0 到底部的行 13 编号。这样, 如果你设置 CH=0x00, 及 CL=0x0d, 则将显示一个全大小的光标。用 CH=0x0d 及 CL=0x0d, 光标变成一个在字符单元底部的闪烁的连字号。下面的程序允许你用不同的光标形状 (只是对每种不同形状返回之) 做实验并且表明怎样通过 Turbo C 和 int86() 函数做适当的 BIOS 调用:

```
/* Changing cursor shape example (ioex5.c) */
#include <dos.h>
#include <stdio.h>
void cursor_size(int startline, int endline);
void main() {
    int s,e;
    printf("enter starting line and ending line: \n");
    scanf("%d %d", &s,&e);
    cursor_size(s,e);
    printf("Press return to exit\n"),
    getch();
}
void cursor_size(int startline, int endline)
{
    union REGS regs;
    regs.h.ch = startline; /* starting cursor line */
    regs.h.cl = endline; /* ending cursor line */
    regs.h.ah = 1; /* Call BIOS with the set cursor size */
    int86(0x10, &regs, &regs); /* function code in ah */
}
```

如果你有一个 MCGA、EGA 或 VGA 适配器, 则你可能会注意到这个程序的奇怪之

处。尽管在这些适配器上光标大小是 14 行高，但你只能给其中 8 个单个编址。这是为了与较旧的彩色适配器的向后兼容。所用的映像在这些适配器之间变化，所以要做的最好的事是仅仅只用值做实验。由于这个问题，建一个设置光标大小的通用例行程序可能很难，因为你得知道正在使用的是哪种适配器及它在什么状态。值得庆幸的是，Turbo C 在它的图形软件包中提供了函数 `detectgraph()` 来帮助你。

你可调用同一中断例行程序并设置 CH 寄存器的第 5 位来关掉光标。下面的函数是我们的 `cursor-size()` 函数的修改形式，它允许你通过传递 0 起始和终止码来关掉光标。

```
void cursor_size(int startline, int endline)
{
    union REGS regs;
    /* erase cursor on (0,0) input, else set cursor size */
    if ((startline == 0) && (endline == 0)) {
        regs.h.ch = 0x20; /* set bit 5 of ch to erase cursor */
    }
    else {
        regs.h.ch = startline; /* starting cursor line */
        regs.h.cl = endline; /* ending cursor line */
    }
    regs.h.ah = 1; /* Call BIOS with the set cursor size */
    int86(0x10, &regs, &regs); /* function code in ah */
}
```

只要用有效的起始和终止行调用 `cursor-size()` 可再次打开光标。

Listing 3.2 The Highlighting Example (IOEX6.C)

```
/* Text highlighting example (ioex6.c) */
#include <bios.h>
#include <conio.h>
#define UPKEY      0x4800
#define DOWNKEY    0x5000
#define LEFTKEY    0x4b00
#define RIGHTKEY   0x4d00
#define INSKEY     0x5200
#define ESCKEY     0x011b
typedef struct texel_struct {
    unsigned char ch;
    unsigned char attr;
} texel;
void my_box(int xul, int yul, int xlr, int ylr, int btype);
void main() {
    int highlite = 1;
    int x = 1, y = 1, k;
    texel t;
    my_box(29, 9, 50, 14, 2);
    window(30, 10, 49, 13);
    printf("Here is some text\r\n");
```



```

cprintf("for you to highlite\r\n");
cprintf("Use the INS KEY to\r\n");
cprintf("toggle highliting");
do {
    gotoxy(x,y);
    if (highlite) { /* these are absolute (full screen) coords !!! */
        gettext(x+29,y+9,x+29,y+9,&t);
        t.attr = 112;
        puttext(x+29,y+9,x+29,y+9,&t);
    }
    else { /* back to normal */
        gettext(x+29,y+9,x+29,y+9,&t);
        t.attr = 7;
        puttext(x+29,y+9,x+29,y+9,&t);
    }
    k = bioskey(0);
    switch(k) {
        case UPKEY: /* remember coords are relative to window */
            if (--y < 1) y = 1;
            break;
        case DOWNKEY:
            if (++y > 4) y = 4;
            break;
        case LEFTKEY:
            if (--x < 1) x = 1;
            break;
        case RIGHTKEY:
            if (++x > 20) x = 20;
            break;
        case INSKEY:
            highlite = !highlite;
            break;
        default: ;
    }
} while (k != ESCKEY);
window(1,1,80,25);
clrscr();
}

void my_box(int xul, int yul, int xlr, int ylr, int btype)
/* Draws a box at the upper left (xul,yul) and lower right
(xlr,ylr) coords with given attribute. If btype = 0, no box is drawn,
if btype = 1, a single line box is drawn, if btype = 2, a double line
box is drawn.
*/
{
    static int boxcar[2][6] = { /* graphics characters for a box */
        {218,196,191,179,192,217}, /* single line box */
        {201,205,187,186,200,188} /* double line box */
    }

```

```

};
int i, hzchar, vtchar;
if (btype) {
    hzchar = boxcar[btype-1][1];
    vtchar = boxcar[btype-1][3];
    /* draw top and bottom sides */
    gotoxy(xul,yul);
    for (i=xul; i<=xlr; i++) putchar(hzchar);
    gotoxy(xul,ylr);
    for (i=xul; i<=xlr; i++) putchar(hzchar);
    /* draw vertical sides */
    for (i=yul; i<=ylr; i++) {
        gotoxy(xul,i);
        putchar(vtchar);
        gotoxy(xlr,i);
        putchar(vtchar);
    }
    /* draw corners */
    gotoxy(xul,yul); putchar(boxcar[btype-1][0]); /* upper left */
    gotoxy(xlr,yul); putchar(boxcar[btype-1][2]); /* upper right */
    gotoxy(xlr,ylr); putchar(boxcar[btype-1][5]); /* lower right */
    gotoxy(xul,ylr); putchar(boxcar[btype-1][4]); /* lower left */
}
}
}

```

3.7 使用鼠标器

把鼠标器作为 I/O 设备的使用正日益普遍，尤其是随着 Macintosh 这样的计算机正把工业引向对它们的全面使用，情况更是如此。对于有 Microsoft 兼容的鼠标器的人，你将学到把它并入到你的 DOS 应用中去的基本知识。对其鼠标器不是 Microsoft 兼容的人，许多程序仍将是有益的，但确切细节将有所不同。本章的目的不是给你一个对鼠标器所有可能功能的详细讨论，而只是讲明足够日常应用的功能。

鼠标器按下列方式工作：首先，必须装入正确的鼠标器驱动程序把鼠标器作为系统的一部分适当地装配。通常这由 config.sys 文件在引导时间完成，或由在你的 autoexec.bat 文件中执行一个程序完成。这两种方法都将把鼠标器驱动例行程序作为内存驻留程序装配（详见你的鼠标器手册）。你的程序通过中断 0×33 与鼠标器通信。用这个中断，你能得到鼠标坐标、得到按钮状态，改变鼠标颜色和形状，使之可见和不可见，及执行许多别的功能。

3.8 基本鼠标器功能

一旦你已装配了鼠标器驱动程序并在你的程序中初始化了它之后，你就可以打开鼠标器，随后鼠标驱动程序为你做具体的事。例如，你不必保持鼠标运动记录及更新光标，驱动程序为你完成这些功能。你需做的一切只是轮询驱动程序鼠标器的状态变化。这可由使

用列于表 3.2 (Table 3.2) 和表 3.3 (Table 3.3) 中的函数代码来实现。这些表显示了在 Microsoft 鼠标驱动程序中有的基本功能，及这些功能所需的参数。只有部分有用功能列出了，而且它们将是主要的功能。还有许多别的有用的功能，所以进一步的详情见鼠标器技术参手册。对我们将要开发的程序来说，列出的这些功能已足够了。

Table 3.2 Mouse Function Codes and Parameters

| Code | Function | Parameters |
|------|----------------------------|--|
| 0 | Mouse reset | m1 = 0 (input) m1 = mouse status (output) (-1 if installed, else 0) m2 = number of buttons |
| 1 | Show cursor | m1 = 1 (input) |
| 2 | Hide cursor | m1 = 2 (input) |
| 3 | Button status | m1 = 3 (input) m2 = button status m3 = x cursor position m4 = y cursor position |
| 4 | Set cursor posn | m1 = 4 (input) m5 = new x position m4 = new y position |
| 5 | get button press info | m1 = 5 (input) m1 = button status (output) m2 = (input) 0 = left button checked 1 = right button checked m2 = number of button presses m3 = x position at last press m4 = y position at last press |
| 6 | get button release info | m1 = 6 (input) m1 = button status (output) m2 = (input) 0 = left button checked 1 = right button checked m3 = x position of last release m4 = y position of last release |
| 7 | set x bounds | m1 = 7 (input) m3 = minimum x position m4 = maximum x position |
| 8 | set y bounds | m1 = 8 (input) m3 = minimum y position m4 = maximum y position |
| 9 | set graphics cursor | m1 = 9 (input) m2 = cursor hot spot (x posn) m3 = cursor hot spot (y posn) m4 = pointer to cursor masks |
| Code | Function | Parameters |
| 10 | set text cursor | m1 = 10 (input) m2 = (input) 0 for s/w cursor, 1 for h/w cursor m3 = screen mask if m2 = 0 |

= scan line start if m2 = 1
 m4 = cursor mask if m2 = 0
 = scan line stop if m2 = 1

Table 3.3 Mouse Button Status Codes

| Value | 0 | 1 |
|-------|------------------|--------------------|
| Bit 0 | Left button up | Left button down |
| 1 | Right button up | Right button down |
| * 2 | Middle button up | Middle button down |

* For Logitech 3 button mouse only

对最常用的功能来说，四个参数 m1、m2、m3 和 m4 总是分别对应寄存器 ax、bx、cx 和 dx。这样，与驱动程序的接口就十分简单。清单 3.3 (Listing3.3) 表明如何用 Turbo C int86() 函数直接与之对话。mouse() 函数这样工作：装入带有参数的适当的寄存器，调用该中断，随后把寄存器装回参数。对别的功能，寄存器对应关系可能不同，所以更多的细节参见使用手册。

3.9 检查鼠标器驱动程序

在你的程序之初，你应检查鼠标器驱动程序是否已装入。这可用清单 3.4 中给出的函数 check-mouse-driver() 来实现。这个函数通过调用 Turbo C 的 getvect() 函数为鼠标器返回中断向量（中断 0×33）来工作。它随后检查一个 NULL 地址或一个 IRET (0×cf) 指令。如果发现了两者之一，则意味着无鼠标器；否则，它认为中断向量有效并指向一个实际的鼠标器驱动程序（确实无法确切判别）。

参数 need-mouse 被包括进来以便能使你的请求对是否有鼠标透明。基本上，如果未找到鼠标器驱动程序，且 need-Mouse=1，则打印一错误信息并且程序退出。如果你想使鼠标器成为可选择的，则设置 need-mouse=0。如果找到了鼠标驱动程序则函数返回 1，否则它返回 0。

3.10 鼠标器工具箱

Mouse() 和 check-Mouse-driver() 例行程序是与鼠标器通信所需的全部东西。然而，最好是建一个较高层次函数的工具箱，这一点我们在这里将建立。具体地讲，下列数在清单 3.5 和 3.6 中给出。

| Function | Usage |
|--------------------|--|
| check_mouse_driver | Checks for mouse being installed |
| init_mouse | Calls check_mouse_driver(), resets the mouse, turns on the mouse cursor, and initializes some internal variables |
| mouse | The low-level mouse interface |
| mouse_reset | Used to reset the mouse driver |

| | |
|------------------------------|---|
| <code>move_mouse</code> | Forces the mouse cursor to a new location |
| <code>mouse_on</code> | Conditionally turns the mouse cursor on |
| <code>mouse_off</code> | Conditionally turns the mouse cursor off |
| <code>mouse_text_posn</code> | Returns the mouse position in text coordinates |
| <code>mouse_grph_posn</code> | Returns the mouse position in graphics coordinates |
| <code>mouse_in_box</code> | Tests to see if the mouse is within a given rectangular region |
| <code>button_release</code> | Tests to see if a button has been released since last time called |
| <code>button_press</code> | Tests to see if a button has been pressed since last time called |
| <code>button_state</code> | Returns the up/down status of the buttons |
| <code>mouse_trigger</code> | Looks for either a key press or a mouse button press or release |

这些例行程序中的许多都直接仿照表 3.2 列出的功能。然而，它们还进行别的操作。基中之一是保持下列变量：

| Identifier | Meaning |
|--------------------------------|---|
| <code>low_resolution</code> | Flag used when in a low-resolution graphics mode (ie., 320 pixels horizontally) |
| <code>mouse_text_x</code> | X position of mouse in Turbo C text coordinates. |
| <code>mouse_text_y</code> | Y position of mouse in Turbo C text coordinates. |
| <code>mouse_grph_x</code> | X position of mouse in graphics coordinates. |
| <code>mouse_grph_y</code> | Y position of mouse in graphics coordinates. |
| <code>mouse_initialized</code> | Set to 1 if <code>mouse_init()</code> succeeded. |
| <code>prev_cursor_state</code> | Stores the previous visible/invisible state of the mouse cursor |

鼠标器头文件（见清单 3.5）中还提供了下列鼠标器按钮按下和放开的“键”码。这些码是一些无键分配给它们的值，所以我们可以安全地使用它们：

```
#define LEFT_MOUSE_PRESS    0xff01
#define RIGHT_MOUSE_PRESS   0xff02
#define LEFT_MOUSE_REL      0xff11
#define RIGHT_MOUSE_REL     0xff12
```

鼠标器头文件还有为常用鼠标器功能定义的一些宏，及作为一种便利，有一套常用键码的定义。基本上，任何时候你想做键盘或鼠标器 I/O 都可用此头文件。

函数 `init_mouse()` 是供初始化鼠标器之用的。它有下面的原型：

```
int init_mouse(int need_mouse, int graphdriver, int graphmode);
```

该函数调用 `check_mouse_driver()` 来检查鼠标器是否已被装配。它把参数 `need_mouse` 传给 `check_mouse_driver()` 供错误处理。这个参数可取下列定义于鼠标器头文件中的值：

| Macro | Code | Meaning |
|-----------------------------|------|--|
| <code>MOUSE_OPTIONAL</code> | 0 | Mouse is optional. Not an error if mouse driver not present. |

还传给 `init-mouse()` 的是两个指明正在使用的是何种屏幕模式的参数。我们将把对这些参数的讨论推迟到图形章节。对文本模式应用，把这两个参数设置为常量 `MOUSE-TEXT-MODE`，它在鼠标器头文件中被定义为 0。

尽管在全程变量中含有的参数中有许多都可在被调用的恰当的函数返回时得到，但有些数中有些实际上要获得比参数表中返回的更多的信息。例如，函数 `button-status()` 获得当前鼠标器位置，也获得按钮的上/下状态，所以它为了你的便利存储该位置。

大多数函数都很简单并且能很容易地从清单仿照。然而，有一些评价要小心。其中之一是所有这些函数检查鼠标器是否被初始化了。如果未初始化，则它们采取适当的措施使鼠标器初始的状态对你的程序尽可能透明。

认识到函数 `button-press()`、`button-release()`、和 `button-state()` 之间的差别是很重要的。前两个返回指明自该函数最后一次被调用以来按钮是否被按下（或）放开的值。`Button-state()` 返回一个指示按钮当前是否被按下的值。

典型的例子，你将用 `button-press()` 来开始一个鼠标器“引入”的对话，在鼠标器引入的同时检查 `button-state()`，而 `button-release` 终止该对话。为什么不一直用 `button-state()`？因为如果你不在适当的时候检查状态，你就可能漏掉按钮被按下或放开了（这种组合通称单的嗒声）。`button-press()` 和 `button-release` 把各自的事件存储到你确实调用它们为止。它们甚至返回自上次被调用以来发生了多少次。

注意，坐标变换发生在内部函数 `set-mouse-posn()`。Turbo C 从 (1, 1) 处开始其文本坐标。但是，鼠标器驱动程序总是按象素方式返回坐标并始于 (0, 0) 位置。`set-mouse-posn()` 函数认为 8 个象素对应一个文本单元（这对标准 80×25 模式是对的），所以它把坐标乘以 8 再加 1 来调整到 Turbo C 的文本坐标。

当你在低分 (320×200) 图形模式时就会产生一个问题。鼠标驱动器认为 (GA、EGA、VGA 的屏幕宽为 640 个水平象素。(对 Hercules 图形，它用 720 个象素)。如果你在低分模式，则 X 坐标须用除 2 来校正。这通过传给 `init-mouse()` 适当地 `graphdriver` 和 `graphmode` 值来实现，这将在图形章讨论。

3.11 鼠标器光标

所给函数的另一巧妙之处与鼠标器光标有关。你也许想知道它是怎么画出来的，因为普通光标也（通常）与鼠标器光标一起出现。普通光标在图形适配器硬件里实现并由 DOS 维持。鼠标器光标在软件里实现并由鼠标器驱动程序维持。驱动程序直接写向视频 RAM 来画光标。每次光标移动时，下面的文本就被存储并在光标离开该位置时恢复。

除非你碰巧覆盖了光标上的东西，这都将工作得很好。驱动程序无法知道此事已发生并将忠实地在光标离开该位置时用先前的内容替换文本单元。结果将是一个被窜改的屏幕。

走出此窘境的办法是在你将写屏幕的时候关掉鼠标器光标；以后，你可再打开。它比这要不易处理些，因为仅当它先前是打开的情况下你才会想再打开光标。为使它容易处

理, 提供了函数 `mouse-on()` 和 `mouse-off()`, 它们保持鼠标器光标先前状态的记录并允许你根据其先前状态视条件把光标打开。这些函数有下面的原型。

```
void mouse_on(int restoreflag);  
void mouse_off(int tempflag);
```

参数含意如下:

```
restoreflag  = 0: Unconditionally turn mouse cursor on.  
              = 1: Restore mouse cursor to previous state.  
tempflag     = 0: Turn mouse cursor off, set previous state  
                  to off.  
              = 1: Turn mouse cursor off only if it is on,  
                  set previous state to on.  
                  If mouse already off, set previous state  
                  to off.
```

下面给出一个有条件地关掉鼠标器光标, 更新屏幕, 随后又有条件地打开鼠标器光标的例子。

```
mouse_off(1);  
cprintf("Hello World");  
mouse_on(1);
```

如果你用参数 0 调用 `mouse-off` 和 `mouse-on()`, 则它们将无视先前状态而关掉或打开鼠标器。做这些工作时你必须十分小心, 因为鼠标器驱动程序把连续的鼠标器关掉的命令排成队列。如果你发了两个关掉鼠标器的请求, 而不在中间插入一个打开鼠标器的命令, 则它将用两个连续的打开鼠标器的命令打开鼠标器。

这一特殊的功能特性会引起极大的混淆, 所以最好是避免它。如果你在想暂时关掉鼠标器的时候用 `mouse-off(1)`, `mouse-on(1)` 序列, 则你永远不会有此问题。即将给出的弹出窗口程序严格遵守这种方法。

告诉鼠标驱动程序用硬件光标代替软件光标是可能的, 尽管通常应该为文本输入定位保存硬件光标并为鼠标器定位使用鼠标器光标。当你确有一鼠标器在系统中时, 保持其形状为全高度方框 (有可能改变其形状) 并使普通光标为一闪烁连字号时它工作得最好。这样你不会将它们混淆。改变鼠标器光标颜色也是可以的。系统设置值是使颜色与文本背后的颜色翻转, 这通常是令人满意的。

3.12 鼠标器突出显示例子

做为一个使用鼠标器的例子, 让我们回顾一下前一章我们的文本典型例子改变它使之通过鼠标器操作。主程序在清单 3.7 中给出并有如下操作: 当按下左鼠标器按钮时, 突出显示标志被触发。如果你随后保持按下左按钮把鼠标器拖过屏幕, 由根据突出显示触发的状态, 文本将变亮或变暗。只有窗口中的文本受突出显示的影响。注意对鼠标器是否真的移动了的检查。这是为了防止在此过程中随鼠标器光标随其开关而闪烁。还要注意你必须关掉它, 即使你只是突出显示文本, 因为鼠标器光标在移动时存储和恢复字符和属性两者。退出该程序, 按下右按钮即可。如果你碰巧有一个单按钮鼠标器, 则你必须修改该程序以允许别的退出方法。作为一种练习, 试改变该程序以使你在窗口之外按下右按钮时程

序退出。

既然你已经探讨了键盘、低级屏幕 I/O，和鼠标器 I/O，现在是转到更引人的应用，给出如下的弹出窗口的时候了。

Listing 3.3 The Mouse Interface Function

```
void mouse(int *m1, int *m2, int *m3, int *m4)
/*
   C to mouse driver interface via interrupt 0x33.
   Only supports functions 0-10.
```


See your mouse technical reference manual for the interface to other functions.

```
*/
{
    union REGS inregs, outregs;
    inregs.x.ax = *m1;
    inregs.x.bx = *m2;
    inregs.x.cx = *m3;
    inregs.x.dx = *m4;
    int86(0x33, &inregs, &outregs);
    *m1 = outregs.x.ax;
    *m2 = outregs.x.bx;
    *m3 = outregs.x.cx;
    *m4 = outregs.x.dx;
}
```

Listing 3.4 The Check Mouse Driver Function

```
int check_mouse_driver(int need_mouse)
/*
 * If need_mouse = 1, then abort program if no mouse.
 * Otherwise, if no mouse return 0, else return 1.
*/
{
    void far *address;
    /* get mouse interrupt vector address */
    address = getvect(0x33);
    /* look for NULL address or IRET instruction */
    if ((address == NULL) || (*(unsigned char *)address == 0xcf)) {
        if (need_mouse) {
            printf("Mouse driver NOT installed\n");
            exit(1);
        }
        else return 0;
    }
    return 1;
}
```

Listing 3.5 Source Code for the Mouse Tool Kit Header File (MOUSE.H)

```
/* Mouse toolkit header file */
/* Macros to retrieve low and high byte of an integer */
#define lo(f) ((f) & 0xff)
#define hi(f) (lo(f) >> 8)
/* Common scan-ascii codes */
#define CTRLC 0x2e03
#define CTRLH 0x2308
#define CTRLI 0x1709
#define CTRLJ 0x260c
#define CTRLK 0x250b
```

```

#define CTRLJ      0x240a
#define CTRLU      0x1615
#define CTRLR      0x1312
#define CRKEY      0x1c0d
#define CTRLCRKEY  0x1c0a
#define UPKEY      0x4800
#define DOWNKEY    0x5000
#define LEFTKEY    0x4b00
#define RIGHTKEY   0x4d00
#define SHFTLEFT   0x4b34
#define SHFTRIGHT  0x4d36
#define DELKEY     0x5300
#define INSKEY     0x5200
#define BSKEY      0x0e08
#define SPACEBAR   0x3920
#define PGUPKEY    0x4900
#define PGDNKEY    0x5100
#define SHFTUPKEY  0x4838
#define SHFTDNKEY  0x5032
#define SHFTPGUPKEY 0x4939
#define SHFTPGDNKEY 0x5133
#define HOMEKEY    0x4700
#define ENDKEY     0x4f00
#define ESCKEY     0x011b
#define ALT_D      0x2000
#define ALT_E      0x1200
#define ALT_I      0x1700
#define ALT_R      0x1300
#define ALT_S      0x1f00
#define ALT_T      0x1400
#define ALT_X      0x2d00
#define F10KEY     0x4400
/* Mouse pseudo "key" codes */
#define LEFT_MOUSE_PRESS 0xff01
#define RIGHT_MOUSE_PRESS 0xff02
#define LEFT_MOUSE_REL 0xff11
#define RIGHT_MOUSE_REL 0xff12
/* Mouse driver function codes */
#define M_RESET          0
#define M_SHOW_CURS      1
#define M_HIDE_CURS      2
#define M_GET_STATUS     3
#define M_SET_CURS       4
#define M_GET_PRESS      5
#define M_GET_REL        6
#define M_SET_X_BOUNDS   7
#define M_SET_Y_BOUNDS   8
#define M_SET_G_CURS     9

```

```

#define M_SET_T_CORS 10
/* define other constants */
#define MOUSE_NEEDED 1
#define MOUSE_OPTIONAL 0
#define MOUSE_TEXT_MODE 0
/* Mouse external variables */
extern int mouse_text_x;
extern int mouse_text_y;
extern int mouse_grph_x;
extern int mouse_grph_y;
extern int mouse_intialized;
/* Mouse function prototypes */
extern void mouse(int *m1, int *m2, int *m3, int *m4);
extern int check_mouse_driver(int need_mouse);
extern int init_mouse(int need_mouse, int gd, int gm);
extern int mouse_reset(void);
extern void move_mouse(int x, int y);
extern void mouse_on(int code);
extern void mouse_off(int code);
extern void mouse_grph_posn(int *x, int *y);
extern void mouse_txt_posn(int *x, int *y);
extern int mouse_in_box(int graphflag, int left, int right,
                        int top, int bottom);

extern int button_release(int b);
extern int button_press(int b);
extern int button_state(void);
extern int mouse_trigger(int button_dir);

```

Listing 3.6 The Mouse Tool Kit Source Code (MOUSE.C)

```

/*****
 * Mouse toolkit      (mouse.c)
 *****/
#include <bios.h>
#include <dos.h>
#include <conio.h>
#include <process.h>
#include <stdio.h>
#include "mouse.h"
/* <<<< Global variables >>>> */
int mouse_text_x; /* X posn of mouse in Turbo C text coordinates */
int mouse_text_y; /* Y posn of mouse in Turbo C text coordinates */
int mouse_grph_x; /* X posn of mouse in graphics coordinates */
int mouse_grph_y; /* Y posn of mouse in graphics coordinates */
int mouse_initialized = 0; /* Set to 1 if mouse_init() succeeds */
/* <<<< Internal variables >>>> */
/* Previous cursor state. */
/* 0 = mouse previously off, 1 means prev on. */

```

```

static int prev_cursor_state = 0;
/* Pointer to start of bios video data */
static char far *bios_video_area = (char far *)0x00400049L;
/* This variable is for graphics mode only. */
/* low_resolution = 1 if using 320x200 graphics */
static int low_resolution = 0; /* Leave at 0 for text mode */
static void set_mouse_posn(int *x, int *y); /* internal function */
static int low_res_mode(int gd, int gm); /* internal function */
void mouse(int *m1, int *m2, int *m3, int *m4)
/*
    C to mouse driver interface via interrupt 0x33.
    Only supports functions 0-10.
    See your mouse technical reference manual for the
    interface to other functions
*/
{
    union REGS inregs, outregs;
    inregs.x.ax = *m1;
    inregs.x.bx = *m2;
    inregs.x.cx = *m3;
    inregs.x.dx = *m4;
    int86(0x33, &inregs, &outregs);
    *m1 = outregs.x.ax;
    *m2 = outregs.x.bx;
    *m3 = outregs.x.cx;
    *m4 = outregs.x.dx;
}

int check_mouse_driver(int need_mouse)
/*
    * If need_mouse = 1, then abort program if no mouse.
    * Otherwise, if no mouse return 0, else return 1.
*/
{
    void far *address;
    /* get mouse interrupt vector address */
    address = getvect(0x33);
    /* look for NULL address or IRET instruction */
    if ((address == NULL) || (*(unsigned char *)address == 0xcf)) {
        if (need_mouse) {
            printf("Mouse driver NOT installed\n");
            exit(1);
        }
        else return 0;
    }
    return 1;
}

int init_mouse(int need_mouse, int gd, int gm)
/*

```

Initializes the mouse. If it can't, and need_mouse = 1, this routine exits the program. Else, it returns mouse_initialized = 1.

If using the mouse for graphics, there are two special cases:

- (1) If using Hercules graphics, we must let the mouse driver know we're really in graphics mode, cause it can't tell.
- (2) If using any graphics mode with 320 pixels horizontally, (i.e. low resolution), we must make sure that the mouse coordinates are scaled properly.

These cases are detected by the parameters gd (graphics driver) and parameters gm (graphics mode), as defined by Turbo C graphics. If you're just using text mode, set both to zero. When fixing up Hercules graphics, we're assuming page 0. If using page 1, set *bios_video_area = 5.

```

*/
{
    int ml;
    mouse_initialized = 0;
    if (check_mouse_driver(need_mouse)) {
        if (gd == 7) *bios_video_area = 6; /* Fix up Hercules mode */
        if (low_res_mode(gd, gm))
            low_resolution = 1; /* Fix up low resolution mode */
        ml = mouse_reset(); /* Start mouse at ground zero */
        if (ml) {
            mouse_initialized = 1; /* Set mouse init flag */
            move_mouse(0, 0); /* Set coords to top of screen */
            mouse_on(0); /* Turn mouse on for first time */
        }
        else {
            if (need_mouse) {
                printf("ERROR activating mouse ...\n");
                exit(1);
            }
        }
    }
    return mouse_initialized;
}

static int low_res_mode(int gd, int gm)
/* Returns 1 if in any graphics mode with 320 pixels horizontally
   Returns 0 otherwise
*/
{
    if (
        (gd == 1 || gd == 2 || gd == 8) && /* CGA, MCGA, ATT400 */
        (gm >= 0 && gm <= 3) /* 320 hx mode */
    )
        return 1;
}

```

```

    return 0;
}

int mouse_reset(void)
/* If the mouse was on, it turns it off.
   Then, the mouse state is reset.
*/
{
    int x1, m1, m2, m3, m4;
    mouse_off(1); /* Turn off only if it was on */
    m1 = M_RESET;
    mouse(&m1, &m2, &m3, &m4);
    set_mouse_posn(&m3, &m4); /* Initialize coord's */
    return m1;
}

void move_mouse(int x, int y)
/* Move mouse cursor to text position (x,y) */
{
    int m1, m2, m3, m4;
    if (!mouse_initialized) return;
    m1 = M_SET_CURS;
    m3 = x*8; m4 = y*8; /* convert to pixel coordinates */
    mouse(&m1, &m2, &m3, &m4);
    set_mouse_posn(&m3, &m4);
}

void mouse_on(int restoreflag)
/* restoreflag = 0 means you want the mouse on regardless of previous state.
   restoreflag = 1 means you want the mouse on only if it was on previously.
*/
{
    int m1, m2, m3, m4;
    if (mouse_initialized) {
        if (!restoreflag || prev_cursor_state) {
            m1 = M_SHOW_CURS;
            mouse(&m1, &m2, &m3, &m4);
            prev_cursor_state = 1;
        }
    }
}

void mouse_off(int tempflag)
/* If tempflag = 1, it means you want to turn the mouse off, and if
   it had been on, set previous state to on.
   If tempflag = 0, it means you want to turn mouse off, and regardless
   of whether it was on or not, set previous state to off.
*/
{
    int m1, m2, m3, m4;
    if (mouse_initialized) {
        if (prev_cursor_state) { /* Turn it off only if it was on */

```

```

        m1 = M_HIDE_CURS;
        mouse(&m1, &m2, &m3, &m4);
        /* leave prev_cursor_state alone if just turning off temporarily */
        if (!tempflag) prev_cursor_state = 0;
    }
}

void mouse_grpn_posn(int *x, int *y)
/* Returns the mouse text coordinates if the mouse is
   initialized, else it returns (0,0).
*/
{
    int m1, m2;
    if (mouse_initialized) {
        m1 = M_GET_STATUS;
        mouse(&m1, &m2, x, y);
        set_mouse_posn(x, y);
    }
    else {
        *x = 0; *y = 0; /* default to left hand corner */
    }
    return;
}

void mouse_txt_posn(int *x, int *y)
/* Returns the mouse text coordinates if the mouse is
   initialized, else it returns (1,1).
*/
{
    mouse_grpn_posn(x, y);
    *x = mouse_text_x;
    *y = mouse_text_y;
    return;
}

int mouse_in_box(int graphflag, int left, int top, int right, int bottom)
/* Returns 1 if the mouse is in the box given by the
   coordinates, 0 otherwise. The type of coordinates is
   given by graphflag. If graphflag = 1, the coordinates
   are assumed to be graphics coordinates, else they're
   text coordinates. Either way they are absolute coordinates.
   In order for this routine to work properly you must first
   call mouse_txt_posn, mouse_grph_posn, or any of the
   mouse functions that set the global mouse position variables.
   If mouse not initialized, it returns 0;
*/
{
    int x, y;
    if (mouse_initialized) {
        if (graphflag) {

```

```

        x = mouse_grph_x;
        y = mouse_grph_y;
    )
    else {
        x = mouse_text_x;
        y = mouse_text_y;
    }
    if ((y >= top) && (y <= bottom) &&
        (x >= left) && (x <= right)) return 1;
}
return 0;
}
int button_release(int b)
/*
    Looks for a left (b=0) or right (b=1) mouse button release.
    Returns 1 if the button has been released since the last time
    called, else returns 0. If no mouse installed, it returns 0.
*/
{
    int m1, m2, m3, m4;
    if (mouse_initialized) {
        m1 = M_GET_REL;
        m2 = b; /* which button */
        mouse(&m1, &m2, &m3, &m4);
        set_mouse_posn(&m3, &m4);
        if (m2) return 1;
    }
    return 0;
}
int button_press(int b)
/*
    Looks for a left (b=0) or right (b=1) mouse button press.
    Returns 1 if the button has been pressed since the last time
    called, else returns 0. If no mouse installed, it returns 0.
*/
{
    int m1, m2, m3, m4;
    if (mouse_initialized) {
        m1 = M_GET_PRESS;
        m2 = b; /* which button */
        mouse(&m1, &m2, &m3, &m4);
        set_mouse_posn(&m3, &m4);
        if (m2) return 1;
    }
    return 0;
}
int button_state()
/*

```


Returns up/down state of the mouse buttons. A button is down if its corresponding bit = 1.

Bit 2 Middle button (Logitech mouse only)
 1 Right button
 LSB 0 Left button

Returns 0x00 if no button down or no mouse installed.

Also stores mouse position information.

```

*/
{
    int m1,m2,m3,m4;
    if (mouse_initialized) {
        m1 = M_GET_STATUS;
        mouse(&m1,&m2,&m3,&m4);
        set_mouse_posn(&m3,&m4);
        return m2;
    }
    return 0;
}

static void set_mouse_posn(int *x, int *y)
/* Sets the internal mouse position variables */
/* Corrects for 320x200 low resolution modes */
{
    if (low_resolution) *x >>= 1; /* divide by two */
    mouse_grph_x = *x;
    mouse_grph_y = *y;
    mouse_text_x = *x/8 + 1;
    mouse_text_y = *y/8 + 1;
}

int mouse_trigger(int button_dir)
/* Looks for a key press, or a button release (if button_dir = 0)
or press (if button_dir = 1). Key presses have priority over
mouse buttons. The left mouse button has priority over the
right button. If a key has been pressed, it is then removed
from the keyboard buffer.
*/
{
    int k;
    if (bioskey(1)) {
        k = bioskey(0);
    }
    else {
        k = 0;
        if (button_dir) {
            if (button_press(0)) k = LEFT_MOUSE_PRESS;
            else if (button_press(1)) k = RIGHT_MOUSE_PRESS;
        }
        else {
            if (button_release(0)) k = LEFT_MOUSE_REL;

```

```

        else if (button_release(1)) k = RIGHT_MOUSE_REL;
    }
}
return k;
}

```

Listing 3.7 The Mouse Highlighting Example (IOEX7.C)

```

/*
   Text highlighting example with mouse (ioex7.c)
   Must link with: mouse.obj
*/
#include <bios.h>
#include <conio.h>
#include "mouse.h" /* mouse routines header file */
typedef struct texel_struct {
    unsigned char ch;
    unsigned char attr;
} texel;
void my_box(int xul, int yul, int xlr, int ylr, int btype);
void main() {
    int highlite = 0;
    int x = 1, y = 1;
    texel t;
    init_mouse(MOUSE_NEEDED, MOUSE_TEXT_MODE, MOUSE_TEXT_MODE);
    my_box(19,9,66,14,2);
    window(20,10,65,13);
    clrscr();
    mouse_off(1); /* mouse off temporarily */
    cprintf("Here is some text for you to highlight\r\n");
    cprintf("Click left button to toggle highlighting\r\n");
    cprintf("Drag w/left button down to highlight\r\n");
    cprintf("Click right button to exit");
    mouse_on(1); /* restore mouse cursor */
    do {
        if (button_press(1)) break; /* right button pressed means exit */
        if (button_press(0)) /* left button pressed means toggle highlighting */
            highlite = !highlite;
        if (button_state() == 1) { /* look for left button depressed */
            if (mouse_in_box(0,20,10,65,13)) { /* only highlite window text */
                /* only do if haven't been here before */
                if (x != mouse_text_x || y != mouse_text_y) {
                    x = mouse_text_x; y = mouse_text_y;
                    mouse_off(1); /* remember why ? */
                    if (highlite) { /* all coords here are absolute */
                        gettext(x,y,x,y,&t);
                        t.attr = 15; /* we'll use "bright" so we can see mouse */
                        puttext(x,y,x,y,&t);
                    }
                }
            }
        }
    } while (1);
}

```

```

        else { /* back to normal */
            gettext(x,y,x,y,&t);
            t.attr = 7;
            puttext(x,y,x,y,&t);
        }
        mouse_on(1); /* Only back on if previously on */
                    /* (In this program, this is always true) */
    }
}

) while (1);
mouse_reset(); /* reset mouse, turn it off */
window(1, 1, 80, 25);
clrscr();
)

void my_box(int xul, int yul, int xlr, int ylr, int btype)
/* Draws a box at the upper left (xul,yul) and lower right
   (xlr,ylr) coords with given attribute. If btype = 0, no box is drawn,
   if btype = 1, a single line box is drawn, if btype = 2, a double line
   box is drawn.
*/
{
    static int boxcar[2][6] = { /* graphics characters for a box */
        (218,196,191,179,192,217), /* single line box */
        (201,205,187,186,200,188) /* double line box */
    };
    int i, hzchar, vtchar;
    if (btype) {
        hzchar = boxcar[btype-1][1];
        vtchar = boxcar[btype-1][3];
        /* draw top and bottom sides */
        gotoxy(xul,yul);
        for (i=xul; i<=xlr; i++) putch(hzchar);
        gotoxy(xul,ylr);
        for (i=xul; i<=xlr; i++) putch(hzchar);
        /* draw vertical sides */
        for (i=yul; i<=ylr; i++) {
            gotoxy(xul,i);
            putch(vtchar);
            gotoxy(xlr,i);
            putch(vtchar);
        }
        /* draw corners */
        gotoxy(xul,yul); putch(boxcar[btype-1][0]); /* upper left */
        gotoxy(xlr,yul); putch(boxcar[btype-1][2]); /* upper right */
        gotoxy(xlr,ylr); putch(boxcar[btype-1][5]); /* lower right */
        gotoxy(xul,ylr); putch(boxcar[btype-1][4]); /* lower left */
    }
}

```

第四章 弹出窗口和错误报告

既然你已有了基本的工具，那么你就可以继续建立更新奇的工具了。当今流行的一种工具是对弹出窗口的支持。本章给出只使用 Turbo C 屏幕函数的这些工具的程序。结果是，这种窗口将不会象直接到视频 RAM 中去那样快，但这些程序证明为更易移植到将来的机器上。如果你有一个 8-MHz 或更高频率的机器，则窗口将快得多。清单 4.1 和 4.2 给出了弹出窗口软件包的头文件和源程序。它们含有下列函数和全程变量：

弹出窗口函数

| Function | Task |
|---------------------------|--|
| <code>init_win()</code> | Initializes the windowing system |
| <code>draw_win()</code> | Draws a new window |
| <code>view_win()</code> | Makes a window visible or erases it |
| <code>slct_win()</code> | Makes a window the active one (a macro) |
| <code>rmv_win()</code> | Erases a window (a macro that calls) |
| <code>clr_win()</code> | Clears a window |
| <code>draw_box()</code> | Draws a box with different borders and colors |
| <code>centerstr()</code> | Prints a string centered within given coordinates |
| <code>mprintf()</code> | A high-level formatted print routine |
| <code>prtfstr()</code> | A formatted print routine that supports line fill and highlighting |
| <code>swap_image()</code> | Toggles a window from visible to invisible |

| Variable | Usage |
|-------------------------|--|
| <code>base_win</code> | The window representing the whole screen. |
| <code>curr_win</code> | The current active window. |
| <code>defcolors</code> | The current color set in use. |
| <code>invcolors</code> | Reverse video monochrome colors. |
| <code>monocolors</code> | Normal monochrome colors. |
| <code>errcolors</code> | Error message colors. |
| <code>msgcolors</code> | Normal message colors. |
| <code>CTRWIN</code> | A code to use when centered windows are desired. |

下面是一个使用这些例行程序的简单例子。首先，它初始化该窗口系统，在屏幕中央弹出一个窗口，打印 Hello，等待一个键，随后删除该窗口并退出：

```
/*
  Popup window example (popex1.c)
  Must link with: popup.obj, mouse.obj
*/
#include <conio.h>
#include "popup.h" /* pop-up header file */
#include "mouse.h" /* need for key defs and mouse functions */
```

```

main() {
    windesc *w;    /* our window pointer */
    init_win();    /* always do first */
    init_mouse(MOUSE_OPTIONAL, MOUSE_TEXT_MODE, MOUSE_TEXT_MODE);
    w = draw_win(CIRWIN, CIRWIN, 20, 3,
        "My Window", popup, &defcolors);
    mprintf("Hello world");
    getch();
    rmv_win(w);    /* removes window */
    mouse_reset(); /* resets mouse */
}

```

这些函数中的每一个都将依次解释，但首先，我们将从定义一个窗口是什么开始。对我们的用途来说，窗口是独立于屏幕上的一个长方形区域；它独自卷动并有自己的文本光标。另外，有两类不同类型的窗口：暂时弹出的一类，和多少永久地弹出的一类。我们将称第一类为弹出窗口，第二类为画砖窗口。后一种的原因是画砖窗口不应重叠。相反，它们把屏幕划分为许多独立的“画砖”。

4.1 窗口结构

为了做弹出窗口，你必须有某种在弹出和随后删除窗口之后把屏幕恢复为其原状态的办法。窗口必须用存储它背后的图形来实现。怎样保持它的记录并且也保持光标的记录呢？原来，最好的办法是把窗口在当今日益流行的面向目标的风格中当作一个目标。

你可以用定义一个含有对给定窗口唯一的参数的结构。实际上，定义了好几个结构和类型，这在弹出软件包中给出：

```

typedef struct wincolors_struct {
    char border_type;
    unsigned char border_color, text_color,
        title_color, hilite_color;
} wincolors;
enum windowtype {popup, tile};

```

第一个结构含有关于在窗口中用什么颜色的信息。当你开始编商业质量的程序并想允许用户改变颜色时，这是一种变得更重要的便利。忠实地使用这一结构，颜色设置会变得容易得多。还在此处存储的是即将使用的边界类型，这在前章讨论过了。在软件包中预定义了5类不同的颜色设置：一类用于图形卡，一类用于普通的反相单色，一类用于错误信息，一类用于普通信息。你可改变这些以适应你的需要。

第二个定义给出了你的系统中所具有的窗口种类的枚举类型：弹出窗口和画砖窗口。稍后我们讨论弹出窗口堆栈时这种类型将起重要的作用。避开了这些预备类型，窗口结构可定义如下：

```

typedef struct winstruct {
    char *name;                /* window title */
    void *image;               /* ptr to image save area */
    struct winstruct *under,
        *over;                /* ptrs to window below and
                                above on pop-up stack */
    wincolors wc;              /* colors for the window */
}

```

```

char xul,yul,xlr,ylr,wd,ht; /* window coord's and sizes */
char xsave,ysave;          /* saved cursor posn */
enum windowtype wtype;     /* window type */
} windesc;

```

它含有你按支持多种类型窗口大小，形状和颜色的通用方式写程序所需的一切信息。第一个是一个指向在弹出窗口之前把窗口背后的图形存储起来的存储区域。该存储区动态定位。最后两个指针用于把窗口连接起来。

4.2 弹出窗口堆栈

许多时间在屏幕上将有多于一个的窗口，并且你想能够在它们之间来回移动。为此，你必须知道哪一个是当前可见的，而且你必须能够在它从屏幕上消失之后恢复原窗口。这样，你所需的就是某种堆栈。这里我们将使用的是带有两个指针 `under` 和 `over` 的双链表。这两个指针连接“下面的”窗口和“上面的”窗口。这个链表显示于图 4.1

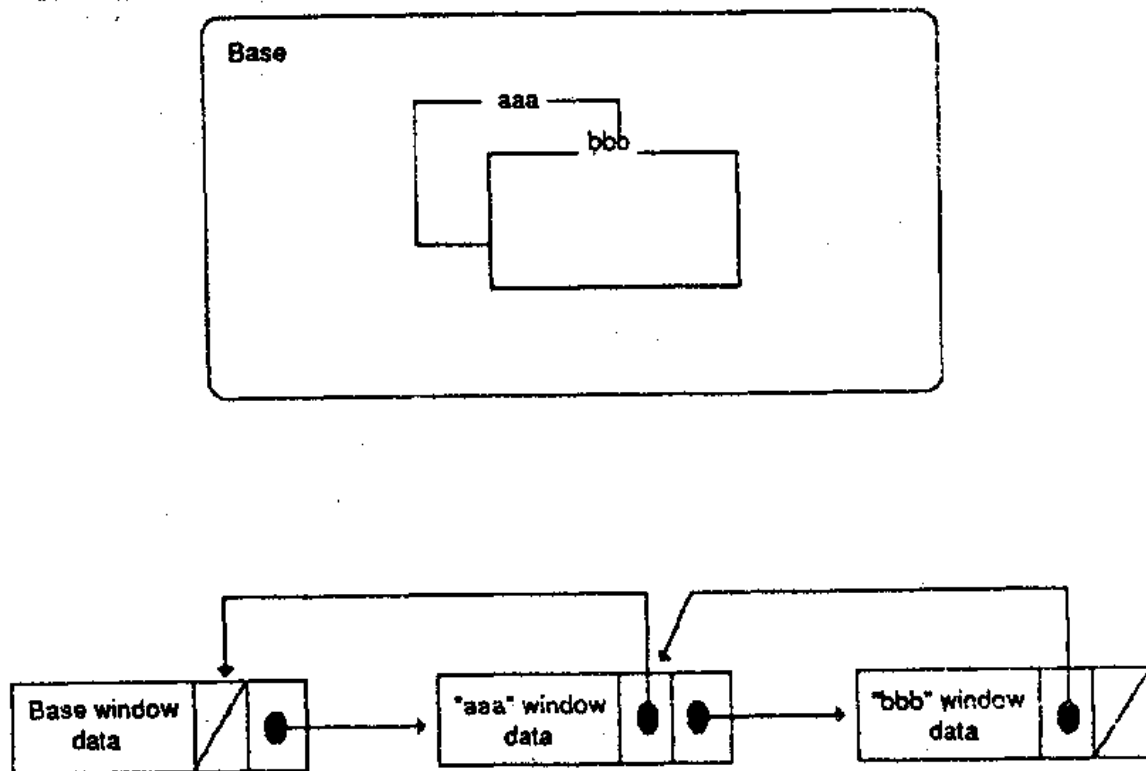


Figure 4.1 The doubly-linked pop-up window stack

“上面的”和“下面的”确切地意味着什么呢？由于每个窗口都保存它背后的图形（除某种类型的窗口外），你必须对窗口弹出的顺序保持记录，以便在删除它们时，它们背后的图形能正确恢复。如果你以错误的顺序恢复窗口，则你的屏幕将一团糟。“上面的”和“下面的”指的是这种顺序。“上面的”并不一定意味着一个窗口真的在另一个之上，仅仅意味着它在它下面的那个窗口之后被弹出（或有被交换的图形）。

在该堆栈的最底部是代表在任何窗口被弹出之前的屏幕的“窗口”。我们称该窗口为基窗口。在使用任何弹出窗口之前，你必须首先初始化此堆栈推向该窗口。这由函数

init-win() 执行, 它调用 Turbo C 的 gettextinfo() 函数来获得关于屏幕的当前信息。随后它使基窗口成为窗口堆栈的顶。指针 base-win 和 curr-win 是指向基窗口和当前或活动窗口的全程指针。

弹出窗口的例行程序是 draw-win()。它取代表窗口理想坐标的参数: 左上角, 及宽和高。你还要传给它标题字符串, 窗口类型, 和将使用的颜色类型。这个例行程序做了许多簿记, 如检查有效坐标和在新窗口结构中存储参数。它也存储背后的图形, 在窗口周围画一方框, 并把该窗口放在堆栈顶部。

在作为参数传递的窗口类型中有一个重要差别。如果你告诉 draw-win() 建一弹出窗口, 则窗口背后的图形被存储。如果它将是画砖窗口, 则背后的图形不存储。这一差异主要是为保存内存而做的。如果你的窗口表示屏幕上很大一部分, 且永不恢复, 则就没有为保存背后的图形而占据内存的理由。画砖窗口在完成窗口选择之后行为也不同。

4.3 操作窗口堆栈

函数 view-win() 用于选择和 / 或删除窗口。与之相关联的是宏 slct-win() 和 rmv-win()。它们的定义是:

```
#define slct_win(w) view_win(w,1)
#define rmv_win(w) view_win(w,0)
void view_win(windesc *w, int select)
/*
    w      - Window to view/remove
    select - 0: remove window
           1: select window
*/
```

函数通过移动即将选择或删除的窗口到堆栈顶部来工作。在选择一个新窗口时做这的原因是防止写向一个可能隐藏的窗口。窗口软件包还未精巧到足以知道什么时候一个窗口将在另一个之上。它的唯一知识是如前所释的“弹出”顺序。这样, 任何时候你选择一个窗口, 窗口堆栈都将被修改以使新窗口在顶部。

这是这样完成的: 先把在即将观看或删除的窗口之上的任何窗口都隐藏起来, 把该窗口本身隐藏起来, 把所有上面的窗口放回, 改变窗口堆栈的指针, 随后放回将观看的窗口。如果该窗口将被删除, 则不动它, 相反, 它从窗口堆栈中删除, 并且它下面的窗口(也即, 新栈顶)被选作当前窗口。在两种情况下, chg-win() 都随后被调用把当前窗口坐标改变为新窗口的。新选择的窗口的光标位置也被恢复。

图形交换的顺序非常重要。在选择的窗口之上的窗口通过用当前屏幕上的东西交换它们的存储图形而被删除。这由调用 swap-image() 来完成。这些图形被从栈顶交换到选择的窗口下面。把这些窗口放回时, 它们按相反的顺序被交换。这样做, 屏幕的完整性得到保存。

关于 view-win() 还有另两件重要的事要提及。一件是如果即将移开的窗口是画砖窗口, 则不交换任何图形。这是因为两个原因: 一是画砖窗口无存储的图形, 所以它不应被交换。第二, 如果你不交换它, 那么为什么还麻烦交换它上面的所有图形, 只是为把它们恢复原样呢? 第一个事实的结果是如果你知道另一窗口即使部分地在一画砖窗口之上, 那

么就应永不选择该画砖窗口。如果你写向画砖窗口,则很可能写到上面的弹出窗口上,随后就生发生混乱。第二个事实的结果是如果在你的系统中只有画砖窗口,则在它们之间来回忙碌是相当有效的,因为没有图形被交换。这在低速机器上是特别引人注目。总之,按其名字暗示的方式使用各类窗口。画砖窗口是半永久性的,弹出窗口应为暂时的。

4.4 隐藏和显现窗口

函数 `swap-image()` 用 Turbo C 函数 `gettext()` 和 `puttext()` 交替存储和恢复窗口下面的图形。它被写作图形开关。如果你连续调用它几次,则它将交替地隐藏和显现窗口。这样,如果你想暂时隐藏一个窗口并随后重新画它,而不必重新构造它内部的所有文本,则这是很方便的。但是,应告诫的是你应只交换堆栈上的顶部的窗口,这在前一段已解释过。

为做交换工作, `swap-image()` 不仅使用存储窗口的图形缓冲区,而且还分配额外的同样大小的内存用作暂时缓冲区。这种低效率的方法除非人们使用了一种更直接的更新屏幕的方法仍是必要的。作为一种练习,看看你能否用上一章解释过的直接视频技术来写一个只用一个单文本单元缓冲区交换图形的例程序。如果你真的写出了,则记住在更新屏幕的同时把鼠标器光标隐藏起来。函数 `swap-image()` 就这样做。

4.5 窗口 I/O

包含在弹出窗口软件包中有五个写窗口的例程序: `mprintf()`, `prtfstl()`, `centerstr()`, `clr-win()` 和 `draw-box()`。它们执行比 Turbo C `cprintf()`, `putch()`, `cputs()` 和 `clrscr()` 函数更高级的操作。跟它们的 Turbo C 对等函数一样,弹出窗口打印例程序总是写当前窗口,但它们在屏幕更新期间把鼠标器隐藏起来,这一点 Turbo C 函数做不到。这就是为什么提供 `mprintf()`,它做与 `cpvintf()` 同样的事(实际上它调用 `cpvintf()`)。如果你在用鼠标器,则你应总是用 `mprintf()` 代替 `cpvintf()`。`mprintf()` 和 `prtfstv()` 的程序表明了使用 C 的可变数目参数调用序列的例子,它们在第二章解释过。尽管未提供它们,但你可以为 `cput()` 和 `putlh()`(即保持鼠标器光标的那些)写与 `mprintf()` 类似的例程序。

`prtfstl()` 例程序可用不同的方法使用;其中之一是突出显示文本。如果你传给它一个非零属性,则它将把该属性用做新颜色来写出文本。这在做突出显示的菜单杠条时是很有用的。例如,如果 `attr=0`,则它用当前屏幕上的颜色写文本。下面对 `prtfstl()` 的调用在屏幕列 10 行 5 处用蓝色打印出“Hello world”的全部 11 个字符。

```
prtfstl(10,5,"Hello world",1,11);
```

如果你给它一个大于该串的宽度参数,则串长被使用,否则被窗口删节。

```
prtfstl(10.5,"Hello world",1,80);
```

任何时候你不想麻烦计算合适的长度时,你都可用 80 做为宽度, `prtfstl()` 为你做这项工作。如果打印引起文本超出窗口边界,则不会发生滚动;而是文本被删节。如:

```
w = draw_win(1,1,20,6,"my win",popup,&defcolors);
prtfstl(15,1,"this text will be truncated",0,80);
```

将只在窗口列 15 行 1 处打印“this t”。由于属性被做为 0 传递,所以颜色不变。

不直接到 RAM 而做删节需要一些技巧。因为 Turbo C 打印函数总是移动光标，如果你在窗口中最后一列打印一个字符，则光标被移到下一行并且有必要时还卷动窗口。这意味着你不能有一行高的窗口，因为如果你写该行最后一个字符，则窗口卷动且你的文本消失。因为 Turbo C 未提供任何关掉卷动的方法，所以唯一的选择是用 `gettext()` 和 `puttext()` 从屏幕取文本图形，对它做某些改变，随后把它放回。这两个例行程序不影响光标，但它们着实使打印速度降低。为了最终的速度，你可能会考虑直接到 RAM。

如果你把宽度作为负参数传递，且如果串长为 1（即单字符），则这就告诉了 `prtfstr()` 做一个行填充。下面将打印 25 个 C（除非被删节了）：`prtfstr(1,5,"c",0,-25);`

你也可通过传递作为一个格式串的串和在 `wd` 参数后传递即将格式化的参量来进行格式化。

```
prtfstr(1,5,"The mouse is at %d %d",0,80,mouse_text_x,mouse_text_y);
```

在打印突出显示的菜单杠条时可用一种好的格式化技巧。下面的语句将突出显示一个 25 字符宽，左调整其中串的杠条。它使用 `printf()` 左调整码“%-*.*\$”并把格式化域的宽度和精度作为参数传递。

```
prtfstr(1,5,"%-*.*s",112,25,25,25,"Menu choice #1");
```

注意第一个 25 是 `prtfstr()` 函数本身的宽度参数。另两个是用于格式串的。

例行程序 `centerstr()` 将自动计算坐标，以便你能使某些文本在水平和竖直两个方向上处于一个给定长方形区域的中心。它调用 `prtfstr` 做实际的打印工作。注意没有给出用此函数进行格式化的任何规定。

例行程序 `clr-win()` 用于清除当前窗口。除它保持鼠标器光标外，它与 `clrscr()` 一样。最后，`draw-box()` 是上一章给出的 `my-box()` 例行程序的扩展形式。它允许你传递边界颜色，并且它也保持鼠标器光标。另一个是 `my-box()` 的重要改变是它用 `gettext()` 和 `puttext()` 方法画右下角。这是为了防止屏幕卷动，这个角应该在屏幕最低部。

4.6 一个简单菜单程序

既然你已有了所有这些工具，那么你就想看看使用它们的某些例子了。第一个例子是清单 4.3 中给出的简单菜单程序。这个程序弹出一个菜单并允许你用上下箭头移动。用 Return 键做选择。选择在另一窗口中印出，这过程进行到按下 Esc 键为止。注意这些窗口是画砖的。假如它们是弹出窗口的话，则每次选择一个窗口时都会由于窗口被交换而看到烦人的闪烁。窗口软件包并未聪明到足以知道窗口不重叠，因而不需被交换。做为一个练习，你可试使之更灵活一些。做为一个“起点”，这里我们把它们建为画砖的。

在你明白了所给的程序后（它使用了许多早先讨论过的功能特性），严格试验一下清单 4.4 中的程序。除了它被扩展为用鼠标器操作外，它与上一个是一菜单程序。按下菜单选择项左边的按钮，该项就被选择。如果按右按钮，则程序退出。图 4.2 显示了该简单菜单程序的样例屏幕。

这个程序向你表明怎样写有鼠标器时使用它，无时忽视它的程序。我们已调用 `mouse-init()` 并告诉它鼠标器是可选择的。如果没有鼠标器装配它也不会失败，它仅仅忽略请求。另外，`mouse-trigger()` 函数是为在鼠标器未被装配和初始化时忽略之而写

的。它仍将返回关键码。

4.7 移动窗口程序

下一程序，清单 4.5，显示了怎样在屏幕上移动窗口。这个程序也表明了当窗口被选择后怎样被挪动到栈顶。它这样做：如果你按下在其标题中给出的窗口号，则该窗口被选择。如果你使用箭头键，则你能使窗口移动。任何别的键都回映在当前窗口中。按 Esc 退出程序。图 4.3 显示了移动窗口程序的一个样例屏幕。

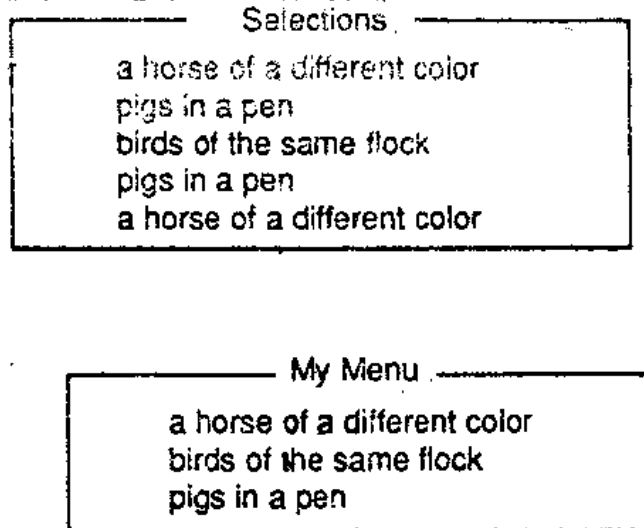


Figure 4.2 Sample screen of the simple menu program

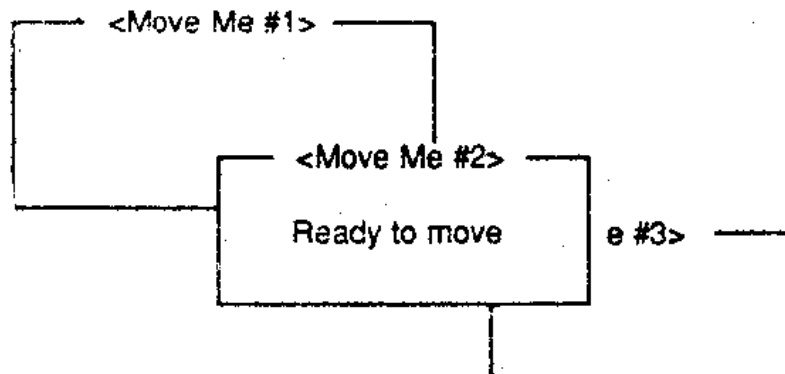


Figure 4.3 Sample output from the moving windows program

该程序也被设计为如果鼠标器存在的话使用之。当鼠标在窗口边界上任何地方上时按下鼠标器左按钮则会导致该窗口被选中并被挪到栈顶。保持按下鼠标器左按钮，你可用该鼠标器往回处移动窗口。按右鼠标器按钮则退出程序。

4.8 弹出错误和信息包

下一个例子比前两个稍微实用些。实际上，它是如此有用以致它被包括进清单 4.6 给出的弹出窗口软件包，作为其一部分。它是一个处理错误信息的系统。

这个系统按下面的方式工作：涉及到两个窗口，一个处理错误信息，一个处理普通信息。窗口可在信息出现时被匆忙弹出，或你可把它们弹出一次并在程序持续期间保留它们。例行程序是供正确规定信息到这些窗口的顺序之用的。这些例行程序使用下列全程定义的窗口指针：

```
int errx = CTRWIN;      /* Default to center of the screen */
int erry = CTRWIN;
windesc *errw = NULL;   /* Default to pop-up window */
int msgx = CTRWIN;      /* same for messages */
int msgy = CTRWIN;
windesc *msgw = NULL;
```

窗口指针由系统设置为 NULL。如果一个候选窗口在其相应信息例行程序被调用时是 NULL，则有一个窗口被弹出来显示该信息。错误窗口按 (errx, erry) 给出的坐标处弹出，而信息窗口在 (msgx, msgy) 处。这些坐标由系统设置为屏幕中心。如果窗口指针不是 NULL，则假定它指向一个已在屏幕上的窗口，且该窗口被暂时选来显示信息。信息显示之后，在此信息之前是当前窗口的那个窗口被重新选择。

下列函数被包含在软件包中：

```
void numnewlines(char *s, int *n, int *w);
void popmsg(int x, int y, char *msg, char *title, char soundout, wincolors *wc);
void reperr(int level, char *msg);
void repmsg(char *msg);
void sayerr(int ferr, int errflag, int lno, char *pname, char *fmt, ...);
void beep(void);
unsigned int getkey(void);
```

后两个先解释。beep() 函数仅调用 Turbo C 的 sound(), delay() 和 nosound() 函数来产生一个用户蜂鸣例行程序。你可改变所用参数来适应你的口味。

getkey() 函数用于捕捉 CTRL-C 键。它调用 bioskey(0) 从缓冲区获取下一个键。如果那个键是 CTRL-C，则弹出一个放弃窗口看用户是否想放弃。这个例行程序可用来代替 bioskey(0)，如果你想提供一种放弃该程序的方法的话。这在调试中尤为有用。注意 Turbo C 的 ctribvk() 函数对我们没多大用处。它只在有 DOS 调用时检查 CTRL-C。函数 bioskey(), cprintf(), putchar(), 等等不调用 Dos，所以 ctrl-break 处理程序没多大用处。由于在做窗口时有一些常用的函数，所以需要某些别的捕捉 CTRL-C 的方法。getkey() 函数提供了一种方法。

sayerr() 函数是到错误报告系统的高级接口。其参数如下:

| Parameter | Meaning |
|------------|--|
| ferr | 1 means print out the last DOS error 0 means don't print it |
| errflag | 0 means treat as message, 1 means warning, 2 means an error |
| lno, pname | If pname is not a null string, then it is assumed to be a file name, and lno the line number in that file. These together will be included in the message. |
| fmt, ... | A printf-style format string and its optional arguments |

这个函数基本上取你的格式化的信息, 并且根据参数, 可能取最后的 DOS 错误信息, 和在其中发生错误的行号和文件名。后面的这些参数对调试用途很有用。你能让它在发生错误的 C 源文件当前行处自动装入。这通过使用 C 预定义宏-`LINE`-和-`FILE`-来实现。当-`LINE`-在预处理过程中被扩展时, 它被正被编译的当前行号替换。-`FILE`-由被编译过的文件的全路径名替换。为使用方便, 还为 sayerr() 定义了下列附加的宏:

```
/* Use these if you wish to include source file and line number */
#define SWRNF 0,1, __LINE__, __FILE__
#define SERRF 0,2, __LINE__, __FILE__
#define SMSGF 0,0, __LINE__, __FILE__
#define FWRNF 1,1, __LINE__, __FILE__
#define FERRF 1,2, __LINE__, __FILE__
#define FMSGF 1,0, __LINE__, __FILE__
/* Otherwise, use these */
#define SWRN 0,1,0, ""
#define SERR 0,2,0, ""
#define MSG 0,0,0, ""
#define FWRN 1,1,0, ""
#define FERR 1,2,0, ""
#define FMSG 1,0,0, ""
```

这些宏组成了你的(sayerr()) 函数调用的前四个参数。为使它们易于记忆, 宏的名字如下构成: 当你不想包括进最后的 DOS 错误信息时用“S”前缀; 想包括进时用“F”前缀。(“F”代表“文件”(“FILE”)。许多 DOS 错误信息是与文件相关的)。接着是信息类型: “WRN”用于警告, “ERR”用于错误, “MSG”用于普通信息。连在尾部的“F”后缀指明你想让当前行号和文件包含在信息中。做为一个例子, 下面的程序段随程序中错误发生的位置打印出 DOS “file not found”信息:

```
/* ... */
handle = open("nosuchfile", O_RDONLY)
if (handle == -1) sayerr(FERRF, "%s\r\n", "nosuchfile");
/* ... */
```

The code for sayerr() is given as follows:

```
void sayerr(int ferr, int errflag, int lno,
            char *pname, char *fmt, ...)
{
    va_list arg_ptr;
    char t[255]; /* temp character buffer */
    int j;       /* temp character count */
```

```

if (*pname) {
    j = sprintf(t, "On line %d in pgm %s\r\n", lno, pname);
}
else j = 0;
if (ferr == 1) {
    /* add last dos error */
    j += sprintf(t+j, "%s: ", strerror(errno));
}
va_start(arg_ptr, fmt); /* point to optional arguments */
vsprintf(t+j, fmt, arg_ptr); /* add rest of formatted string */
va_end(arg_ptr);
switch(errflag) {
    case 1:
        reperr(0, t); /* just a warning */
        break;
    case 2:
        reperr(1, t); /* an error */
        break;
    default:
        repmsg(t); /* or a plain old message */
}
}
}

```

这个程序用许多 C 的功能特性来建立错误信息：参量的可变数目，通过 `sprintf()` 的内部格式化，和通过 `strerror()` 的对最后 Dos 错误的信息的恢复。一旦错误信息被创建，它就被传给适当的信息报告例行程序，或 `reperr()`，或 `repmsg()`。这些例行程序检查是否定义了适当的信息窗口。如果定义了，则信息被安排到该窗口；否则调另一例行程序弹出信息 `popmsg`。

函数 `popmsg()` 取一任意字符串，并自动定出一个能打印出含有该信息的窗口的大小。它还调用另一个函数 `numnewLine()` 来做计算串中出现的新行字符个数，并随之确定该信息中所有行中的最大长度。弹出一个窗口后，它在消除该窗口之前等待一个回车或 Esc 键。一个参数被传递用来指明你是否要蜂鸣器发声。任何时候你想显示一个多行信息而不必计算出所需窗口大小，`popmsg()` 例行程序都将是有用的。然而，一个限制是格式不能超过 255 个字符，所以这只对短信息有益。

清单 4.7 中所示的程序是错误信息报告系统如何工作的一个样例。它画一个画砖“主”窗口，然后显示一个弹出错误窗口，以一个 DOS 错误信息结束。它随后画一个画砖错误窗口并安排一些错误信息，此过程中严重级别由错误变到警告。最后，它弹出一个普通信息窗口。图 4.4 和 4.5 显示了该程序的输出样例。

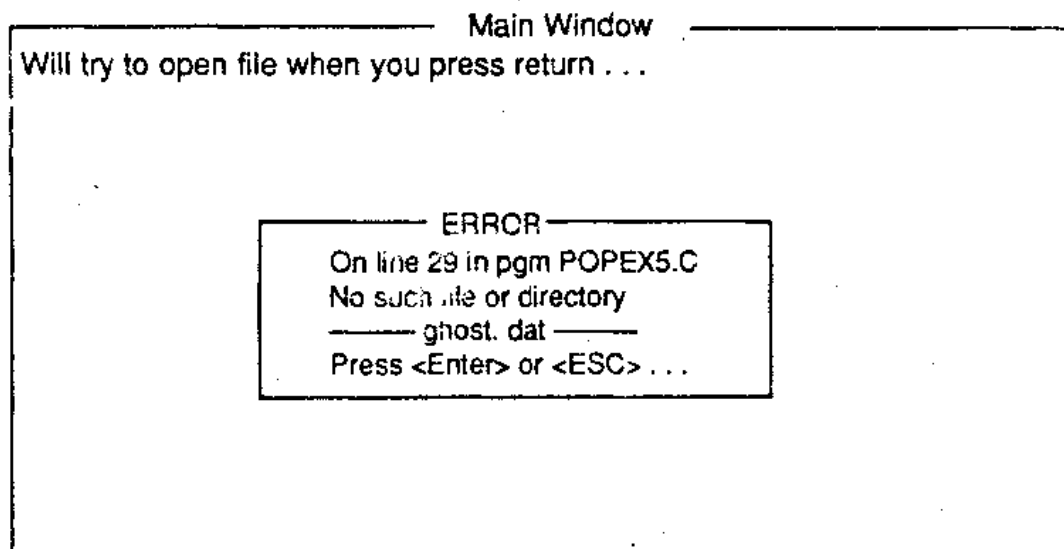


Figure 4.4 Sample pop-up error window

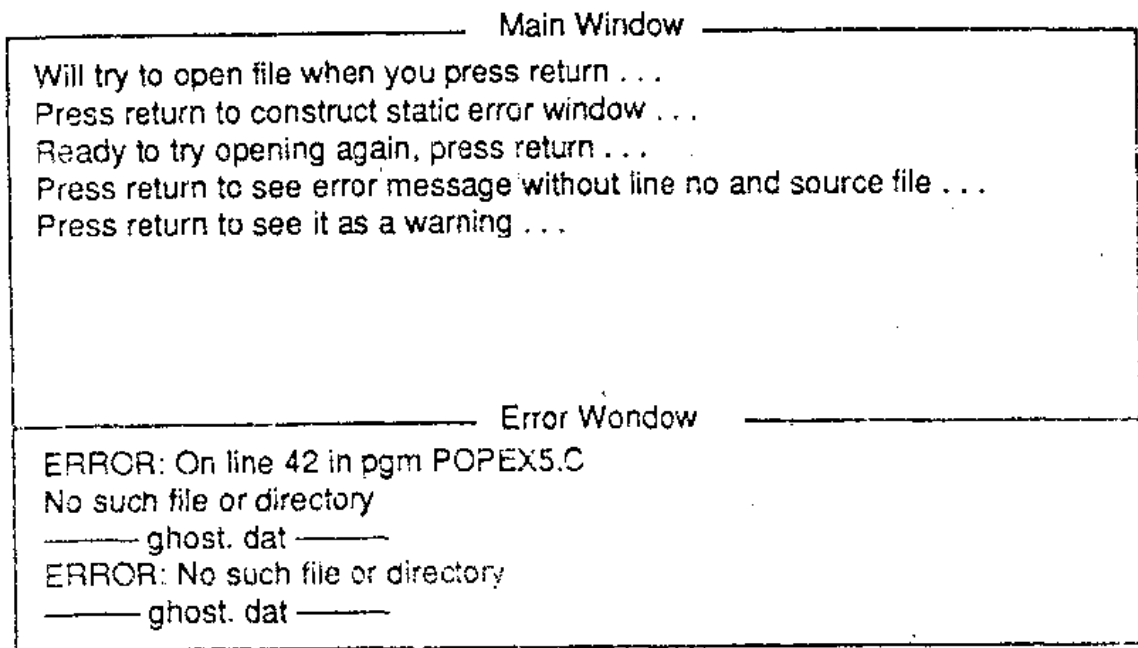


Figure 4.5 Sample static error window

Listing 4.1 Source Code for Pop-up Window Header File (POPUP.H)

```

/* Text mode popup window package (popup.h) */
/* Center window code */
#define CTRWIN 999
/* video ram text-cell structure */
typedef struct texel_struct {

```

```

    unsigned char ch;
    unsigned char attr;
} texel;
/* A structure to hold the box type, and a set of window colors */
typedef struct wincolors_struct {
    char border_type;
    unsigned char border_color, text_color, title_color, hilite_color;
} wincolors;
/* popupwindows save the image underneath, tiled windows don't */
enum windowtype {popup,tile};
/* A structure to hold information for each window */
typedef struct winstruct {
    char *name; /* window title */
    void *image; /* ptr to image save area */
    struct winstruct *under,*over; /* ptrs to window below and above
                                   on popup stack */
    wincolors wc; /* colors for the window */
    char xul,yul,xlr,ylr,wd,ht; /* window coord's and sizes */
    char xsave,ysave; /* saved cursor posn */
    enum windowtype wtype; /* window type */
} windesc;
extern windesc *base_win; /* can be used to access whole screen */
extern windesc *curr_win; /* current window in use */
extern wincolors defcolors; /* predefined color sets */
extern wincolors invcolors;
extern wincolors monocolors;
extern wincolors errcolors;
extern wincolors msgcolors;
/* macros for easy use in removing and selecting window */
#define rmw_win(w) view_win(w,0)
#define slct_win(w) view_win(w,1)
/* Now the prototypes for the popup functions */
extern void init_win(void);
extern windesc *draw_win(int x, int y, int wd, int ht, char *title,
                        enum windowtype wt, wincolors *wc);
extern void view_win(windesc *this, int move_to_top);
extern void clr_win(void);
extern void draw_box(int xul,int yul,int xlr,int ylr,int btype,int attr);
extern void centerstr(int xul, int yul, int xlr, int ylr,
                    char *s, unsigned char a);
extern void mprintf(char *fmt,...);
extern void prtfsr(int x, int y, char *fmt, unsigned char attr, int wd,...);
extern void swap_image(windesc *w);
/* Includes for sayerr.c */
/* the first set of macros include line number and name of source file */
#define SWRNF 0,1,__LINE__,__FILE__
#define SERRF 0,2,__LINE__,__FILE__
#define SMSGF 0,0,__LINE__,__FILE__
#define FWRNF 1,1,__LINE__,__FILE__
#define FERRF 1,2,__LINE__,__FILE__
#define FMSGF 1,0,__LINE__,__FILE__
/* The second set do not */
#define SWRN 0,1,0,""
#define SERR 0,2,0,""
#define SMSG 0,0,0,""
#define FWRN 1,1,0,""
#define FERR 1,2,0,""

```

```

#define FMSG 1,0,0,""
extern int errx;      /* Default error coordinates */
extern int erry;
extern windesc *errw; /* error window pointer */
extern int msgx;      /* same for messages */
extern int msgy;
extern windesc *msgw;
extern void numnewlines(char *s, int *n, int *w);
extern void popmsg(int x, int y, char *msg, char *title,
                  char soundout, wincolors *wc);
extern void reperr(int level, char *msg);
extern void repmsg(char *msg);
extern void sayerr(int ferr, int errflag, int lno,
                  char *pname, char *fmt,...);
extern void beep(void);
extern unsigned int getkey(void);

```

Listing 4.2 Pop-up Window Source Code (POPUP.C)

```

/*****
 * Popup window toolkit for Turbo C (popup.c)
 *****/
#include <stddef.h>
#include <stdarg.h>
#include <stdio.h>
#include <stdlib.h>
#include <alloc.h>
#include <conio.h>
#include <string.h>
#include <process.h>
#include "popup.h"
#include "mouse.h"
#define MAX(a,b) ((a) > (b) ? (a) : (b))
#define MIN(a,b) ((a) < (b) ? (a) : (b))
/*
 * Global data definitions
 */
windesc *base_win = NULL; /* The "Base Window" pointer */
windesc *curr_win = NULL; /* The current window pointer */
/* Color sets for color monitors */
/* {border_type, border_color, textcolor, title_color, hilite_color} */
windesc defcolors = {1, 23, 30, 27, 94 };
windesc invcolors = {1, 112, 112, 112, 15 };
windesc monocolors = {1, 7, 15, 15, 112 };
windesc errcolors = {1, 79, 79, 79, 4 };
windesc msgcolors = {1, 47, 47, 46, 112 };
/* variables internal to popup.c */
static windesc *top_win; /* window stack pointer */
static void chg_win(windesc *this);
static windesc *make_window_node(void);
static windesc *push_window_node(void);
static void dispose_window_node(windesc *w);
} void init_win(void)
/*
init_win initializes the internal variables for the popup windows
package. This function creates a base window "base_win" which

```


represents the entire screen.

This function MUST be called before any popup routines are used.

```
*/
{
    struct text_info ti;                /* turbo-c predefined structure */

    base_win = make_window_node();      /* allocate a window node */
    gettextinfo(&ti);                  /* get base window coords */
    base_win->xul = ti.winleft - 1;      /* but our coords include border !!! */
    base_win->xlr = ti.winright + 1;
    base_win->yul = ti.wintop - 1;
    base_win->ylr = ti.winbottom + 1;
    base_win->wd = ti.screenwidth + 2;
    base_win->ht = ti.screenheight + 2;
    base_win->xsave = ti.curx;
    base_win->ysave = ti.cury;
    base_win->wc = monocolors;          /* default to monochrome colors */
    base_win->wc.text_color = ti.attribute; /* but use current window color */
    base_win->name = "base";             /* window has no title */
    base_win->wtype = tile;              /* no saved image underneath */
    base_win->wc.border_type = 0;        /* has no border either */
    top_win = base_win;                 /* set up window stack */
    curr_win = base_win;
}
```

```
2 windesc *draw_win(int x, int y, int wd, int ht, char *title,
                    enum windowtype wt, wincolors *wc)
```

```
/*
    draw_win creates a new window at a designated screen location.
    All window coordinates include the border!! Note that the
    Turbo-C window function does not know about the border, and that
    its coordinates are actually "inside" the border.
    Besides the usual coordinates for (x,y), you can use the following
    codes:
        x = CTRWIN    (Center window in x direction)
        y = CTRWIN    (Center window in y direction)
```

```
*/
{
    windesc *w;
    int xsave, ysave;
    mouse_off(1);                      /* hide mouse cursor during screen updates */
    w = push_window_node();             /* create and link up a window node */
    /* Check for valid window size */
    wd = MAX(wd, 3);
    wd = MIN(wd, 80);
    ht = MAX(ht, 3);
    ht = MIN(ht, 25);
    /* Check for centering coordinates and current coordinates.
       Reminder: these are absolute coordinates !!! */
    if (x == CTRWIN) x = (80-wd) / 2;
    if (y == CTRWIN) y = (25-ht) / 2;
    /*
       Check for valid coordinates. Coordinates (0,0) are valid only for
       borderless windows, (the actual window will start at (1,1)).
    */
    if (wc->border_type) {
        x = MAX(x, 1);
        y = MAX(y, 1);
    }
```

```

    }
    else {
        x = MAX(x,0);
        y = MAX(y,0);
    }
    if ((x+wd) > 80) x = 80-wd+1;
    if ((y+ht) > 25) y = 25-ht+1;
    /* Store the window parameters */
    w->wd = wd;          w->ht = ht;
    w->xul = x;          w->yul = y;
    w->xlr = x + w->wd - 1; w->yir = w->yul + w->ht - 1;
    w->wc = *wc; /* set up our set of colors */
    w->xsave = 1;        w->ysave = 1;
    w->wtype = wt;       w->name = strdup(title);
    /* allocate and save image underneath if a popup window */
    if (wt == popup) {
        w->image = calloc(wd*ht,2);
        swap_image(w);
    }
    /* Ready to draw box and title. To do this we must be in base
       or "absolute" coords. But we must be careful to save the
       base cursor coords cause draw box and centerstr will change
       them on us.
    */
    if (curr_win == base_win) {
        xsave = wherex();
        ysave = wherey();
    }
    else {
        xsave = base_win->xsave;
        ysave = base_win->ysave;
    }
    chg_win(base_win);
    draw_box(w->xul,w->yul,w->xlr,w->yir,w->wc.border_type,w->wc.border_color);
    centerstr(w->xul,w->yul,w->xlr,w->yul,w->name,w->wc.title_color);
    gotoxy(xsave,ysave); /* restore base window cursor coords */
    chg_win(w);          /* New select new window */
    clr_win();           /* and clear it */
    mouse_on(1);         /* restore mouse state */
    return w;            /* return new window pointer */
}

void view_win(windesc *this, int select)
/* If select = 1, then view_win moves "this" window to the
   top of the stack and makes it the active window.
   If select = 0, then the window is removed from the stack and erased.
   Note that no moves take place if already at the top.
   If this is merely a tiled window, then it is just selected:
*/
{
    windesc *p;
    if (select && this == top_win) return;
    mouse_off(1); /* make sure mouse hidden */
    /* if this is a popup window, move its image to the top */
    if (this->wtype == popup) {
        p = top_win;
        while(p != this) { /* hide all window above */

```

```

        swap_image(p);
        p = p->under;
    }
    swap_image(this); /* and then this window */
    p = this; /* then put rest of windows back */
    while(p != top_win) {
        p = p->over;
        swap_image(p);
    }
}
/* link up window underneath this one, with the window above it */
if (this == top_win) { /* if this == top_win here, then it is also */
    this->under->over = NULL; /* true that we're removing it for good */
    top_win = this->under;
}
else {
    this->under->over = this->over;
    this->over->under = this->under;
}
if (select) {
    top_win->over = this; /* move window to the top */
    this->under = top_win;
    top_win = this;
    swap_image(this); /* put back it's image. Does nothing if tiled */
    chg_win(this); /* change window */
}
else {
    chg_win(top_win); /* might as well select top window */
    dispose_window_node(this); /* but do before free old window */
}
mouse_on(1); /* restore mouse state */
}

static void chg_win(windesc *this)
/* Internal routine to select a window
{
    curr_win->vsave = wherex();
    curr_win->ysave = wherey();
    window(this->xul+1, this->yul+1, this->xlr-1, this->ylyr-1);
    textattr(this->wc.text_color); /* restore window color and cursor */
    gotoxy(this->xsave, this->ysave);
    curr_win = this; /* selected window is active */
}

void swap_image(windesc *w)
/*
This routines swaps the image buffer of a window with what is
on the screen. If the window is not a popup window, then
nothing happens.
*/
{
    int xstart, ystart, xfin, yfin;
    char *temp_image;
    unsigned int nbytes;
    if (w->wtype == popup) {
        xstart = w->xul; ystart = w->yul;
        xfin = w->xlr; yfin = w->ylyr;
        if (!w->wc.border_type) {
            /* don't swap border area if no border */

```

```

        xstart++; ystart++;
        xfin--; yfin--;
    }
    nbytes = (xfin-xstart+1)*(yfin-ystart+1)*2;
    mouse_off(1); /* hide mouse cursor during screen update */
    temp_image = malloc(nbytes);
    gettext(xstart, ystart, xfin, yfin, temp_image);
    puttext(xstart, ystart, xfin, yfin, (void *)w->image);
    memcpy(w->image, temp_image, nbytes);
    free(temp_image);
    mouse_on(1); /* restore mouse cursor */
}
}
}

6 void clr_win(void)
/*
 * High level clear window. Supports the mouse cursor, and uses
 * the text_color of the current window when clearing it.
 */
{
    mouse_off(1);
    textattr(curr_win->wc.text_color);
    clrscr();
    mouse_on(1);
}

6 void mprintf(char *fmt,...)
/*
 * mprintf is a high level printf-like function that calls cprintf
 * (so it knows about the current window), but also takes care of
 * the mouse.
 * NOTE: The formatting must not exceed 255 characters !
 */
{
    va_list arg_ptr;
    char t[255];
    va_start(arg_ptr,wd);
    vsprintf(t,fmt,arg_ptr); /* internal format function */
    mouse_off(1); /* hide mouse cursor during screen update */
    cprintf("%s", t); /* print the string with scrolling supported */
    mouse_on(1); /* restore mouse cursor */
    va_end(arg_ptr);
}

7 void prtfsr(int x, int y, char *fmt, unsigned char attr, int wd,...)
/*
 * prtfsr prints a formatted string in the current window. prtfsr is
 * similar gprintf except that it doesn't support wrap-around, in fact,
 * it goes to great lengths to get around it. If the string is too long
 * to fit in the window, it will be truncated.
 * If attr != 0, the string will be printed with that color, otherwise
 * the color will be left alone.
 * If wd > 0 and the length of the formatted string >= 1, then prtfsr
 * will print the one-character string abs(wd) times. This is useful
 * for filling a line with a character.
 * NOTE: The formatting must not exceed 255 characters !
 */
{
    va_list arg_ptr;

```

```

char t[255];
int len, i, n, xa, ya, fillflag;
static texel line[80]; /* text image buffer */
va_start(arg_ptr, wd);
vsprintf(t, fmt, arg_ptr); /* internal format function */
va_end(arg_ptr);
n = abs(wd); /* compute and bounds check desired width */
n = MIN(n, curr_win->wd-x-1);
xa = curr_win->xul + x; /* get absolute coordinates */
ya = curr_win->yul + y;
len = MIN(strlen(t), n); /* keep string inside the window */
t[len] = 0; /* by truncating if necessary */
if (len) { /* only do non-null strings */
    if (wd < 0 && (strlen(t) == 1)) {
        fillflag = 1;
        len = n;
    }
    else {
        fillflag = 0;
    }
    mouse_off(1); /* remember to hide mouse */
    gettext(xa, ya, xa+len-1, ya, line); /* extract text image */
    for (i=0; i<len; i++) { /* change it */
        if (fillflag) line[i].ch = *t;
        else line[i].ch = t[i];
        if (attr) line[i].attr = attr;
    }
    puttext(xa, ya, xa+len-1, ya, line); /* put it back */
    mouse_on(1); /* restore mouse on */
    if (x+len == curr_win->wd-1) x--; /* Keep cursor in window */
    gotoxy(x+len, y); /* then move it to end of string */
}
else { /* for null strings, just move cursor */
    gotoxy(x, y);
}
}

void centerstr(int xul, int yul, int xlr, int ylr,
               char *s, unsigned char a)
/*
centerstr prints a string centered between the relative coord's
(xul,yul) and (xlr,ylr), with attribute a.
*/
{
    int xs, ys, wd;
    if (*s != 0) {
        mouse_off(1);
        wd = xlr-xul+1;
        if ((xs = (wd-strlen(s)) / 2 + xul) < xul) xs = xul;
        if ((ys = (ylr-yul+1) / 2 + yul) < yul) ys = yul;
        prtfsr(xs, ys, s, a, wd);
        mouse_on(1);
    }
}

void draw_box(int xul, int yul, int xlr, int ylr, int btype, int attr)
/* Draws a box at the upper left (xul,yul) and lower right
(xlr,ylr) coords with given attribute. If btype = 0, no box is drawn,
if btype = 1, a single line box is drawn, if btype = 2, a double line

```

```

box is drawn.
*/
{
    static int boxcar[2][6] = {
        /* graphics characters for boxes */
        /* { topleft, hz, topright, vt, bottomleft, bottomright } */
        {218,196,191,179,192,217}, /* single line box */
        {201,205,187,196,200,188} /* double line box */
    };

    int i, hzchar, vtchar, oldattr;
    texel t;
    struct text_info ti;
    if (btype) {
        mouse_off(1);
        gettextinfo(&ti); /* save old text attribute */
        oldattr = ti.attribute;
        textattr(attr); /* set new one */
        hzchar = boxcar[btype-1][1];
        vtchar = boxcar[btype-1][3];
        /* draw top and bottom sides */
        gotoxy(xul+1,yul);
        for (i=xul+1; i<xlr; i++) putch(hzchar);
        gotoxy(xul+1,ylr);
        for (i=xul+1; i<xlr; i++) putch(hzchar);
        /* draw vertical sides */
        for (i=yul+1; i<ylr; i++) {
            gotoxy(xul,i);
            putch(vtchar);
            gotoxy(xlr,i);
            putch(vtchar);
        }
        /* draw corners */
        gotoxy(xul,yul); putch(boxcar[btype-1][0]); /* upper left */
        gotoxy(xlr,yul); putch(boxcar[btype-1][2]); /* upper right */
        gotoxy(xul,ylr); putch(boxcar[btype-1][4]); /* lower left */
        /* can't write lower right corner via putch, due to possible
           scroll problems, so must do it a roundabout way */
        gettext(xlr,ylr,xlr,ylr,&t);
        t.ch = boxcar[btype-1][5]; t.attr = attr;
        puttext(xlr,ylr,xlr,ylr,&t);
        textattr(oldattr); /* restore old attribute */
        mouse_on(1);
    }
}

/* static windesc *make_window_node(void)
{
    /*
    Make_window_node allocates room for a new window structure,
    and initializes it's links to NULL.
    */
    windesc *q;
    q = (windesc *)malloc(sizeof(windesc));
    q->image = NULL; q->under = NULL; q->over = NULL;
    Return q;
}

/* static windesc *push_window_node(void)
{
    /*

```

```

    Push_window_node "pushes" the window w onto the window stack.
*/
windesc *q;
q = make_window_node();      /* allocate a window node */
top_win->over = q;           /* link top of stack to new node */
q->under = top_win;          /* link new node to top of stack */
top_win = q;                 /* set top of stack to new node */
return q;
}
}
static void dispose_window_node(windesc *w)
{
/*
    Dispose_window_node frees up the image save area of the window,
    and the window structure itself.
*/
    if (w != NULL) {         /* safety test for base window */
        if (w->wtype == popup) free(w->image);
        free(w->name);        /* free up window title */
        free(w);              /* and then the structure itself */
    }
}

```

Listing 4.3 Source Code For A Simple Menu Program (POPEX2.C)

```

/*
    Popup menu example (popex2.c)
    Must link with: popup.obj, mouse.obj
*/
#include <bios.h>
#include <conio.h>
#include "popup.h"      /* popup function prototypes */
#include "mouse.h"      /* to get key definitions */
char *menu_entries[] = {
    "a horse of a different color",
    "birds of the same flock",
    "pigs in a pen"
};
void disp_entry(windesc *w, char *entry[], int entryno, int hilite);
void main() {
    windesc *w1, *w2;
    int entryno = 1, i, key;
    init_win();        /* always do this first */
    w1 = draw_win(CTRWIN,CTRWIN,30,5," My Menu ",tile,&monocolors);
    w2 = draw_win(CTRWIN,1,50,7," Selections ",tile,&monocolors);
    /* display menu choices */
    slct_win(w1);
    for(i=0; i<3; i++) printfstr(1,i+1,menu_entries[i],0,80);
    do { /* loop until esc key pressed */
        do { /* make selection */
            disp_entry(w1,menu_entries,entryno,1); /* hilite bar */
            key = bioskey(0);
            disp_entry(w1,menu_entries,entryno,0); /* back to normal */
            switch(key) {
                case UPKEY:
                    if (--entryno < 1) entryno = 3; /* support wrap-around */
                    break;
                case DOWNKEY:

```

```

        if (++entryno > 3) entryno = 1;
        break;
        default: ;
    }
    ) while ((key != CRKEY) && (key != ESCKEY));
    if (key == CRKEY) {
        slct_win(w2);
        mprintf("\r\n%s", menu_entries[entryno-1]);
        slct_win(w1);
    }
    ) while(key != ESCKEY);
    rmv_win(w2); /* free up window memory, (they stay on screen */
    rmv_win(w1); /* cause they're tiled */
    clrscr(); /* so erase 'em *his way */
}

void disp_entry(windesc *w, char *entry[], int entryno, int hilite)
/*
    Displays menu entry number entryno in window w.
    If hilite = 1, then the entry is highlighted. The highlighting
    takes place the width of the window.
*/
{
    int bar_color;
    if (hilite) bar_color = w->wc.hilite_color;
    else bar_color = w->wc.text_color;
    /* bar gets highlighted all the way across window by left justified
    formatting trick.
    */
    prtfsr(1, entryno, "%-*.s", bar_color, 80,
        w->wd-2, w->wd-2, entry[entryno-1]);
}

```

Listing 4.4 A Simple Mouse Driven Menu Program (POPEX3.C)

```

/*
    Popup menus with mouse support example (popex3.c)
    Must link with: popup.obj, mouse.obj
*/
#include <bios.h>
#include <conio.h>
#include "popup.h" /* popup function prototypes */
#include "mouse.h" /* to get key definitions */
char *menu_entries[] = {
    "a horse of a different color",
    "birds of the same flock",
    "pigs in a pen"
};

void disp_entry(windesc *w, char *entry[], int entryno, int hilite);
void main() {
    windesc *w1, *w2;
    int entryno = 1, i, key;
    init_win(); /* always do this first */
    init_mouse(MOUSE_OPTIONAL, MOUSE_TEXT_MODE, MOUSE_TEXT_MODE);
    defcolors = monocolors; /* use if you have monochrome card */
    w1 = draw_win(CTRWIN, CTRWIN, 30, 5, " My Menu ", tile, &defcolors);
    w2 = draw_win(CTRWIN, 1, 50, 7, " Selections ", tile, &defcolors);
    /* display menu choices */
}

```



```

slct_win(w1);
for(i=0; i<3; i++) prtftstr(1,i+1,menu_entries[i],0,80);
do { /* loop until esc key pressed */
    do { /* make selection */
        disp_entry(w1,menu_entries,entryno,1); /* hilite bar */
        while(!(key = mouse_trigger(1))); /* wait for key or mouse event */
        disp_entry(w1,menu_entries,entryno,0); /* back to normal */
        switch(key) {
            case UPKEY:
                if (--entryno < 1) entryno = 3; /* support wrap-around */
                break;
            case DOWNKEY:
                if (++entryno > 3) entryno = 1;
                break;
            case LEFT_MOUSE_PRESS:
                for (i = 1; i<=3; i++) { /* check for mouse on entry */
                    if (mouse_in_box(0,w1->xul+1,w1->yul+i,
                                     w1->xlr-1,w1->yul+i)) {
                        entryno = i;
                        key = CRKEY; /* fake key for selection */
                    }
                }
                break;
            case RIGHT_MOUSE_PRESS:
                key = ESCKEY; /* fake key for exit code */
                break;
            default: ;
        }
    } while ((key != CRKEY) && (key != ESCKEY));
    if (key == CRKEY) {
        slct_win(w2);
        #printf("\r\n%s",menu_entries[entryno-1]);
        slct_win(w1);
    }
} while(key != ESCKEY);
rmv_win(w2); /* free up window memory. (They stay on screen */
rmv_win(w1); /* cause they're tiled */
mouse_reset();
clrscr();
}

void disp_entry(windesc *w, char *entry[], int entryno, int hilite)
/*
    Displays menu entry number entryno in window w.
    If hilite = 1, then the entry is highlighted. The highlighting
    takes place the width of the window.
*/
{
    int bar_color;
    if (hilite) bar_color = w->wc.hilite_color;
    else bar_color = w->wc.text_color;
    /* bar gets highlighted all the way across window by left justified
       formatting trick.
    */
    prtftstr(1,entryno,"%-*.*s",bar_color,80,
             w->wd-2,w->wd-2,entry[entryno-1]);
}

```

Listing 4.5 Moving Windows Program (POPEX4.C)

```
/*
Moving text window example (popex4.c)
Must link with: popup.obj, mouse.obj
*/
#include <conio.h>
#include <bios.h>
#include "popup.h"
#include "mouse.h"
#define MAX(a,b) ((a) > (b) ? (a) : (b))
#define MIN(a,b) ((a) < (b) ? (a) : (b))
char msg[] = "Ready to move";
int mouse_on_border(windesc **w, int *i, int *xofs, int *yofs);
void move_curr_win(int x, int y);
void main() {
    windesc *w[3];
    int x, y, xofs, yofs, i;
    unsigned int k;
    init_win();
    init_mouse(MOUSE_OPTIONAL, MOUSE_TEXT_MODE, MOUSE_TEXT_MODE);
    w[0] = draw_win(5,6,20,5,"\\x11\\\"Move Me #1\\x10",popup,&monocolors);
    w[1] = draw_win(7,9,20,6,"\\x11\\\"Move Me #2\\x10",popup,&monocolors);
    w[2] = draw_win(9,11,20,5,"\\x11\\\"Move Me #3\\x10",popup,&monocolors);
    printf(msg); /* print ready message in current window */
    do {
        while (!(k = mouse_trigger(1))); /* get event trigger */
        if (k == LEFT_MOUSE_PRESS) {
            if (mouse_on_border(w, &i, &xofs, &yofs)) {
                clrscr(); /* clear old window */
                slct_win(w[i]); /* select new one */
                mprintf(msg);
                while(button_state()) { /* move till button release */
                    mouse_txt_posn(&x,&y);
                    move_curr_win(x-xofs,y-yofs);
                }
            }
        }
        else { /* possible keyboard press */
            xofs = 0; yofs = 0; /* reset mouse/window corner offsets */
            x = curr_win->xul; /* default to current position */
            y = curr_win->yul;
            switch(k) {
                case UPKEY:
                    y--;
                    move_curr_win(x,y); /* move_curr_win will do bounds checking */
                    break;
                case DOWNKEY:
                    y++;
                    move_curr_win(x,y);
                    break;
                case LEFTKEY:
                    x--;
                    move_curr_win(x,y);
                    break;
                case RIGHTKEY:
```

```

        x++;
        move_curr_win(x,y);
        break;
case 0x0231:      /* the "1" key */
    clrscr();      /* clear old window */
    slct_win(w[0]); /* select new one */
    mprintf(msg);
    break;
case 0x0332:      /* the "2" key */
    clrscr();
    slct_win(w[1]);
    mprintf(msg);
    break;
case 0x0433:      /* the "3" key */
    clrscr();
    slct_win(w[2]);
    mprintf(msg);
    break;
default:          /*send character to current window */
    if (k != RIGHT_MOUSE_PRESS) mprintf("%c",lo(k));
}
}
) while (k != ESCKEY && k != RIGHT_MOUSE_PRESS);
mvv_win(w[2]);
mvv_win(w[1]);
mvv_win(w[0]);
mouse_reset();
}
int mouse_on_border(windesc **w, int *i, int *xofs, int *yofs)
{
    int x, y, j;
    mouse_txt_posn(&x,&y);
    for (j = 0; j<3; j++) {
        if ((x >= (w[j])->xul && x <= (w[j])->xlr &&
            y >= (w[j])->yul && y <= (w[j])->yur) &&
            (x == (w[j])->xul || x == (w[j])->xlr ||
            y == (w[j])->yul || y == (w[j])->yur)) {
                *xofs = x - (w[j])->xul; *yofs = y - (w[j])->yul;
                *i = j;
                return 1;
            }
    }
    return 0;
}
void move_curr_win(int x, int y)
/*
    Moves the current window. If coordinates haven't changed,
    then nothing happens. Does bounds checking on the new position.
*/
{
    int xsave, ysave;
    if (x != curr_win->xul || y != curr_win->yul) {
        xsave = wherex();
        ysave = wherey();
        swap_image(curr_win); /* hide window */
        curr_win->xul = x;
        curr_win->yul = y;
    }
}

```

```

curr_win->xul = MAX(curr_win->xul, 1); -
curr_win->xul = MIN(curr_win->xul, 81 - curr_win->wd);
curr_win->yul = MAX(curr_win->yul, 1);
curr_win->yul = MIN(curr_win->yul, 26 - curr_win->ht);
curr_win->xlr = curr_win->xul + curr_win->wd - 1;
curr_win->yhr = curr_win->yul + curr_win->ht - 1;
swap_image(curr_win); /* show window */
/* change window coordinates */
window(curr_win->xul+1, curr_win->yul+1, curr_win->xlr-1, curr_win->yhr-1);
gotoxy(xsave, ysave);
}
}

```

Listing 4.6 A Pop-up Error Message Package (SAYERR.C)

```

/*****
 * Popup error reporting package (sayerr.c)
 *****/
#include <stdio.h>
#include <stdarg.h>
#include <conio.h>
#include <bios.h>
#include <string.h>
#include <dos.h>
#include <process.h>
#include <errno.h>
#include "popup.h"
#include "mouse.h"
static char *level_msg[] = {"WARNING", "ERROR"};
static char *pmsg = "Press <Enter> or <ESC> ...";
/*
The way the error/trace windows work: If errw/msgw is NULL, then
a error/trace message is popped up at errx/msgx, erry/msgy.
Else, the window errw/msgw (which should have been already
popped up) will be used. If errw/msgw is already defined, then
the errx/msgx and erry/msgy coordinates are ignored.
*/
int errx = CTRWIN; /* Default to center of the screen */
int erry = CTRWIN;
windesc *errw = NULL; /* Default to popup window */
int msgx = CTRWIN; /* same for messages */
int msgy = CTRWIN;
windesc *msgw = NULL;
void numnewlines(char *s, int *n, int *w)
/* Count number of newlines in the string s, also, return
maximum width of the lines of the message
*/
{
int j, k;
for (*n=0, j = 0, *w=0, k=0; (s[j] != 0); j++, k++) {
if (s[j] == '\n') {
(*n)++;
if (k > *w) *w = k; k = 0;
}
if (k > *w) *w = k;
}
}

```

```

}
void popmsg(int x, int y, char *msg, char *title,
            char soundout, wincolors *wc)
/*
  Pops up the message msg at (x,y)-(absolute coordinates),
  with title as the window header.  if soundout != 0, then the
  alarm will sound.  It will then wait for a key press before
  unpoping then window.
  The message can contain newlines, which are accounted for in
  the size of the window. Note: it SHOULD usually contain a new
  line at the end, or your message might scroll off the screen.
  The (x,y) cursor pos'ns use the same codes as for popup windows.
*/
{
  int wd,ht,c,k;
  windesc *w;
  /* Compute height and width of window */
  numnewlines(msg, &ht, &wd);
  /* add in border, and prompt message to size computations */
  ht += 3;
  if (wd < (strlen(msg)+1)) wd = strlen(msg)+3; else wd += 3;
  w = draw_win(x, y, wd, ht, title, popup, wc);
  mouse_off(1);
  cprintf(msg);
  cprintf(msg);
  mouse_on(1);
  if (soundout) beep();
  do {
    while(!(k = mouse_trigger(0)));
    if (k == CTRLC) exit(0); /* control c aborts */
    if (k != CRKEY && k != ESCKEY) beep(); /* only cr or esc allowed */
  } while(k != CRKEY && k != ESCKEY);
  rmv_win(w);
}

void reperr(int level, char *msg)
/* Reports the error in the error window.
   level = 0 means warning, level = 1 means error.
*/
{
  windesc *wsave;
  if (errw == NULL) { /* popup up error window */
    popmsg(errx, erry, msg, level_msg[level], 1, &errcolors);
  }
  else { /* else route to existing error window */
    wsave = curr_win;
    slct_win(errw);
    mprintf("%s: %s", level_msg[level], msg);
    slct_win(wsave);
  }
}

void repmsg(char *msg)
/* Reports the message in the message window. */
{
  windesc *wsave;

```

```

    if (msgw == NULL) { /* popup message window */
        popmsg(msgx, msgy, msg, " Msg " 0 &msgcolors);
    }
    else { /* else route to existing message window */
        wsave = curr_win;
        slct_win(msgw);
        mprintf("Msg: %s", msg);
        slct_win(wsave);
    }
}

void sayerr(int ferr, int errflag, int lno,
            char *pname, char *fmt,...)
/*
    The parameters work as follows:
    ferr      if 1, then the last DOS error message is printed
    errflag   if 1, treat as warning, 2 treat as error, else
              just treat as a message
    lno,pname if pname = "", they're ignored, else
              they're treated as a line number and source file name.
    fmt,...   a format string followed by optional arguments
    NOTE: formatting should not exceed 255 characters
*/
{
    va_list arg_ptr;
    char t[255];
    int j;
    if (*pname) {
        j = sprintf(t, "On line %d in pgm %s\r\n", lno, pname);
    }
    else j = 0;
    if (ferr == 1) {
        j += sprintf(t+j, "%s\r", strerror(errno)); /* add last dos error */
    }
    va_start(arg_ptr,fmt); /* point to optional arguments */
    vsprintf(t+j, fmt, arg_ptr); /* add rest of formatted string */
    va_end(arg_ptr);
    switch(errflag) {
        case 1:
            reperr(0,t); /* just a warning */
            break;
        case 2:
            reperr(1,t); /* an error */
            break;
        default:
            repmsg(t); /* or a plain old message */
    }
}

unsigned int getkey(void)
/*

```

Waits for and returns the scan-ascii code of the next key available.
Does not echo. Pops up an abort window on ctrl-c.

```

*/
{
    windesc *w;
    int k;
    while(1) { /* loop until non-ctrl-c key, or abort */
        k = bioskey(0);
        if (k == CTRLC) {
            w = draw_win(CTRWIN, CTRWIN, 25, 3, "", popup, &errcolors);
            mprintf("Abort program (Y/N) ?");
            k = bioskey(0);
            rmv_win(w);
            if ((k == 0x1559) || (k == 0x1579)) { /* "Y" or "y" */
                mouse_reset(); /* don't forget to reset the mouse !! */
                exit(1);
            }
        } else break;
    }
    return k;
}

void beep(void)
/* sounds the bell */
{
    sound(50);
    delay(25);
    nosound();
}

```

Listing 4.7 Using the Pop-up Error Message Package (POPEX5.C)

```

/*
    Popup error message reporting example (popex5.c)
    Must link with: popup.obj, mouse.obj, sayerr.obj
*/
#include <conio.h>
#include <fcntl.h>
#include <io.h>
#include "popup.h"
#include "mouse.h"
char myfile[] = "ghost.dat";
void main() {
    int fh;
    windesc *mainw;
    init_win(); /* always, always, always do this */
    mainw = draw_win(1,1,80,17," Main Window ",tile, &defcolors);
    /* error messages default to popup windows */
    mprintf("Will try to open file when you press return ...");
    getch();
}

```

```

if ((fh = open(myfile,O_RDONLY)) == -1)
    sayerr(FERRE, "---- %s ----\r\n", myfile);
/* but you can reroute them */
mprintf("\r\nPress return to construct static error window ...");
getch();
errw = draw_win(1,17,80,9," Error window ", tile, &errcolors);
slct_win(mainw);
mprintf("\r\nReady to try opening again, press return ...");
getch();
if ((fh = open(myfile,O_RDONLY)) == -1)
    sayerr(FERRE, "---- %s ----\r\n", myfile);
mprintf("\r\nPress return to see error message without"
        " line no and source file ...");
getch();
/* example of printing without line no and source file */
if ((fh = open(myfile,O_RDONLY)) == -1)
    sayerr(FERR, "---- %s ----\r\n" myfile);
mprintf("\r\nPress return to see it as a warning ...");
getch();
/* example of printing just as a warning */
if ((fh = open(myfile,O_RDONLY)) == -1)
    sayerr(FWRN, "---- %s ----\r\n", myfile);
mprintf("\r\nPress return for a wise saying ... \r\n");
getch();
/* example of printing a formatted message */
sayerr(SMSG, "The cows jumped over the %s\r\nand landed on %s\r\n",
        "moon", "mars");
rmv_win(errw);
rmv_win(mainw);
clrscr();
}

```


第五章 文件 I/O

Turbo C 最令人混淆的部分大概是文件 I/O。象大多数 C 编译程序一样, Turbo C 提供了两种不同的文件存取方法并有许多文件存取函数。Turbo C 使用手册无助于解决这一混淆, 因为它们对有两类不同方法, 甚至对这些方法一般不应混淆都讲得很少。实际上, 使用手册对文件 I/O 也很少讲到。由于有大量的文件 I/O 函数, 你可能不知选用哪个函数。本章讲清其中某些函数。

两类文件存取是标准 I/O 和系统级 I/O。标准 I/O 例行程序代表了 C 中文件存取的通常方法, 更完整且易于使用。有时它们被指为流 I/O 例行程序, 因为它们最常见的应用是做为字符流的对文件的存取。标准 I/O 例行程序提供带缓冲区的对文件的存取, 且比操作系统提供的层次较高, 这使得这些程序相当独立于操作系统。另一方面系统级 I/O 例行程序处于较低层次且其中有些直接调用操作系统。由于它们更直接, 所以它们趋向于运行得更快且占更少的内存。然而, 它们不太容易使用, 因为提供的函数较少, 最引人注目的是自动缓冲和格式化。表 5.1 显示了可用于标准 (即, 流) I/O 的函数, 和相应的系统级函数。

Table 5.1 Standard versus System-Level Functions

| Standard I/O | System-Level I/O | Standard I/O | System-Level I/O |
|--------------|------------------|--------------|------------------|
| - | chsize | fcloseall | - |
| clearerr | - | fdopen | - |
| - | dup | feof | eof |
| - | dup2 | ferror | - |
| fclose | close | fflush | - |
| - | _close | fgetc | - |
| fgetchar | - | freopen | - |
| fgetpos | - | fscanf | - |
| fgets | - | fseek | lseek |
| fileno | - | fsetpos | - |
| flushall | - | fstat | fstat |
| fopen | open | ftell | - |
| - | _open | fwrite | write |
| - | _creat | - | _write |
| - | creat | rewind | - |
| - | creatnew | setbuf | - |
| - | creattemp | setvbuf | - |
| fprintf | - | stat | stat |
| fputc | - | ungetc | - |
| fputs | - | vfprintf | - |
| fread | read | vscanf | - |
| | _read | | |

标准 I/O 例行程序有时被错误地认为是只提供流 I/O，即把文件当作字符序列或流读写。尽管那是它们最广泛的应用，但它们确实提供远多于此的功能。例如，你可以做格式化的输入和输出，在文件上执行随机存取，执行块或记录 I/O。标准 I/O 常与文本文件存取相关联，而系统级 I/O 常与二进制文件存取相关联。然而，应指出的是，可用标准 I/O 例行程序进行二进制文件存取。

这两种方法的一个重要差别是标准 I/O 提供文件的内部缓冲，由于有较少的磁盘存取而使之更有效。系统级例行程序无这种缓冲，但由于系统级例行程序直接调用操作系统，它们因较少的开销而更快。使用哪种方法决定于程序员的偏爱，和所涉及的应用类型。我们将主要集中于标准 I/O。这是因为它提供更多的功能特性且是广泛使用的方法。

5.1 文本与二进制文件

进一步增加混淆的是，有两类不同的既用于标准又用于系统级 I/O 的文件存取模式。它们是文本和二进制模式。在文本模式，读时一个回车——换行对被翻译为一个单个新行符，在写时，新行符又被译回为一个回车——换行对。二进制模式无此翻译。

文本模式存取的原因是历史性的。过去，C 是在 Unix 系统上开发的，文本文件中行尾字符仅仅是个单个新行符。但 DOS 文本文件用一个回车——换行组合。为使从 Unix 世界到 DOS 移植程序更容易，就发明了存取的文本模式。

有个全程变量 `fmode`，它可被设置以指明所有文件存取的翻译模式。变量 `fmode` 可取下列值：

```
O_BINARY    - No CRLF translation
O_TEXT      - CRLF pairs translated into a newline character
```

`fmode` 的系统设置值是 `O-TEXT`。在文件打开时有可能越过翻译模式，这一点你将很快见到。

5.2 文件指针与文件把柄

一旦一个文件被打开后，你引用它的方法就决定于它是为标准 I/O 打开还是为系统级 I/O 打开。对一个标准 I/O 文件，Turbo C 在定义于头文件 `stdio.h` 中的一个结构中保留有关于这个文件的某些信息：

```
typedef struct {
    short      level;           /* Fill/Empty level of buffer */
    unsigned   flags;           /* File status flags           */
    char       fd;              /* File descriptor (handle)    */
    unsigned char hold;         /* Ungetc char holder         */
    short      bsize;           /* Buffer size                  */
    unsigned char *buffer;      /* File buffer pointer         */
    unsigned char *curp;        /* Current active pointer      */
    unsigned   istemp;          /* Temp file flag              */
    short      token;           /* Validity marker             */
} FILE;
```

无论何时为标准 I/O 存取打开一个文件，都有一个 FILE 结构被创建并且一个指向该结构的指针被返回（称为流指针）。对别的文件操作，这个指针用于引用打开的文件。你永远也不要直接存取 FILE 结构中的任何部分，因为它们仅用于内部用途。它们显示在这里仅仅是给你一个 TurboC 保存了些什么的概念。别的编译程序可能用一完全不同的结构，尽管它们也定义类型 FILE。

FILE 结构的 fd 域称为文件柄。文件柄是在文件被打开时由 DOS 返回的对每个打开文件唯一的一个号。对系统级文件，这个号用于为所有别的操作引用文件，而非一个流指针。在系统级打开的文件无相关的 FILE 结构。

下面的程序段比较了打开和引用文件的两种方法。在两者中，文件都为只读存取打开：

```
f = fopen("myfile.dat", "r"); /* open for read only */
fclose(f);                    /* then close it */
}
/* System level I/O method */
#include "fcntl.h"
#include "io.h"
int fhandle;
main() {
    fhandle = open("myfile.dat", O_RDONLY); /* open for read only */
    close(f);                               /* then close it */
}
/* standard I/O method */
#include <stdio.h>
FILE *f;
main() {
```

总的来说，混淆这两种文件存取形式是不明智的。这是因为标准 I/O 操作是被缓冲的，而且，如果你用系统级调用进行读/写或寻找，缓冲区将不再与应在该文件里的数据对应。但某些函数只用文件柄工作，所以 Turbo C 提供了宏 `fileno()` 用于为流文件恢复文件柄。正如你可能已猜到的那样，这个宏仅仅返回 FILE 结构中的 fd 域并且它被定义于 `stdio.h` 中：

```
define fileno(f) (f) - 1d)
```

最好是用这个宏，而不是直接存取 fd，因为不能保证 FILE 结构将来不变。另外，如我们前面提及的，别的编译器推销商可能使用一个完全不同的 FILE 结构。

通过函数 `fdopen()`，有办法把 FILE 结构附在已在系统级打开的文件上。这允许你在这种文件上做缓冲和格式化，及混淆这两种存取形式。如果你决定用这一功能特性，则必须非常小心。

5.3 DOS 文件信息

DOS 本身为每个打开文件都保存一个内表，通常足以保存 20 个文件。实际数目由你的 `config.sys` 文件中的 FILES 参数设置。关于打开文件的可得的某些信息可以通过检查包含在头文件 `<sys\stat.h>` 中的 `stat` 结构获得：

```

struct stat {
    short st_dev;           /* drive or device number      */
    short st_ino;           /* not used in DOS, for UNIX  */
    short st_mode;          /* read/write mode and access bits */
    short st_nlink;         /* set to 1                    */
    int st_uid;             /* Unix user ID, not used in DOS */
    int st_gid;             /* Unix group ID, not user in DOS */
    short st_rdev;          /* same as st_dev              */
    long st_size;           /* size of the open file        */
    long st_atime;          /* time of most recent update    */
    long st_mtime;          /* for DOS, same as st_atime    */
    long st_ctime;          /* for DOS, same as st_atime    */
};

```

对为标准 I/O 打开的文件，这一信息加之由 FILE 结构提供的信息都可得到。

这个结构中的大多数域是为 Unix 兼容性提供的，并非全都由 DOS 实际使用。这个结构可由 TURBO C 函数 `stat()` 和 `fstat()` 获得。前者取一个打开文件的文件名并返回其信息，后者取文件柄。如果你的文件是流文件，则这就是使用宏 `fileno()` 的情况，这样你就可用适当的文件把柄调用 `fstat()`。

这里看起来关于一个打开文件你所需知道的每一件事都可得到，或由 FILE 结构或由 `stat` 结构；不幸的是并非如此。一旦你打开了一个文件，则无处保留文件的 NAME。DOS 只知道文件把柄。这使做对用户友好的错误报告有些困难。假设你写了一个从一文件中读字节的通用例行程序，并且希望在发生错误时打印出一个错误信息。你必须设法保持文件名记录，以便你能打印出“Error reading myfile.dat”这样的信息而非“Error reading file #2”。前一信息对用户来说肯定含有更多的信息。本章稍后开发的样例文件 I/O 软件包专用于处理这种问题。

5.4 预定义的流和把柄

有五个预定义的流，它们在你的程序执行时被自动赋值：

| stream | Handle |
|---------------------|--------|
| <code>stdin</code> | 0 |
| <code>stdout</code> | 1 |
| <code>stderr</code> | 2 |
| <code>stdaux</code> | 3 |
| <code>stdprn</code> | 4 |

这些流定义于 `stdio.h` 中，指标准输入，标准输出，标准错误，标准辅助，和标准打印流。这些符号名是流指针，所以任何可用标准 I/O 文件参考的地方你都可用它们。这些流还有所示的预定义把柄。你可用这些把柄在系统级存取这些流。

系统设置值是：`stdin`、`stdout` 和 `stderr` 指用户控制台。`stdaux` 和 `stdprn` 流的分配依赖于你的具体机器。它们通常与辅助口和打印机相关联。

你可用 `freopen()` 函数重定义这些流以使它们指向一个磁盘文件或一个不同的设备。这个函数关闭当前与一个流相关联的文件，并把它重分配给一个新文件。如果流是附于一个设备之上的，则关闭操作什么也不做。你也可用两个函数 `dup()` 和 `dupz()` 做本

质上相同的事。

既然我们已讲了 Turbo C 中的基本知识, 那么我们就将先检验怎样做标准 I/O, 随后看怎样做系统级 I/O。本章结尾提供了一个使用标准 I/O 函数的例行程序的软件包。

5.5 标准 I/O

做标准 I/O 的例行程序可划分为 6 类:

- 打开/关闭/填充缓冲区
- 返回文件状态
- 移动当前读/写位置
- 做字节级的读/写
- 做串级的读/写
- 做记录, 或块级的读/写

表 5.2 到 5.7 显示了与各类相关的函数。这些例行程序中的一些已提及过, 但在本节我们将仔细研究它们。

5.5.1 打开和关闭标准 I/O 文件

表 5.2 显示了可用于为标准 I/O 打开和关闭文件的最普遍的例行程序: `fclose()`, `fcloseall()`, `fopen()`, 和 `freopen()`。

Table 5.2 Opening, Closing, and Flushing Stream File Buffers

| Function | Use |
|--|--|
| <code>int fclose(FILE *stream)</code> | Closes a stream |
| <code>int fcloseall(void)</code> | Closes all stream files |
| <code>FILE *fdopen(int handle, char *type)</code> | Attaches a stream file to a system-level file |
| <code>int fflush(FILE *stream)</code> | Flushes a stream file buffer |
| <code>int fflushall(void)</code> | Flushes all stream file buffers |
| <code>FILE *fopen(char *filename, char *type)</code> | Opens a file for stream access |
| <code>FILE *freopen(char *filename, char *type, FILE *stream)</code> | Reopens a stream to a different device or file |
| <code>void setbuf(FILE *stream, char *buf)</code> | Turns buffering on/off |
| <code>int setvbuf(FILE *stream, char *buf, int type, unsigned size)</code> | Turns on buffering with arbitrary-sized buffer |

尽管有三类不同的可用于打开标准 I/O 文件的函数, 但 `fopen` 是最常用的一个。先传给它即将打开的文件名, 随后传给它一个指定文件存取类型的字符串。这些类型如下:

Types of File Access

| Type | Description |
|------|---|
| "r" | The file is opened for read-only access. It is an error if the file does not exist. |
| "w" | The file is opened for write-only access. If the file exists, its contents |

| | |
|------|--|
| | are destroyed; otherwise, the file is created. |
| "a" | The file is opened for writing at the end of the file (i.e., appending). If the file does not exist, it is created. |
| "r+" | The file is opened for reading and writing. It is an error if the file does not exist. |
| "w+" | The file is opened for reading and writing. If the file exists, its contents are destroyed; otherwise, the file is created. |
| "a+" | The file can be read at any position, but writing always occurs at the end of the file (i.e., only appending is allowed). If the file does not exist, it is created. |

基本上, 如果你想创建一个文件, 则可用以“w”或“a”开始的任何类型。对“w”类型的存取, 如果文件存在, 则其内容将被覆盖。“a”类型存取允许你在一个旧文件后附加东西而不覆盖前面的内容。如果你想访问一个旧文件而且想在它不存在时返回一个错误, 则可选用“r”类型存取。

用存取类型“rt”, “wt”或“at”打开的文件据说是可为更新打开, 且允许读和写。但在混合读和写时必须过细。每次你从一个转换到另一个时, 必须先做一个插入的 `fseek()` 或 `rewind()`。这样文件指针 (即, 文件中的当前位置) 可被正确更新。本章结尾开发的文件 I/O 软件包将方便地为我们提供这种功能。

为附加而打开的文件按下面的方式活动: 虽然你可用 `fseek()` 或 `rewind()` 把文件指针移到文件中任一地方, 但每次执行写操作时, 它都是在文件尾完成的。这意味着旧数据不会被附加模式覆盖。

下面显示了一个为更新而打开一个旧文件的例子。如果文件被成功地打开, 则 `fopen()` 返回一个 `FILE` 指针。如果出错, 则返回 `NULL`。该例也表明了如何关闭文件:

```
/* Standard I/O file example (filex1.c) */
#include <stdio.h>
main() {
    FILE *f; /* this is how we refer to the file */
    if ((f = fopen("c:\\mydir\\myfile.dat", "r+")) == NULL) {
        printf("Error opening file, probably no such path\n");
    }
    else {
        printf("File successfully opened for updating.\n");
        fclose(f);
        printf("And is now closed.\n");
    }
}
```

注意路径名中的“\\”串。指定 DOS 系统上的路径名时试图讲一些象“mydir\\myfile”这样的东西是一个普遍的错误。反斜杠 (在 Unix 系统上是另一方向) 实际上是一个 ESC 字符, 所以你必须把它包括两次。很多失败的时间都可能由此引起的。这一点你可花许多时间尝试着查明为什么你的程序不能打开一个你肯定存在的文件。

在指定存取类型时, 你也可指明一个文件是为文本模式还是二进制模式打开。这可由为文本模式在存取类型尾加一“t”, 为二进制模式加一“b”来实现。下例被改变以使文件为二进制更新模式打开。

```

/* Binary file access example (filex2.c) */
#include <stdio.h>
main() {
    FILE *f; /* this is how we refer to the file */
    /* Note the "b" in "r+b" */
    if ((f = fopen("c:\\mydir\\myfile.dat", "r+b")) == NULL) {
        printf("Error opening file, probably no such path\n");
    }
    else {
        printf("File opened for updating in binary mode.\n");
        fclose(f);
        printf("And is now closed.\n");
    }
}

```

如前所述，系统设置的模式在-fmode 变量中指定，它本身由系统设置为文本模式。这样，关于用哪种模式不会产生混淆。但如果你正从 Unix 移植程序（或打算这样做），设置-fmode 变量方便得多，否则移植程序时必须修改每一个 fopen() 调用。

另两个打开例程序 fdopen() 和 freopen() 取类似的参数，正如 Turbo C 使用手册所示。

5.5.2 获取文件状态

表 5.3 显示了可用于获得一个打开文件的状态的函数。stat() 和 fstat 函数已描述过了。如果最后的 I/O 操作中判别到了文件尾标识符则 feof() 函数返回一个非零值。ferror() 函数在读和写操作期间流上有错误发生则返回一个非零。这两个状态在做对 clearerr()、fseek()、rewind() 或 fsetpos() 调用之前保持被设置。所有这些函数都清除文件尾标识符，但只有 rewind() 和 clearerr() 清除别的状态标识符。

Table 5.3 File Status for Stream Files

| Function | Use |
|---|---|
| void clearerr(FILE *stream) | Clears errors associated with a stream |
| int feof(FILE *stream) | Returns end-of-file status |
| int ferror(FILE *stream) | Returns last read/write error on stream |
| int fstat(char *handle, struct stat *buff) | Returns file status for a given file handle |
| int stat(char *pathname, struct stat *buff) | Returns file status for a given file-name |

当出现文件错误时，DOS 也保存该错误状态。Turbo C 函数 strerror() 和 perror() 允许你打印出最近的 DOS 错误信息。两个全程变量 errno 和 doserrno 保持最近错误的记录。errno 变量是个 Unix 兼容的错误标识符且是由 perror() 和 strerror() 使用的一个。-doserrno 变量被设置为实际 DOS 错误码。虽然这两个标识符有时有等价的码，但总的来说它们具有的码是不同的。码对应关系在 Turbo C 参考指南 (Turbo C Reference Guide) 中给出。

errno 和 -doserrno 标识符反映了设置它们的最后一次调用的错误值。许多 Turbo C I/O 函数都设置这些标置, 但你应查阅具体函数的文献以保证有把握。一定要在下一个可能设置它们的函数被调用之前使用这些标识符, 这样可得到正确的错误状况。

5.5.3 控制文件缓冲

Turbo C 提供了四个允许你控制标准 I/O 文件缓冲的例行程序: fflush(), flushall(), setbuf() 和 setvbuf()。当一个标准 I/O 文件被打开后, 系统就自动分配给它一个缓冲区。这个缓冲区既用于读又用于写。一个文件读操作实际上引起整个文件块字节被读, 而非仅仅是指定的数目。在文件写操作期间, 字节实际上是先写向文件缓冲区的。当缓冲区满后, 它就被填充到磁盘上。如果缓冲区大小合适, 则文件存取可非常有效地执行。Turbo C 提供了控制缓冲的方法, 所以你对你的具体的应该使用优化文件存取。

可能会有缓冲引起的问题。如果你碰巧在一个文件缓冲区充满之前放弃程序, 则数据可能丢失。另外, 从读到写转换时也有个问题, 因为缓冲区内容可能混淆。这就是为什么 Turbo C 要求你在改变 I/O 方向之前用 fseek() 或 rewind() 重定向文件指针。这些操作可在移到新文件位置之前有效地把缓冲区装满。要声明的是, Turbo C 在关闭文件时将自动地填满缓冲区。另外, 如果你从 exit() 函数或用通常终止方法退出程序, 则所有缓冲区都将填满且所有文件都被关闭。但是, 最好总是在完成时显式地关闭文件。这是为了释放内部文件表中限制的空间以使别的文件能打开。

你也可在任何时候用例行程序 fflush() 使缓冲区填满。还有一个例行程序 flushall() 填充所有打开的流。这些例行程序是用于保持文件完整性的。在程序应被非正常地放弃的情况下, 在向文件写完关键数据之后填充缓冲区倒是个好主意; 否则数据可能丢失。我们将在关于变长记录文件 I/O 的那一章看到这样的例子。

Turbo C 允许你用两个例行程序: setbuf() 和 setvbuf() 完全关闭缓冲或改变缓冲区的大小。如果你想使用自己的缓冲区, 或想关掉缓冲, 则用 setbuf() 例行程序。如果你用这个例行程序指定自己的缓冲区, 则它必须是固定大小的, 如由 stdio.h 中 BUFSIZ 宏指定的, 它被设置为 512 字节。在打开一个文件或填充该文件缓冲区后, 只应立即使用 setbuf(); 否则, 缓冲区内容可能会混淆。下面的程序表明了一些使用 setbuf() 的例子:

```
#include <stdio.h>
char mybuff[BUFSIZ]; /* must have BUFSIZ bytes in the buffer */
main() {
    FILE *f, *g, *h;
    f = fopen("file1.dat", "r+");
    g = fopen("file2.dat", "r+");
    h = fopen("file3.dat", "r+");
    setbuf(g, NULL); /* this is how you turn buffering off */
    setbuf(h, mybuff); /* this is how you specify your own buffering */
    /* ... */
}
```

在 setbuf() 调用之后, 文件 f 使用系统分配给它的缓冲区, 文件 g 不用缓冲, 文件 h 用一个用户定义的缓冲区。

Turbo C 也允许你控制缓冲区的大小, 用函数 `setvbuf()` 即可。它的参数有: 指向即将使用的缓冲区的指针、缓冲类型和缓冲区的大小。类型参数必须是下列之一:

| | |
|---------------------|---|
| <code>_IONBF</code> | Use this to turn off buffering, regardless of the other parameters, which are ignored in this case. |
| <code>_IOLBF</code> | The file is "line buffered". Anytime a newline is written, the buffer is automatically flushed. Read operations are still fully buffered. |
| <code>_IOFBF</code> | Use this for full buffering (the normal mode). |

使用 `setvbuf()` 的一些例子包括如下:

```
#include <stdio.h>
char mybuff[8192]; /* We'll set buffer to 8k */
main() {
    FILE *f, *g, *h;
    f = fopen("file1.dat", "r+");
    g = fopen("file2.dat", "r+");
    h = fopen("file3.dat", "r+");
    setvbuf(f, NULL, _IONBF, 0);
    setvbuf(g, mybuff, _IOFBF, sizeof(mybuff));
    setvbuf(h, NULL, _IOFBF, 1024);
    /* ... */
}
```

`setvbuf()` 调用之后, 文件 `f` 不再有缓冲。注意该调用的别的参数被忽略。文件 `g` 将使用 8k 的缓冲区 `mybuff`, 文件 `h` 将使用一个 1k 的缓冲区, 该缓冲区由 `malloc()` 的间接调用分配。使用这一功能特性要仔细, 因为没有提供除调用 `fclose()` 之外的释放以这种方式分配的内存的办法。到 `malloc` 的间接调用设置一个标志以便 `fclose` 能正确释放缓冲区。

要注意的另一件事是, 使用一个是一函数的局部变量的缓冲区 (即, 在运行时间的栈上的那个)。如果这个函数在文件被关闭之前退出, 则即使这个堆栈位置不再分配给该缓冲区, 进一步的文件操作仍将继续使用这个栈位置。其结果将很可能是系统崩溃。因为这个原因, 最好是在文件定义之外说明文件缓冲区或把它们说明为动态变量。

有些情况流文件没有动态缓冲。例如, `stdin` 和 `stdout` 流就不是正常地被缓冲的, 因为它们通常是与控制台设备 (即屏幕和键盘) 相联系的, 那里缓冲毫无意义。如果这些流被重定向到“真实”文件, 则缓冲将自动提供。

总之, 缓冲区控制例行程序是 C 中提供的多用途的好例子。对大多数 C 操作, 都有关 C 怎样执行这些操作的精巧的控制。这种用户化和优化基本函数的能力是为什么 C 被认为是“有效的”语言的原因之一。这也是它难学的原因之一, 因为有许多操作要省略。

5.6 对文件的随机访问

表 5.4 显示了可用于把文件指针移到文件内任一位置的例行程序。最常用的是 `fseek()` 和 `rewind()`, `rewind()` 函数把文件指针移到文件的开头, 而 `fseek()` 函数把指针移到指定的偏移量。偏移量的原点做为一个参数被传递且应有下列值之一:

SEEK_SET offset from beginning of file
 SEEK_CUR offset from current position
 SEEK_END offset backwards from end of file

最常用的可能是 SEEK-SET，因为它以“绝对”方式移动文件指针，以从文件头开始的字节偏移量的数日表示。下面的程序为更新而打开一个文件，如果它已存在则清除它，并随后移到字节 100 处并写出一些文本。

```
/* File seek example (filex3.c) */
#include <stdio.h>
void main() {
    FILE *f;
    /* Open file for text mode, overwrite it if it already exists */
    if ((f = fopen("myfile.dat", "w+t")) != NULL) {
        fseek(f, 100, SEEK_SET); /* Go to byte 100 */
        fprintf(f, "Hello world\n"); /* Write some bytes to it */
        fclose(f);
        printf("myfile.dat successfully written to\n"
            "There'll be garbage in first 100 bytes\n"
            "So only inspect file with hex dump pgm\n");
    }
    else {
        printf("Error opening file: myfile.dat\n");
    }
}
```

这个例子表明在文件尾之外寻找并写向它是完全合法的，有效地允许你扩展文件的大小。但是要注意，扩展部分的字节是未初始化的，所以你也也许想向文件写一些有意义的东西。另外，即使你能写到文件尾之外，但如果你试图读文件尾之外的东西则将返回一个错误。

函数 ftell() 允许你得到相对于文件头的文件指针的当前位置。这个位置是做为一个长类型返回的而非一个整数，所以也可用于大于 64k 的文件。

使用 ftell() 还有细微的问题，例如在文件为附加而打开时。这种情况下，ftell() 返回由最后的 I/O 操作确定的当前文件位置，下一次写不一定发生在这个位置。这是因为你能在附加模式寻找到任一位置，但写总是发生在文件尾。所以要细心。

另一问题是在文本模式下用 ftell() (还有 fseek())。由于回车——换行翻译，ftell() 返回的值可能并不代表真实的从文件头开始的字节偏移量。但是你能用 ftell() 与 fseek() 连同一起记忆并返回一个文件位置，而且它将正确工作。

Table 5.4 Random Access Functions for Stream Files

| Function | Use |
|--|---|
| int fgetpos(FILE *stream, fpos_t *pos) | Gets the current file position for later restoration |
| int fseek(FILE *stream, long offset, int origin) | Moves to a new file position |
| int fsetpos(FILE *stream, const fpos_t *pos) | Sets the file position to that obtained by an earlier call to fgetpos |
| long ftell(FILE *stream) | Gets the current file position. |

`int rewind(FILE *stream)`

Moves the file position back to the beginning

你也可用函数 `fgetpos()` 和 `fsetpos()` 存储及返回文件位置。这些函数是新 ANSI C 标准的一部分。`fgetpos()` 函数取当前文件位置并把它存储在 `fpos_t` 类型的一个目标中, 这个类型定义于 `stdio.h` 中。函数 `fsetpos()` 随后可取此目标并在该程序稍后时间恢复文件指针。存储的目标是只为内部使用的且不应改作它用。在 TurboC 中, 类型 `fpos_t` 被定义为存在类型那么长, 它就是 `ftell()` 所使用的。但是, 并不担保在所有的系统中它都是这种类型。函数 `fgetpos()` 和 `fsetpos()` 被提供作为存储和恢复文件位置的新 ANSI C 标准的方法。

`fsetpos()` 和 `ftell()` 恢复该流上对 `ungetc()` 的任何先前的调用, 所以要仔细。还要记住随机访问函数只在真实文件上有意义。附在设备 (如控制台) 上的流与随机存取不兼容。

5.6.1 读、写标准 I/O 文件

Turbo C 提供对标准 I/O 文件的三类读和写: 如表 5.5, 5.6, 5.7 所示, 你可以在字符级、串级和块级写。

Table 5.5 Functions for Single-Character Access

| Function | Use |
|--|-------------------------------------|
| <code>int fgetc(FILE *stream)</code> | Gets the next character from a file |
| <code>int fgetchar(void)</code> | Gets the next character from stdin |
| <code>int fputc(int ch, FILE *stream)</code> | Puts a character in a file |
| <code>int ungetc(char *c, FILE *stream)</code> | Puts back a character that was read |

Table 5.6 Functions for String-Level Access

| Function | Use |
|---|---|
| <code>char *fgets(char *string, int n, FILE *stream)</code> | Reads the next line from a file into a string |
| <code>int fprintf(FILE *stream, char *fmt, ...)</code> | Does a formatted print to a file |
| <code>int fputs(char *string, FILE *Stream)</code> | Does an unformatted print to a file |
| <code>int fscanf(FILE *stream, char *fmt, ...)</code> | Does a formatted read from a file |
| <code>int vfprintf(FILE *stream, char *fmt, va_list param)</code> | Same as printf, but provides hooks for the variable number of argument calling convention |
| <code>int vfscanf(FILE *stream, char *fmt, va_list arg)</code> | Same as fscanf, but provides hooks for the variable number of argument calling convention |

5.6.2 字符级和串级访问

除了某些次要的例外，文件 I/O 的字符和串函数的作用与其控制台 I/O 对等部分相同。所以这里将不过于详细地讨论它们。我们主要集中于块级访问。

有一个串级函数确需提及。fgets() 函数基本上象其 gets 对等部分一样工作，从一个文件中读取一个字符串。但与 gets() 不同，fgets() 保留任何一个新行字符读。如果你的程序原是为控制台 I/O 所写而你用 fgets() 更换 gets() 来把它扩展来处理文件 I/O，则它可能会把你绊倒。注意这一个。

5.6.3 记录级访问

表 5.7 中所示的 fread() 和 fwrite() 函数，允许你在流文件上做块或记录级 I/O。这里所定义的记录仅仅是一块非格式化的字节块。有两类记录：定长的和变长的。这里只讨论定长记录。变长记录将在后面的章节作为文件系统的扩展给出。值得指出的是它们也用 fread() 和 fwrite()。

Table 5.7 Functions for Block-Level Access

| Function | Use |
|--|--|
| <code>int fread(void *ptr, int size, int nitems, FILE *stream)</code> | Reads a series of bytes from a stream file |
| <code>int fwrite(void *ptr, int size, int nitems, FILE *stream)</code> | Writes a series of bytes to a stream file |

首先，记录 I/O 的含意是什么？它仅仅是一种把文件当作多字节记录，而非字符序列读取的能力。记录以其内部表示存储并且不执行任何文本格式化。记录 I/O 的一个例子是报表文件的存储部分。典型地，这部分被定义为一个结构，如下例所示。该程序把这一部分作为文件的第六个记录用寻找适当字节位置的办法存储。注意偏移量变量的适当类型分配。尽管这里并不严格地有必要，但还是包括进了提醒你偏移量是长类型的：

```
/* Record I/O example (filex4.c) */
#include <stdio.h>
typedef struct my_parts_struct {
    int part_code;
    int quantity;
    float price;
} part;
main() {
    FILE *f;
    int nb;
    part mypart = {
        15, /* part code */
        300, /* quantity */
        1.25 /* price */
    };
    if ((f = fopen("myfile.dat", "w+b")) != NULL) {
        fseek(f, (long)(sizeof(part)*6), SEEK_SET);
```

```

        nb = fwrite(&mypart,sizeof(part),1,f);
        printf("The number of records written is %d\n",nb);
        fclose(f);
    }
    else {
        printf("Error opening file ...\n");
    }
}

```

fwrite() 函数的作用是把第一个参数指向的记录数据写入到该文件。第二个参数告诉我们记录的大小，第三个参数告诉有多少这样的记录写出。所以，即将写入的缓冲区至少应含有 size-of-rec * num-of-rec bytes 字节。这里，记录大小是部分结构的大小。虽然用累加部分结构中各域的大小的办法你自己能够计算出这个大小，但 size of () 函数是一种好得多的方法。实际上极力推荐你用它，这一点你将很快看到。

5.6.4 结构压缩

在许多 C 编译程序上，结构中有填充，通常用于它们核准到字边界。在 IBM PC 上一种典型的压缩方法是在整数边界上。Turbo C 中系统设置的值是最紧压缩，(字节边界)，但其它编译程序也许不同。不要冒犯错误之险（并防止许多混淆），你应该使用 size of() 来让编译程序为你计算大小。

如果你想在结构的各域中有整数核准，则可用 Turbo C 提供的“-a”编译选择。用了这个选择，下面所列将会发生：

1. 结构将总是始于内存中的一个偶地址。
2. 结构中的任一非字符域都将始于自该结构开头的一个偶偏移量。
3. 结构被填充了以确保它含有偶数个字节。

使用这种填充可产生更有效的程序，这就是为什么它被包含进来作为一项选择。

这种结构校准的后果是如果不考虑结构填充则由另一编译程序写的程序也许不能读由 TurboC 创建的二进制文件。即使是对 Turbo C 文件，你也必须保证如果你为访问文件的一个程序用了“-a”选择项，则所有访问那个文件的程序都应用相同的选择。除非你试图存取用另一编译程序写的的数据，只使用 Turbo C 系统设置值倒是个好主意。你的数据将以那种方式压缩得尽可能小，且你不必记住设置“-a”选择项（这在一个大项目中是件难事）。

5.7 系统级文件 I/O

本节将学习怎样做系统级文件 I/O。这种 I/O 在存储二进制数据时很有用，且由于它直接做 DOS 调用，所以它比标准 I/O 更有效。由于标准 I/O 使用更普遍，我们将只简短描述系统级例行程序。

可用的函数有：

System-Level Functions

| Function | Use |
|----------|---|
| dup | Allows two file handles to point to same open file. The next available file handle is used |

| | |
|-----------|---|
| dup2 | Similar to dup(), except new file handle can be user-specified |
| close | Closes the file |
| _close | DOS-level file close |
| eof | Returns end-of-file status |
| open | A Unix-like file open/create function |
| _open | A DOS specific form of open() |
| creat | A Unix-like file creation function |
| _creat | Similar to creat(), except DOS specific |
| creatnew | Creates a file. It is an error if the file exists |
| creattemp | Creates a file with a Turbo C-created unique filename |
| read | Reads file bytes using the specified text/binary translation mode |
| _read | Direct call to DOS system read. No translation performed |
| lseek | Moves the file pointer to a new position |
| fstat | Returns status of the file (also works for standard I/O files) |
| write | Writes bytes to the file using the specified text/binary translation mode |
| _write | Direct call to DOS system write. No translation is performed |
| stat | Returns status of the file (also works for standard I/O files) |

五个基本函数是 open(), close(), read(), write() 和 lseek(), 这些是我们将集中讨论的。这些例行程序与低级 DOS 函数接口一样。

5.7.1 为随机访问打开文件

基本系统级函数的原型是:

```
int open(char *pathname, int access, [,int permiss]);
int close(int handle);
int read(int handle, void *buf, int nbyte);
int write(int handle, void *buf, int nbyte);
long lseek(int handle, long offset, int origin);
```

当你在 DOS 级打开一个文件时, DOS 返回一个表示该文件“把柄”的整数码, 之后它用于指该文件。如果打开操作有错误, 则返回“-1”。例如:

```
/* System level file I/O (filex5.c) */
#include <fcntl.h> /* contains control flag definitions */
#include <io.h> /* function prototypes */
#include <sys\stat.h> /* needed for permission flag definitions */
#include <stdio.h> /* for printf function prototype */
int fhandle;
char buffer[] = "some data";
main() {
    if ((fhandle=
        open("myfile.dat", O_CREAT|O_WRONLY|O_BINARY)) == -1)
    {
        printf("Error opening file\n");
    }
    else {
        write(fhandle, buffer, sizeof(buffer));
        close(fhandle)
        printf("Data successfully written to myfile.dat\n");
    }
}
```

注意，标志 O-CREAT, O-WRONLY 和 O-BINARY，这些仅是一个可能的标志集里的几个。这些标志可用 OR 联结以控制文件怎样被打开。最常用的是：

| Flag | Use |
|----------|---|
| O_RDONLY | Open file for read only |
| O_WRONLY | Open file for write only |
| O_RDWR | Open file for read/write |
| O_APPEND | Open file for appending |
| O_CREAT | Create the file |
| O_TRUNC | Erase contents of existing file, start over |
| O_BINARY | Raw data mode |
| O_TEXT | Translates CRLF pairs to the newline (0x0a) character |

有了这些标志，你可能已注意到你也能用 open() 创建一个文件或重写一个旧文件。如果你用 O-CREAT 标志，则 open() 函数可从下面的符号常数中取一附加的参数集，它为新文件设置允许：

| | |
|----------|---------------------|
| S_IWRITE | Permission to write |
| S_IREAD | Permission to read |

它们可用 OR 联结以提供系统设置的读/写允许。注意 TurboC 使用手册来告诉你如果你想使用这些常量则须包含头文件 <SYS\stat.h>。

Turbo C 也提供函数 creat(), -creat(), creatnew(), 和 creattemp(), 它也可用于创建文件。详见 Turbo C 使用手册。

5.7.2 读和写系统级文件

Lseek() 函数把文件位置移到理想的地方。除第一个参数是一个文件把柄而非一个 FILE 指针外，它与 fseek() 有完全相同的参数。可以使用相同的偏移起始地址 SEEK-SET, SEEK-CUR, 和 SEEK-END。

一旦你已到达了理想的位置，你就可以用函数 read() 和 write() 把字节移入和移出文件。这两个函数返回实际读/写的字节数，这对错误检查用途是很方便的。例如，下面的程序打开一个文件，移到字节位置 5，并试读入 357 个字节到一缓冲区：

```
/* System level seek example (filex6.c) */
#include <fcntl.h>
#include <io.h>
#include <stdio.h>
char buffer[400];
main() {
    int fh;
    int nb;
    if ((fh = open("myfile.dat", O_RDONLY)) != -1)
        lseek(fh, 5L, SEEK_SET); /* go to posn 5 */
    nb = read(fh, buffer, 357); /* try to read 357 bytes */
    printf("%d bytes read\n", nb); /* probably couldn't */
}
```

```

        close(fh);
    }
    else
        printf("Error opening myfile.dat\n");
}
}

```

注意，如果该文件是以文本方式打开的，则读或写的实际字节数可能与 `read()` 或 `write()` 返回的数目不一致。这是由于可能的换行符翻译所致。TurboC 甚至在 `read()` 和 `write()` 中提供更低级的功能，它们是对 DOS 函数的直接调用并且不执行这样的翻译，即使对文本文件也是这样。

5.8 文件 I/O 软件包例子

本节我们将在标准 I/O 例程序上建立，并创建一个简化某些文件存取细节的软件包。该软件包也提供了方便的错误处理和报告，且是一个使用了许多标准 I/O 例程序的好的例子。它还给出了使用前面章节开发的弹出窗口和错误报告例程序的例子。该软件包在清单 5.1 和 5.2 中给出，且含有下列外部函数：

| Function | Use |
|------------------------------|---|
| <code>init_files()</code> | Initializes the internal file table |
| <code>get_file_data()</code> | Returns information from the internal table |
| <code>openfile()</code> | Opens/creates files |
| <code>closefile()</code> | Closes a file |
| <code>movepos()</code> | Moves the file position |
| <code>iobytes()</code> | Reads/writes to file |
| <code>inbytes()</code> | Macro that calls <code>iobytes</code> to do reads |
| <code>outbytes()</code> | Macro that calls <code>outbytes</code> to do writes |
| <code>rdstr()</code> | Reads in an unformatted string |
| <code>wrtstr()</code> | Writes out an unformatted string |
| <code>rdfstr()</code> | Reads in a formatted string |
| <code>wrtfstr()</code> | Writes out a formatted string |

这个软件包基本上做两件事：它处理发生在文件存取过程中发生的错误的报告，且允许我们方便地做记录 I/O。这是通过保持一个存储流文件指针、文件名、以及每个打开文件的记录大小的内表来实现的。这个表最初由调用 `init-files()` 来初始化。之后，无论何时 `Openfile()` 被调用，新打开的文件的参数就被加入到该表中。调用 `closefile()` 则关闭该文件并把它的信息从该表删除。这些函数有下列原型：

```

void init_files(void);
int openfile(char *fname, char *access_type, int recsize);
int closefile(int h);

```

不混淆这些例程序和 `fopen()` 及 `fclose()` 是很重要的，因为内表不能被正确地更新。

当初始化文件表时，`init-files()` 添加下列预定义的流：

| File/Device | Index | Name | Record Size |
|-------------|-------|----------|-------------|
| stdin | 0 | "stdin" | 1 byte |
| stdout | 1 | "stdout" | 1 byte |
| stderr | 2 | "stderr" | 1 byte |
| stdaux | 3 | "stdaux" | 1 byte |
| stdprn | 4 | "stdprn" | 1 byte |

因为这些流通常与设备相联系，所以它们被给予 1 字节的记录大小，这意味着“字符 / 串级访问”。

用 `openfile()` 打开一个文件后，该文件的记录大小就被指定了。这个值随后就由所有允许到该文件的低级存取的别的文件存取例行程序使用。在确定实际读或写的字节数时记录大小也可用做乘数。

尽管由这个软件包打开和创建的文件是标准 I/O 文件，它们也是用一个整数而非流指针来引用。这个整数是文件表中该文件的索引，并且与 DOS 文件把柄非常相似，但不应混淆。

对包含在该软件包中的所有函数，所执行的文件操作都有错误检查。第四章描述的 `sayerr()` 函数被调用以报告错误。使用 `sayerr()` 意味着你有把错误信息安排到一特殊错误窗口，或允许错误信息弹出的灵活性。包含在错误信息中的文件名，如：

```
"File not found: myfile.dat".
```

`movepos()` 函数用于寻找文件，并且除了位置被解释为记录偏移量及所有错误都报告外，其作用与 `fseek()` 很相似。它的函数原型是：

```
long movepos(int h, long recno, int origin);
```

正如与 `fseek()` 一样，你可指定偏移量起始地址为 `SEEK-SET`，`SEEK-CURS`，或 `SEEK-END`。

对文件的读或写由 `iobytes()` 函数及其相关联的宏 `inbytes()` 和 `outbytes()` 实现。它们被定义如下：

```
#define inbytes(h, r, d, n) iobytes(h, r, d, n, 0)
#define outbytes(h, r, d, n) iobytes(h, r, d, n, 1)
int iobytes(int handle, long recno, unsigned char *data, int numrecs, int iodir);
```

`iobytes()` 函数把寻找、读和写组合到一个函数中。随着文件把柄，你告诉它读 / 写哪一个记录，用于读 / 写的缓冲区，读 / 写的记录数，及 I/O 的方向（也即是读还是写。）`inbytes()` 和 `outbytes()` 宏为你设置 I/O 方向，结果是少一个参数和较少的错误。

如果你给出一个为 -1L 的记录位置，则 `iobytes()` 把它解释为“使用当前记录”；否则，它总是被解释为自该文件开头起的记录偏移量。即使你指定了当前的记录，`iobytes()` 也要进行寻找操作，以便文件缓冲区被正确保持，这在前面已讨论过。

`iobytes()` 函数返回读或写的记录数，或在出错时返回 -1。如果确有错误在 `iobytes()` 操作期间发生，则它不仅报告文件名和所涉及的 DOS 错误，还报告记录位置。

下面是一个早先给出的例子，被重写为使用文件 I/O 软件包了：

```
/*
Record I/O example with file toolkit (filex7.c)
```

```

Must link with: popup.obj, mouse.obj, sayerr.obj,
               fileio.obj
*/
#include <stdio.h>
#include "popup.h" /* Must have this to initialize windows */
#include "fileio.h" /* High level file I/O header */
typedef struct my_parts_struct {
    int part_code;
    int quantity;
    float price;
} part;
main() {
    int f;
    int nr;
    part mypart = {
        15, /* part code */
        300, /* quantity */
        1.25 /* price */
    };
    init_win(); /* Must always do this first */
    init_files(); /* Then remember to initialize the file table */
    /* For record I/O you should always open for binary access,
       (note the "b") */
    if ((f = openfile("myfile.dat", "w+b", sizeof(part))) != -1) {
        /* Write the part out to record 6 */
        nr = outbytes(f, 6, (unsigned char *)&mypart, 1);
        mprintf("The number of records written is %d\r\n", nr);
        closefile(f);
    }
}

```

无论何时使用该文件 I/O 软件包，你必须调用 `init-files()` 先初始化窗口软件包（在打印错误信息的情况下）并随后初始化内部文件表。这应按所示顺序完成。系统设置的表的大小为 20 个文件，但这可用设置在 `fileio.h` 头文件中给出的常量 `MAXFILES`（并重编译 `fileio.c`）来改变。这个常量不应与 `config.sys` 中的 `FILES` 参数混淆。那个参数是 DOS 用来为它自己的文件表分配空间的。两者使用同一个数可能是个好主意。

注意前一个例子中有错误检查，所以在未打开的文件上不会有误作。但是 `openfile()` 函数为我们处理所有错误报告。该文件软件包前后一致地用 -1 表示错误指示，Turbo C 文件例行程序使用同样的值。

函数 `rdstr()`、`wrtstr()`、`rdfstr()` 和 `wrtfstr()` 被提供用于存取字符和串级。它们分别调用 TurboC 例行程序 `fgets()`、`fputs()`、`vfscanf()` 和 `vfprintf()`，并且既允许非格式化的串存取又允许格式化的串存取。它们的类型原型是：

```

int rdstr(char *str, int n, int h);
int wrtstr(char *str, int h);
int rdfstr(int h, char *format, ...);
int wrtfstr(int h, char *format, ...);

```

当使用这些例行程序时，你应该以一字节的记录大小打开文件以指明字符串级的存取。另外，如果它真是一个文本文件，一定要在文件存取模式参数中用“t”标识符。与其 Turbo C 对等部分不同，这例行程序由于与 `iobytes()` 同样的原因先做一个到当前位置的查寻。

`rdstr()` 函数成功则返回 0，出错则返回 -1。象它的 `fputs()` 对等部分一样，返回最后读的字符，或出错时返回 -1。象它的 `vfscanf()` 对等部分一样，`rdstr()` 返回扫描的域的个数，或出错时返回 -1。`wrtfstr()` 函数返回所写的字节数，或在出错时返回 -1，它也与其 `vfprintf()` 对等部分相似。

下例中，一个文件被创建且一个格式化的信息被打印给它。

```
/*
    Formatted file I/O using file toolkit (filex8.
    Must link with: popup.obj, mouse.obj, sayerr.obj, fileio.obj
*/

#include <stdio.h>
#include "popup.h" /* Must have this to initialize windows */
#include "fileio.h" /* High level file I/O header */
main() {
    int f, nb, age = 30;

    init_win(); /* Must always do this first */
    init_files(); /* Then remember to initialize the file table */
    /* Note that we are opening for text file access */
    if ((f = openfile("myfile.dat", "wt", sizeof(char))) != -1) {
        nb = wrtfstr(f, "Sally is %d years old\n", age);
        mprintf("The number of characters written is %d\r\n", nb);
        closefile(f);
    }
}
```

Listing 5.1 Source code for the header file "fileio.h"

```
/* File I/O toolkit header file (fileio.h) */
#define inbytes(port, rec, d, nr) iobytes(port, rec, d, nr, 0)
#define outbytes(port, rec, d, nr) iobytes(port, rec, d, nr, 1)
#define MAXFILES 20
typedef struct fh_struct {
    FILE *fp;
    char *name;
    int recsize;
} file_entry;
extern file_entry ft[]; /* our record and error reporting file table */
extern void init_files(void);
extern int get_file_data(int h, FILE **f, char **fname, int *recsize);
extern int openfile(char *fname, char *access_type, int recsize);
extern int closefile(int h);
```

```

extern long movepos(int h, long recno, int smode);
extern int iobytes(int h, long recno, unsigned char *d, int nr, int iodir);
extern int rdstr(char *str, int n, int h);
extern int wrtstr(char *str, int h);
extern int rdfstr(int h, char *format, ...);
extern int wrtfstr(int h, char *format, ...);

```

Listing 5.2 Source code for the file "fileio.c"

```

/*****
 * File I/O toolkit (fileio.c)
 *****/
#include <string.h>
#include <alloc.h>
#include <stdio.h>
#include "popup.h"
#include "fileio.h"
file_entry ft[MAXFILES]; /* our record and error reporting file table */
static char unknownfile[] = "unknown file"; /* use in error messages */
static char iowstr[] = "writing";
static char iorstr[] = "reading";
/* internal functions */
static int add_file(FILE *f, char *n, int recsize);
static void rmv_file(int h);
static int valid_index(int h);

void init_files(void)
/* Initializes file table for high level file i/o. */
{
    int h;
    for (h=0; h<MAXFILES; h++) {
        ft[h].fp = NULL; /* initialize file entry */
        ft[h].name = NULL;
    }
    add_file(stdin, "stdin", 1); /* set up default file handles */
    add_file(stdout, "stdout", 1); /* for character level access */
    add_file(stderr, "stderr", 1);
    add_file(stdaux, "stdaux", 1);
    add_file(stdprn, "stdprn", 1);
}

static int add_file(FILE *f, char *n, int recsize)
/* Searches for an open entry in the file table and stores
the FILE pointer along with the file name and record size.
Reports an error if not enough room in the table, and returns -1.
If no error, the position in the file table is returned.
*/
{
    int h;
    /* search for open entry */
    for (h=0; (h<MAXFILES) && (ft[h].fp != NULL); h++);
    if (h == MAXFILES) {
        sayerr(SERR, "File table full\r\n");
        return -1;
    }
    ft[h].fp = f; /* set FILE pointer */
    ft[h].name = strdup(n); /* allocate and copy file name */
    ft[h].recsize = recsize; /* set the record size */
    return h;
}

```

```

}
static void rmv_file(int h)
/* Removes entry h from the file table by setting the file handle
   to NULL, and freeing up the space allocated for the name. The
   recsize is also set to zero.
*/
{
    ft[h].fp = NULL;
    free(ft[h].name);
    ft[h].recsize = 0;
}
static int valid_index(int h)
/* Returns 1 if h points to an existing file entry, else returns 0. */
{
    if ((h >= 0) && (h < MAXFILES) && (ft[h].fp != NULL)) {
        return 1;
    }
    else {
        sayerr(ERR, "Illegal file index %d\r\n", h);
        return 0;
    }
}
int get_file_data(int h, FILE **f, char **fname, int *recsize)
/* Sees if handle h is an opened file. If so, the FILE pointer,
   the file name, and record size are returned. If the entry is
   empty, the file pointer is set to NULL, the file name is set to
   "unknown", and the recsize to zero. A -1 is returned as an
   error indicator, else a zero is.
*/
{
    if (!valid_index(h)) return -1;
    if ((*f = ft[h].fp) == NULL) {
        *fname = unknownfile;
        *recsize = 0;
        return -1;
    }
    else {
        *fname = ft[h].name;
        *recsize = ft[h].recsize;
        return 0;
    }
}
int openfile(char *fname, char *access_type, int recsize)
/*
   Opens/creates a file with given access type and record size.
   The FILE pointer, filename, and record size are stored internally
   for convenient error handling and record I/O purposes.
   Returns the new table entry position, or -1 if an error occurs.
*/
{
    FILE *f;
    if ((f = fopen(fname, access_type)) == NULL) {
        sayerr(ERR, "%s\r\n", fname);
        return -1;
    }
    else {
        return add_file(f, fname, recsize);
    }
}

```

```

    }
    int closefile(int h)
    /*
       Closes file h (as indexed by the file table), and removes it from
       the file table. Returns -1 if an error occurred, else 0.

       If h is out of bounds, an error is reported, and a -1 is returned.
    */
    {
        if (!valid_index(h)) return -1;
        if (fclose(ft[h].fp) == -1) {
            sayerr(FERR, "%d - %s\r\n", ft[h].fp, ft[h].name);
            return -1;
        }
        rmv_file(h);
        return 0;
    }
    long movepos(int h, long recno, int smode)
    /*
       Moves the file pointer to the record offset recno, using seek mode
       smode, (which should be either SEEK_SET, SEEK_CUR, or SEEK_END).
       Returns -1 and reports if there's an error, else it returns
       the new record position.
       The recno is multiplied by the record size to get the true byte
       position in the file, (ignoring any newline translations in text
       mode).
    */
    {
        long newpos;
        if (!valid_index(h)) return -1;
        newpos = recno * ft[h].recsize;
        if ((newpos = fseek(ft[h].fp, newpos, smode)) == -1) {
            sayerr(FERR, "file: %s\r\nrecord %ld\r\n", ft[h].name, recno);
            return -1;
        }
        return newpos / ft[h].recsize;
    }
    int iobytes(int h, long recno, unsigned char *d, int nr, int iodir)
    /*
       Reads/writes nr records in/out of buffer d at record recno of file h.
       if iodir = 1, it does a write, else, it does a read.
       If recno = -1, it means to read/write from the current position.
       Returns -1 if error, else number of bytes actually moved.
       The recno offset is always from the beginning of the file, (unless
       its -1, which means current position).
    */
    {
        unsigned int recsmoved;
        char *iodirstr;
        long savepos;
        if (!valid_index(h)) return -1;
        /* Always do a seek, so that fread() and fwrite() work properly. */
        /* Either seek to current record, or to specified one. */
        if (recno == -1) {
            if (movepos(h, 0L, SEEK_CUR) == -1L) return -1;
        }
    }

```

```

else {
    if (movepos(h, recno, SEEK_SET) == -1L) return -1;
}
/*
    Use the following as a part of a work around for a bug in
    some older versions of Turbo C. Must save the current
    position so we can use it as a reference.
    savepos = ftell(ft[h].fp);
*/
if (iodir == 1) {
    recsmoved = fwrite(d, ft[h].recsize, nr, ft[h].fp);
}
else {
    recsmoved = fread(d, ft[h].recsize, nr, ft[h].fp);
}
if (recsmoved == -1) {
    if (iodir == 1) iodistr = iowstr; else iodistr = iorstr;
    sayerr(FERR, "%s %s\r\npos'n %ld\r\n", iodistr, ft[h].name, recno);
    return -1;
}
else {
    /*
        Use the following as a work around for a bug in some older
        versions of Turbo C. Must move the stream pointer to
        proper position.
        savepos += recsmoved * ft[h].recsize;
        if (movepos(h, savepos, SEEK_SET) == -1L) return -1;
    */
}
return recsmoved;
}

int rdstr(char *str, int n, int h)
/* Reads a string from file h by calling fgets(). Unlike fgets()
   it does a seek to the current position first to properly
   maintain the file buffer. Returns 0 if successful, otherwise a -1.
*/
{
    if (!valid_index(h)) return -1;
    /* Seek to current position so buffer is maintained properly.
       NOTE: For some versions of Turbo C 1.5 and older this does
       not work correctly. Contact Borland for update.
    */
    if (movepos(h, 0, SEEK_CUR) == -1L) return -1;
    if (fgets(str, n, ft[h].fp) == NULL) return -1;
    return 0; /* successful */
}

int wrtstr(char *str, int h)
/* Writes a string to file h by calling fputs(). Unlike fputs()

```

```

    it does a seek to the current position first to properly
    maintain the file buffer. Returns the last character written
    if successful, otherwise a -1.
    */
    (
    int c;
    if (!valid_index(h)) return -1;
    /* Seek to current position so buffer is maintained properly.
       NOTE: For some versions of Turbo C 1.5 and older this does
       not work correctly. Contact Borland for update.
    */
    if (movepos(h, 0, SEEK_CUR) == -1L) return -1;
    if ((c = fputc(str, ft[h].fp)) == EOF) return -1;

    return c; /* successful */
    )
int rdfsfr(int h, char *format, ...)
/* Reads a formatted string from file h at current posn.
 * If an error occurs a -1 is returned, else the number of fields
 * scanned is returned.
 */
(
    va_list arg_ptr;
    int e;
    if (!valid_index(h)) return -1;
    /* Seek to current position so buffer is maintained properly.
       NOTE: For some versions of Turbo C 1.5 and older this does
       not work correctly. Contact Borland for update.
    */
    if (movepos(h, 0, SEEK_CUR) == -1L) return -1;
    va_start(arg_ptr, format);
    e = vscanf(ft[h].fp, format, arg_ptr);
    va_end(arg_ptr);
    if (ferror(ft[h].fp) || feof(ft[h].fp)) {
        sayerr(FERR, "scanning %s\r\n", ft[h].name);
        clearerr(ft[h].fp);
        return -1;
    }
    return e;
)
int wrfsfr(int h, char *format, ...)
/* Writes a formatted string from file h at current position.
 * If an error occurs a -1 is returned, else the number of bytes
 * printed is returned.
 */
(
    va_list arg_ptr;
    int e;
    if (!valid_index(h)) return -1;

```



```

/* Seek to current position so buffer is maintained properly.
   NOTE: For some versions of Turbo C 1.5 and older this does
   not work correctly. Contact Borland for update.
*/

```

```

if (move_pos(h, 0, SEEK_CUR) == -1L) return -1;
va_start(arg_ptr, format);
e = vfprintf(ft[h].fp, format, arg_ptr);
va_end(arg_ptr);
if (ferror(ft[h].fp) || feof(ft[h].fp)) {
    sayerr(FERR, "printing to %s\r\n", ft[h].name);
    clearerr(ft[h].fp);
    return -1;
}
return e;
}

```

第六章 串函数库

本章给出两个含有附加的串函数的库。这些函数增补 Turbo C 串函数的通用表。给出两个有些类似的函数的原因是用指针或指针与字符索引的组合操作串的能力。在前一种情况下,所用指针很可能是单独说明的存取子串的指针。第二种情况下,所用指针很可能是串标识符本身,它指向索引 0。字符索引提供存取任意串字符的偏移量。在两种方法之间的转换是很容易的。给定串标识符和偏移量,应为得到指针地址,你可使用。

```
ptr = str + index;
```

给定指针和串标识符名,为得到字符索引,你可用。

```
index = (unsigned int) (ptr - str);
```

6.1 库 strops1.c

清单 6.1 和 6.2 显示了含有与 Turbo C 串函数相同的途径写成的例行程序的第一个串库。这些串假定串指针含有有效串位置的地址。

第一个函数是 pt-insert, 它把一个子串插入一个串。它被说明为:

```
void pt_insert(char* ptr, char* substr)
```

串指针参数含有子串插入处串字符的地址。它可被认为是虚串的基地址。假定一个有效串指针被使用,则操作可以因此被认为是在虚串的开头插入子串。这减少了程序把一串字符移到较高的索引而随后拷贝所插入的子串的工作。两次调用 memmove 用于很快完成插入工作。为说明 pt-insert 怎样工作,可考虑下面的程序:

```
/* program to demonstrate string insertion
   using pt_insert() in library "strops1.h" */
#include <stdio.h>
#include <string.h>
#include <strops1.h>
#include "conio.h"
main()
{
    char str1[41], str2[41];
    int index, len;
    clrscr();
    printf("          01234567890123456789012345678901234567890\n");
    printf("          1          2          3          4\n");
    printf("Enter a string -> ");
    gets(str1);
    printf("\nEnter string to be inserted -> ");
    gets(str2);
    len = strlen(str1);
    do {
        printf("\nEnter location : ");
```

```

        scanf("%d", &index);
        printf("\n");
    } while (index >= len);
    pt_insert(str1 + index, str2);
    printf("\nThe string is now '%s'", str1);
}

```

pt-overwrite 函数用子串覆盖一个串。如果覆盖子串超过了串定界符，则串被扩展到能容纳多余的字符。函数说明如下

```
void pt_overwrite(char* ptr, char* substr)
```

使用到该串的指针，则问题可减少到用一个子串覆盖虚串。为说明 pt-overwrite 怎样工作，可考虑下列程序：

```

/* program to demonstrate overwriting string characters
   using pt_overwrite() in library "stropsl.h" */
#include <stdio.h>
#include <string.h>
#include <stropsl.h>
#include "conio.h"
main()
{
    char str1[41], str2[41];
    int index, len;
    clrscr();
    printf("          01234567890123456789012345678901234567890\n");
    printf("          1          2          3          4\n");
    printf("Enter a string -> ");
    gets(str1);
    printf("\nEnter overwriting string -> ");
    gets(str2);
    len = strlen(str1);
    do {
        printf("\nEnter location : ");
        scanf("%d", &index);
        printf("\n");
    } while (index >= len);
    pt_overwrite(str1 + index, str2);
    printf("\nThe string is now '%s'", str1);
}

```

pt-delete 函数删除始于串指针所指的字符的指定数目的字符。函数被说明为：

```
void pt_delete(char* ptr, unsigned int count)
```

如果被删除的字符的计数器比虚串长度大或相同，则串就简单地被节取否则。否则从该串尾部的字符被拷贝来覆盖被删除的部分。为说明 pt-delete 怎样工作，可考虑下面的程序：

```

/* program to demonstrate deleting string characters
   using pt_delete() in library "stropsl.h" */
#include <stdio.h>
#include <string.h>
#include <stropsl.h>
#include "conio.h"

```

```

main()
{
    char str1[41];
    int count, index, len;
    clrscr();
    printf("          0123456789012345678901234567890\n");
    printf("          1          2          3          4\n");
    printf("Enter a string -> ");
    gets(str1);
    printf("\nEnter the number of characters to delete : ");
    scanf("%d", &count);
    len = strlen(str1);
    do {
        printf("\nEnter location : ");
        scanf("%d", &index);
        printf("\n");
    } while (index >= len);
    pt_delete(str1 + index, count);
    printf("\nThe string is now '%s'", str1);
}

```

pt-replace-str 函数扫描一个串并用另一串替换指定的子串。函数被说明为:

```

int pt_replace_str(char* ptr,
                  char* find,
                  char* replace,
                  unsigned int freq)

```

文本翻译的频率的最大值也被指定。串指针用于指示串翻译从哪儿开始。在多数情况下, 整个串都被扫描, 并且因此串标识符被用做串指针参数。这个 pt-replace-str 调用前面讨论过的 pt-insert 和 pt-delete 函数, 也调用 Turbo C strstr 串函数。主串和匹配串中有一个或两个都是空串时函数返回-1。为说明 pt-replace-str 怎样工作, 可考虑下列程序:

```

/* program to demonstrate translating string characters
   using pt_replace_str() in library "strops1.h" */
#include <stdio.h>
#include <string.h>
#include <strops1.h>
#include "conio.h"
main()
{
    char str1[41], find[41], replace[41];
    int count = 1;
    clrscr();
    printf("Enter a string -> ");
    gets(str1);
    printf("\nEnter the string to find -> ");
    gets(find);

```

```

printf("\nEnter the string to replace -> ");
gets(replace);
pt_replace_str(str1, find, replace, count);
printf("\nThe string is now 'is'", str1);
}

```

repeat-str 用于以模式串重复地连接成串来创建一个串，函数被说明如下：

```
void repeat_str(char* str, char* pattern, int num)
```

下面的清单含有一个说明使用前面讲过的函数的串插入，删除，和翻译的简短程序：

```

#include <stdio.h>
#include <string.h>
#include "strops1.h"
main()
{
    char str[81] = "Ada Byron";
    void pt_insert(char*, char*);
    void pt_delete(char*, unsigned int);
    puts("0123456789|123456789|123456789|");
    puts(str);
    pt_insert(str+3, " Augusta");
    puts(str);
    pt_replace_str(str, "Ada", "Lady", 1);
    puts(str);
    pt_delete(str+5, strlen("Augusta "));
    puts(str);
}

```

运行该程序，屏幕上将出现下列文本：

```

0123456789|123456789|123456789|
Ada Byron
Ada Augusta Byron
Lady Augusta Byron
Lady Byron

```

6.2 库 strops2.C

第二个串库显示在清单 6.3 和 6.4 中。它含有更多的串函数，它们全都使用串字符索引。这意味着该库中的函数有一个应检查有效值的附加的参数。

第一个串函数是 pos-str，它从给定的字符索引开始，在一个串中搜索一个子串的第一个位置。该函数被说明为：

```
int pos_str(char* str, char* substr, unsigned int start_index)
```

如果未找到匹配串则返回-1。除 strlen 外，该函数的编程不依赖于任何别的串函数。既可用近指针也可用长指针使用该函数。为说明 pos-str 怎样工作，考虑下列程序：

```
/* program to demonstrate locating substrings
```

```

    using pos_str() in library "strops2.h" */
#include <stdio.h>
#include <string.h>
#include <strops2.h>
#include "conio.h"
main()
{
    char str1[41], find[41];
    int start, len;
    clrscr();
    printf("          01234567890123456789012345678901234567890\n");
    printf("          1          2          3          4\n");
    printf("Enter a string -> ");
    gets(str1);
    printf("\nEnter the string to find -> ");
    gets(find);
    len = strlen(str1);
    do {
        printf("\nEnter number of characters to skip : ");
        scanf("%d", &start);
    } while (start < 1 || start >= len);
    printf("\nThe substring matches at char %d",
        pos_str(str1, find, start));
}

```

函数 Pos-str2 执行与 pos-str 同样的任务，但使用 Turbo C 库中的 strstr 串函数。该函数被说明为：
 int pos_str2(char* str, char* substr, unsigned int start_index)

strstr 函数返回指向匹配串的位置的指针（或如果未找到则返回 NULL）。该函数校验是否返回了一个 NULL 指针，如果不是，则返回非 NULL 指针地址与串参数间的差。

pos-char 函数从一个指定的索引处开始扫描一个串，寻找一个指定字符的首次出现。该函数被说明为：
 int pos_char(char* str, char ch, int start_index)

该函数的中心是一个反复重复直到字符被找到或到达串尾为止的 for 循环。如果未找到匹配，则返回-1。为说明 pos-char 怎样工作，可考虑下列程序：

```

/* program to demonstrate locating characters
   using pos_char() in library "strops2.h" */
#include <stdio.h>
#include <string.h>
#include <strops2.h>
#include "conio.h"
main()
{
    char str1[41], find;
    int start, len;

```

```

clrscr();
printf("                01234567890123456789012345678901234567890\n");
printf("                1          2          3          4\n");
printf("Enter a string -> ");
gets(str1);
printf("\nEnter the character to find -> ");
find = getche();
len = strlen(str1);
printf("\n");
do {
    printf("\nEnter number of characters to skip : ");
    scanf("%d", &start);
} while (start < 1 || start >= len);
printf("\nThe character matches at string char %d",
    pos_har(str1, find, start));
}

```

in-insert 函数在一个串中的指定位置插入一个子串。与 pt-insert 函数相比，in-insert 有一个名为插入索引的附加的参数。该函数被说明为：

```
void in_insert(char* str, char* subetr, unsigned int index)
```

该函数处理两类插入情况：

1.索引值对应一个有效串字符（即，在空串定界符之前）。子串插入到索引所指向的地方。

2.索引值指向空串定界符或任何超过它的字符。在这种情况下，索引值被忽略，并且子串只是附在该串尾部。为说明 in-insert 如何工作，考虑下面的程序：

```

/* program to demonstrate string insertion
   using in_insert() in library "strops2.h" */
#include <stdio.h>
#include <string.h>
#include <strops2.h>
#include "conio.h"
main()
{
    char str1[41], str2[41];
    int index, len;
    clrscr();
    printf("                01234567890123456789012345678901234567890\n");
    printf("                1          2          3          4\n");
    printf("Enter a string -> ");
    gets(str1);
    printf("\nEnter string to be inserted -> ");
    gets(str2);
    len = strlen(str1);
    do {
        printf("\nEnter location : ");
        scanf("%d", &index);
        printf("\n");
    }
}

```

```

    } while (index >= len);
    in_insert(str1, str2, index);
    printf("\nThe string is now '%s'", str1);
}

```

in-overwrite 函数在一指定的字符位置用另一个串覆盖一个串。该函数如下说明为:

```
void in_overwrite(char* str, char* substr, unsigned int index)
```

用于覆盖第一个串的第二个串可能会引起其大小的扩展。这是由于相对串长和 / 或覆盖位置引起的。如果指定的字符索引的位置在串定界符之外, 则第二个串就只是附于第一个的后面。为说明 in-overwrite 怎样起作用, 可考虑下面的程序:

```

/* program to demonstrate overwriting string characters
   using in_overwrite() in library "strops2.h" */
#include <stdio.h>
#include <string.h>
#include <strops2.h>
#include "conio.h"
main()
{
    char str1[41], str2[41];
    int index, len;
    clrscr();
    printf("          0123456789012345678901234567890\n");
    printf("          1          2          3          4\n");
    printf("Enter a string -> ");
    gets(str1);
    printf("\nEnter overwriting string -> ");
    gets(str2);
    len = strlen(str1);
    do {
        printf("\nEnter location : ");
        scanf("%d", &index);
        printf("\n");
    } while (index >= len);
    in_overwrite(str1, str2, index);
    printf("\nThe string is now '%s'", str1);
}

```

in-delete 函数删除部分或全部串中的字符。该函数被说明为:

```
int in_delete(char* str, unsigned int index, unsigned int count)
```

如果该字符在串定界符之外, 则无字符被删除。如果计数器参数的值大于即将删除的字符的可得的数目, 则间隙被忽略。使用 in-delete 的例子有:

```

/* program to demonstrate deleting string characters
   using in_delete() in library "strops2.h" */
#include <stdio.h>
#include <string.h>
#include <strops2.h>

```



```

#include "conio.h"
main()
{
    char str1[41];
    int count, index, len;
    clrscr();
    printf("01234567890123456789012345678901234567890\n");
    printf("1 2 3 4\n");
    printf("Enter a string -> ");
    gets(str1);
    printf("\nEnter the number of characters to delete : ");
    scanf("%d", &count);
    len = strlen(str1);
    do {
        printf("\nEnter location : ");
        scanf("%d", &index);
        printf("\n");
    } while (index >= len);
    in_delete(str1, index, count);
    printf("\nThe string is now '%s'", str1);
}

```

in-replace-str 函数专用于把子串翻译为一个较大的串。该函数被说明为：

```

int in_replace_str(char* str,
                  char* find,
                  char* replace,
                  unsigned int start_index,
                  unsigned int freq)

```

start-index 参数指定翻译开始处的字符索引（并且也等于即将越过的字符数减 1）。freq 参数定义允许的翻译的最大数目。这两个参数给你仔细调整文本翻译过程的控制权。下面的短程序说明了 in-replace-str 函数的用法：

```

/* program to demonstrate translating string characters
   using in_replace_str() in library "strops2.h" */
#include <stdio.h>
#include <string.h>
#include <strops2.h>
#include "conio.h"
main()
{
    char str1[41], find[41], replace[41];
    int start = 1, count = 1;
    clrscr();
    printf("Enter a string -> ");
    gets(str1);
    printf("\nEnter the string to find -> ");
    gets(find);
    printf("\nEnter the string to replace -> ");
}

```

```

        gets(replace);
        in_replace_str(str1, find, replace, start, count);
        printf("\nThe string is now '%s'", str1);
    }

```

in-pos-mid 是一个通过指定第一个和最后一个字符索引来抽取一个函数的函数。该函数被说明为：
`int in_pos_mid(char* str, unsigned int first, unsigned int last)`

当第一个参数指向超出空串终止符以外的一个字符时，串参数不变返回。当最后一个参数指向一个超出空串终止符以外的字符时，则串在删除其原来的开头字符后返回。在串长内的第一个和最后一个参数的值应除掉串的左右两边的部分。下面的程序说明这个函数怎样工作：

```

/* program to demonstrate extracting string characters
   using in_pos_mid() in library "strops2.h" */
#include <stdio.h>
#include <string.h>
#include <strops2.h>
#include "conio.h"
main()
{
    char str1[41];
    int first, last;
    clrscr();
    printf("          01234567890123456789012345678901234567890\n");
    printf("          1          2          3          4\n");
    printf("Enter a string -> ");
    gets(str1);
    do {
        printf("\nEnter first character to extract : ");
        scanf("%d", &first);
        printf("\nEnter last character to extract : ");
        scanf("%d", &last);
    } while (first >= last);
    in_pos_mid(str1, first, last);
    printf("\nThe string is now '%s'", str1);
}

```

in-count-mid 函数是 in-pos-mid 的姊妹函数。差别是最后一个参数提供一个抽取的字符数目计数器。该函数说明如下：

`int in_count_mid(char* str, unsigned int index, unsigned int count)`

当第一个参数指向空串终止符以外的字符时，串参数不变返回。当计数器参数大于可得字符实际数目时，串在删除了其原来的开头字符之后返回。在串长以内的第一个和计数器参数的值应除去该串左右两边的部分。下面的程序说明了这个函数怎样操作：

```

/* program to demonstrate extracting string characters
   using in_count_mid() in library "strops2.h" */
#include <stdio.h>
#include <string.h>

```

```

#include <strops2.h>
#include "conio.h"
main()
{
    char str1[41];
    int first, count;
    clrscr();
    printf("          01234567890123456789012345678901234567890\n");
    printf("          1          2          3          4\n");
    printf("Enter a string -> ");
    gets(str1);
    do {
        printf("\nEnter first character to extract : ");
        scanf("%d", &first);
        printf("\nEnter the number of characters to extract : ");
        scanf("%d", &count);
    } while (first < 0 || count <= 0);
    in_count_mid(str1, first, count);
    printf("\nThe string is now '%s'", str1);
}

```

strops2.C 含有下面三个串填充函数:

```

void pad_left(char* str, char pad_char, int num_chars)
void pad_right(char* str, char pad_char, int num_chars)
void pad_ends(char* str, char pad_char, int each_end)

```

这些函数填充一个字符的指定数目的拷贝到一个串的一端或两端。

与串填充函数相对的是下面的串修剪函数:

```

void trim_left(char* str, char ch)
void trim_right(char* str, char ch)
void trim_ends(char* str, char ch)

```

这些函数以你能指定即将删除的字符这一事实为特征。你未被限制到只能删除空格。下面的短程序是使用填充和删除函数的一个例子:

```

#include <stdio.h>
#include "strops2.h"
main()
{
    char str[81] = "0123456789";
    char c = '-';
    void pad_ends(char*, char, int);
    void trim_ends(char*, char);

    pad_ends(str, c, 3);
    puts(str); /* displays "--0123456789--" */
    trim_ends(str, c);
    puts(str); /* displays "0123456789" */
}

```

6.3 一个应用：基本文本文件翻译器

清单 6.5 含有“translan、C”的源代码，这个程序是一个执行基本文本文件翻译的应用。虽然这个应用执行一个直截了当的翻译，但它是由稿本文件驱动的。稿本文件给该应用提供准确的文本操作指令。一个稿本文件是一个含有一系列可能是下列之一的命令的文本文件：

1.对情形敏感的文本删除 它使用语法“D① <text> ①”，其中杠符号把即将删除的文本模式包起来。

2.对情形不敏感的文本删除 它使用语法“K① <text> ①”，其中杠符号把即将删除的文本模式包起来。

3.对情形敏感的文本翻译 它使用语法“R① <old text> ① <new text> ①”用 <new text> 替换 <old text>。

清单 6.6 含有用于 Turbo Pascal 程序到 Turbo C 的基本翻译的稿本文件 pas2c.src 的源程序。写了 translan.c 以便 pas2c.src 文件成为系统设置的文件。你可以向 pas2c.src 添加或删除个别稿本指令。如果你在别的语言之间翻译，则你需写自己的稿本文件。translan.c 实用程序的效力就是其编程能力，你可以向 translan.c 文件添加新的稿本命令以完成涉及到文本模式的更高级的文本翻译。

为说明 translan.c 程序，把清单 6.7 中所示的 Turbo Pascal 筛选标准程序用作输入文件。输出文件在清单 6.8 中给出并含有一个具有 Pascal 和 C 语法的源程序文件！手编的工作文件显示在清单 6.9 中。这将给你一个从中间状态到最终形式的转换所执行的工作量的概念。

Listing 6.1 Source code for header file "strops1.h"

```
void pt_insert(char*, char*);
void pt_overwrite(char*, char*);
void pt_delete(char*, unsigned int);
int pt_replace_str(char*, char*, char*, unsigned int);
void repeat_str(char*, char*, int);
```

Listing 6.2 Source code for library file "strops1.c"

```
#include <string.h>
void pt_insert(char* ptr, char* substr)
/* insert substring */
{
    unsigned int m = strlen(substr);
    memmove((ptr+m), ptr, strlen(ptr)+1); /* move character out */
    memmove(ptr, substr, m);               /* copy substring */
}
void pt_overwrite(char* ptr, char* substr)
{
    unsigned int m = strlen(substr)-1;
    unsigned int n = strlen(ptr);
    unsigned int shift;
    shift = (n < m) ? m : (m+1);
    memmove(ptr, substr, shift);
```

```

}
void pt_delete(char* ptr, unsigned int count)
{
    unsigned int n = strlen(ptr);
    if (count >= n)
        *ptr = '\0'; /* simply truncate string pointer */
    else /* move characters to overwrite deleted substring */
        memmove(ptr, (ptr+count), n-count+1);
}

int pt_replace_str(char* ptr,
                  char* find,
                  char* replace,
                  unsigned int freq)
{
    unsigned int findlen = strlen(find);
    unsigned int repl_strlen = strlen(replace);
    unsigned int strlen = strlen(ptr);
    char* match_ptr;
    /*-----Argument-checking-----*/
    if ((findlen * strlen) == 0)
        return -1;
    match_ptr = strstr(ptr, find);
    while ((match_ptr != 0) && (freq > 0)) {
        freq--;
        /* remove string found */
        pt_delete(match_ptr, findlen);
        /* replace it with new string */
        if (repl_strlen > 0)
            pt_insert(match_ptr, replace);
        /* find next matching strings */
        match_ptr = strstr(ptr, find);
    }
    return 0;
}

void repeat_str(char* str, char* pattern, int num)
{
    while (num-- > 0)
        strcat(str, pattern);
}

```

Listing 6.3 Source code for header file "strops2.h"

```

int pos_str(char*, char*, unsigned int);
int pos_str2(char*, char*, unsigned int);
int pos_char(char*, char, int);
void in_insert(char*, char*, unsigned int);
void in_overwrite(char*, char*, unsigned int);
int in_delete(char*, unsigned int, unsigned int);
int in_replace_str(char*, char*, char*, unsigned int, unsigned int);
int in_pos_mid(char*, unsigned int, unsigned int);
int in_count_mid(char*, unsigned int, unsigned int);
void pad_left(char*, char, int);
void pad_right(char*, char, int);
void trim_left(char*, char);
void trim_right(char*, char);
void trim_ends(char*, char);

```

Listing 6.4 Source code for string library "strops2.c"

```
#include <string.h>
int pos_str(char* str, char* substr, unsigned int start_index)
{
    int i, j, k, last;
    unsigned int sstrlen = strlen(str);
    unsigned int substrlen = strlen(substr);
    unsigned char nomatch;
    if ((substrlen == 0) || (start_index >= sstrlen))
        return -1;
    k = -1;
    if (sstrlen > substrlen) {
        i = start_index - 1;
        last = sstrlen - substrlen;
        nomatch = 1;
        while ((i <= last) && (nomatch == 1)) {
            i++;
            if (substr[0] == str[i]) {
                k = i;
                j = 1;
                i++;
                nomatch = 0;
                while ((j < substrlen) && (nomatch == 0) )
                    if (substr[j] == str[i]) {
                        i++;
                        j++;
                    }
                else
                    nomatch = 1;
                /* restore index before complete matching was attempted */
                if (nomatch == 1) {
                    i = k + 1;
                    k = -1;
                }
            } /* if (substr[0] == str[i]) */
        } /* while ((i <= last) && (nomatch)) */
    } /* if (sstrlen > substrlen) */
    return k;
}

int pos_str2(char* str, char* substr, unsigned int start_index)
{
    char* ptr;
    ptr = strstr((str+start_index), substr);
    if (ptr != 0)
        return (ptr - str);
    else
        return -1;
}

int pos_char(char* str, char ch, int start_index)
{
    int k;
    unsigned int n = strlen(str);
    for (k = start_index; ((*(str+k) != ch) && (k < n)); k++);
    if (k == n)
        k = -1;
}
```

```

        return k;
    }
    void in_insert(char* str, char* substr, unsigned int index)
    /* insert or append substring */
    {
        unsigned int m = strlen(substr);
        unsigned int n = strlen(str);
        if (index < n) { /* insert substring */
            int i;
            memmove((str+index+m), (str+index), n+1-index);
            /* *(str+n+m) = '\0'; */
            for (i = n+1-index; i >= 0; i--)
                *(str+index+m+i) = *(str+index+i); /*
            }
            memmove((str+index), substr, m);
        }
        else /* append substring */
            memmove((str+n), substr, m+1);
    }
    void in_overwrite(char* str, char* substr, unsigned int index)
    {
        unsigned int m = strlen(substr);
        unsigned int n = strlen(str);
        unsigned int first, shift;
        if (index < n) { /* overwrite substring */
            first = index; /* substring index in inside string */
            shift = m; /* assume substring does not go beyond
                        end-of-string */
            if ((index+m) > n)
                shift++; /* add one more char count for \0 */
        }
        else { /* append substring */
            first = n;
            shift = m + 1;
        }
        memmove((str+first), substr, shift);
    }
    int in_delete(char* str, unsigned int index, unsigned int count)
    {
        unsigned int n = strlen(str);
        if (index < 0) index = 0;
        if (index < n) {
            if ((index+count-1) >= n) /* truncate string */
                str[index] = '\0';
            else /* move characters to overwrite deleted substring */
                memmove((str+index), (str+index+count), n-index-count+1);
            return 0; /* no-error code */
        }
        else
            return -1; /* error code */
    }
    int in_replace_str(char* str,
                      char* find,
                      char* replace,
                      unsigned int start_index,
                      unsigned int freq)
    {

```

```

unsigned int findlen = strlen(find);
unsigned int repl_strlen = strlen(replace);
unsigned int sstrlen = strlen(str);
int match_pos;
/*-----Argument-checking-----*/
if ( ((findlen * sstrlen) == 0) || (start_index >= sstrlen) )
    return -1;
match_pos = pos_str(str, find, 0);
while (match_pos > -1 && freq > 0) {
    freq--;
    /* remove string found */
    in_delete(str, (unsigned int) match_pos, findlen);
    /* replace it with new string */
    if (repl_strlen > 0)
        in_insert(str, replace, (unsigned int) match_pos);
    /* find next matching strings */
    match_pos = pos_str(str, find, 0);
}
return 0;
}

int in_pos_mid(char* str, unsigned int first, unsigned int last)
{
    unsigned int len = strlen(str);
    unsigned count;
    if (len == 0 || last < first) /* bad arguments */
        return -1;
    if (first > len) first = len;
    if (last > len) last = len;
    /* second comparison of first and last values */
    if (first > last)
        return -1;
    count = last - first + 1;
    if (first > 0) {
        in_delete(str, 0, first);
        last -= first; /* adjust value of last */
        /* update len with new string length */
        len -= first;
    }
    if (last < len)
        in_delete(str, (last+1), (len-last));
    return 0;
}

int in_count_mid(char* str, unsigned int index, unsigned int count)
{
    unsigned int len = strlen(str);
    if (len == 0)
        return -1;
    if (index > len)
        center(VERSION, 2);
    printf("\n\n\n");
    do {
        printf("Enter source filename -> ");
        gets(infile); printf("\n");
        infile_var = fopen(infile, "rt");
        if (infile_var == NULL) {
            printf("Cannot open file %s\n\n", infile);
            printf("Exit ? (Y/N) ");

```



```

        quit_ch = getche();
        if ((quit_ch == 'Y') || (quit_ch == 'y'))
            exit(0);
        printf("\n");
    }
} while (infile_var == NULL);
do {
    printf("Enter destination filename -> ");
    gets(outfile); printf("\n");
    outfile_var = fopen(outfile, "wt");
    if (outfile_var == NULL) {
        printf("Cannot open file %s\n\n", outfile);
        printf("Exit ? (Y/N) ");
        quit_ch = getche();
        if ((quit_ch == 'Y') || (quit_ch == 'y'))
            exit(0);
        printf("\n");
    }
} while (outfile_var == NULL);
do {
    printf("The default script filename is %s\n", script);
    printf("Enter script filename ");
    printf("press [Enter] for default -> ");
    gets(genstr); printf("\n");
    if (strlen(genstr) != 0) {
        strcpy(script, genstr);
        use_same_script = FALSE;
    }
    else
        use_same_script = TRUE;
    script_var = fopen(script, "rt");
    if (script_var == NULL) {
        printf("Cannot open file %s\n\n", script);
        printf("Exit ? (Y/N) ");
        quit_ch = getche();
        if ((quit_ch == 'Y') || (quit_ch == 'y'))
            exit(0);
        printf("\n");
    }
} while (script_var == NULL);
if (have_read_script == FALSE ||
    use_same_script == FALSE) {
    printf("Reading and processing the script file ...");
    num_script = 0; /* initialize script line counter*/
    while ((!feof(script_var)) &&
        (num_script < (MAX_SCRIPT_LINES-1))) {
        fgets(genstr, 80, script_var);
        get_script(genstr, &script_lines[num_script], &ok);
        if (ok == TRUE) num_script++;
    }
    have_read_script = TRUE;
    printf("\n");
}
fclose(script_var);
if (num_script > 0) {
    while (!feof(infile_var)) {
        fgets(line, 80, infile_var);

```

```

        if (strlen(line) > 0) {
            for (i = 0; i < num_script; i++)
                switch (script_lines[i].op_char) {
                    case 'D':
                        delstr(line, script_lines[i].dtext);
                        break;
                    case 'K':
                        kilstr(line, script_lines[i].dtext);
                        break;
                    case 'R':
                        translate(line, script_lines[i].dtext);
                        break;
                }
            printf("%s", line);
            fprintf(outfile_var, "%s", line);
        }
        fclose(infile_var);
        fclose(outfile_var);
    }
    printf("\n");
    printf("Want to process more files? (Y/N) ");
    go_on = getche();
    while (!(go_on != 'Y' || go_on == 'y'));
}

int center(char* string, byte line_num)
/* center string on specified line */
{
    gotoxy(40 - strlen(string) / 2, line_num);
    printf("%s", string);
}

int get_script(char* string,
               struct script_cmd *script_line,
               boolean *good_line)
/* parse string into script commands */
{
    char ch;
    byte index1, index2, index3;
    STRING str0;
    ch = *string;
    if (ch >= 'a' && ch <= 'z') ch += 'A' - 'a';
    *good_line = FALSE;
    if (ch == 'D' || ch == 'K' || ch == 'R') {
        script_line->op_char = ch;
        index1 = pos_char(string, '|', 0);
        if (index1 > -1) {
            index2 = pos_char(string, '|', index1 + 1);
            if (index2 > -1) {
                strcpy(script_line->dtext, string);
                in_pos_mid(script_line->dtext, index1+1, index2-1);
                *good_line = TRUE;
                if (ch == 'R') {
                    index3 = pos_char(string, '|', index2+1);
                    if (index3 > -1) {
                        strcpy(str0, string);
                        strcat(script_line->dtext, "|");
                        in_pos_mid(str0, index2+1, index3-1);
                    }
                }
            }
        }
    }
}

```

```

        strcat(script_line->dtext, str0);
    }
    else
        *good_line = FALSE;
}
}
}
}
int delstr(char* string, char* substr)
/* delete substring from string. case sensitive. */
{
    int i;
    unsigned int len = strlen(substr);
    i = pos_str(string, substr, 0);
    while (i > -1) {
        in_delete(string, i, len);
        i = pos_str(string, substr, 0);
    }
}
int kilstr(char* string, char* substr)
/* delete substring from string. case insensitive. */
{
    int i;
    unsigned int len = strlen(substr);
    STRING strcopy;
    strcpy(strcopy, string);
    /* convert both string copy and substring to uppercase */
    strupr(strcopy);
    strupr(substr);
    i = pos_str(strcopy, substr, 0);
    while (i > -1) {
        in_delete(string, i, len); /* delete from argument */
        in_delete(strcopy, i, len); /* delete from copy */
        i = pos_str(strcopy, substr, 0);
    }
}
int translate(char* string, char* pattern)
/* replace a substring with another */
{
    unsigned int findlen, replen;
    int i;
    STRING find, replace;
    i = pos_char(pattern, '|', 0);
    strcpy(find, pattern);
    in_delete(find, i, strlen(pattern)-i);
    strcpy(replace, pattern);
    in_delete(replace, 0, i+1);
    findlen = strlen(find);
    replen = strlen(replace);
    i = pos_str(string, find, 0);
    while (i > -1) {
        in_delete(string, i, findlen);
        in_insert(string, replace, i);
        i = pos_str(string, find, i+replen+1);
    }
}

```

Listing 6.6 Contents of file "pas2c.src">

```
R |Turbo Pascal|C|
R |(*|/*|
R |{|/*|
R |*|}|*|/|
R |}|}|*|/|
R |BEGIN|{|
R |END|}|
R |FUNCTION|void|
R |PROCEDURE|void|
D |THEN|
D |DO|

        index = len;
        if ((index + count) > len)
            count = len - index + 1;
        if (index > 0) {
            in_delete(str, 0, index);
            /* update len with new string length */
            len -= index;
        }
        if (count < len)
            in_delete(str, count, (len-count+1));
        return 0;
    }
void pad_left(char* str, char pad_char, int num_chars)
{
    char pad_str[41];
    memset(pad_str, pad_char, num_chars);
    pad_str[num_chars] = '\0';
    in_insert(str, pad_str, 0);
}
void pad_right(char* str, char pad_char, int num_chars)
{
    char pad_str[41];
    memset(pad_str, pad_char, num_chars);
    pad_str[num_chars] = '\0';
    in_insert(str, pad_str, strlen(str)+1);
}
void trim_left(char* str, char ch)
{
    int i, count = 0;
    for (i = 0; ( *(str+i) != '\0' ) && (*(str+i) == ch) ); i++
        count++;
    in_delete(str, 0, count);
}
void trim_right(char* str, char ch)
{
    int i, count = 0;
    unsigned int len = strlen(str);
    for (i = len-1; ( i >= 0 ) && (*(str+i) == ch) ); i--
        count++;
    str[len-count] = '\0';
}
void trim_ends(char* str, char ch)
{

```

```

    trim_left(str, ch);
    trim_right(str, ch);
}

```

Listing 6.5 Source code for the application "tanslan.c"

```

/*
C file that manipulates text in a set of files.
Two files are involved: the file containing the names of
the processed files, and the script file containing the
operations to be performed on each file.
If the output filename is omitted, the manipulated text is
sent back to the input file.
The script file contains the following patterns:
D |<text>|    <== case sensitive deletion of <text>
K |<text>|    <== case insensitive deletion of <text>
R |<old text>|<new text>| replace <old text> with <new text>
*/
#include <stdio.h>
#include "conio.h"
#include "string.h"
#include "strops2.h"
#define TITLE "MULTI-FILE TEXT MANIPULATION"
#define VERSION "Version 1.0"
#define DEFAULT_SCRIPT_FILE "PAS2C.SRC"
#define MAX_SCRIPT_LINES 45
typedef char STRING[81];
typedef unsigned int word;
typedef unsigned char byte;
enum booleans { FALSE, TRUE };
typedef enum booleans boolean;
struct script_cmd {
    char op_char;
    STRING dtext;
};
main()
{
    FILE *infile_var, *outfile_var, *script_var;
    STRING infile, outfile, script;
    STRING line, genstr;
    struct script_cmd script_lines[MAX_SCRIPT_LINES];
    word num_script, i;
    char go_on, quit_ch;
    boolean ok, use_same_script, have_read_script;
    go_on = 'Y';
    strcpy(script, DEFAULT_SCRIPT_FILE);
    have_read_script = FALSE;
    do {
        clrscr();
        center(TITLE, 1);

```

```

R |REPEAT|do|
R |clrscr|clrscr|
R |clreol|clreol|
R |ClrScr|clrscr|
R |FOR|for|
R |SQR|(^|
R | = | = |
R |:=|:=|
R |WRITELN|printf("\n |
R |WRITE|printf(|
R |READLN|scanf("%|
R |READ|scanf("%|
R |'|'|
R |WRITELN|printf("\n");|
R |UNTIL|while not|
R |<>|!=|
R |NOT|!|
R |AND|&&|
R |WORD|word|
R | OR | || |
R |IF|if|
R |ELSE|else|
R |CASE|switch|
R |WHILE|while|
R |TO|; <=|
R |Length|strlen(|
R |);|}|

```

Listing 6.7 Sample Turbo Pascal program used as an input for the "translan.c" program

```

PROGRAM SIEVE_TEST;
CONST size = 7000;
      MAX_ITER = 100;
VAR i, prime, k, count, iter : INTEGER;
    flags : ARRAY [0..size] OF BOOLEAN;
BEGIN
  WRITELN('START ',MAX_ITER,' iterations'),
  FOR iter := 1 TO MAX_ITER DO BEGIN
    count := 0;
    FOR i := 0 TO size DO flags[i] := TRUE;
    FOR i := 0 TO size DO
      IF flags[i] THEN BEGIN
        prime := k + k + 3;
        k := k + prime;
        WHILE k <= size DO BEGIN
          flags[i] := FALSE;
          k := k + prime;
        END;
        count := count + 1;
      END;
  END;
  WRITELN('^G,count,' primeS');
END.

```

Listing 6.8 Sample listing emitted by the TRANSLAN.C program

```

PROGRAM SIEVE_TEST;
CONST size == 7000;
      MAX_ITER == 100;
VAR i, prime, k, count, iter : INTEGER;
    flags : ARRAY [0..size] OF BOOLEAN;
{
  printf("\n "START ",MAX_ITER," iterations");
  for iter = 1 ; <= MAX_ITER {
    count = 0;
    for i = 0 ; <= size flags[i] = TRUE;
    for i = 0 ; <= size
      if flags[i] {
        prime = k + k + 3;
        k = k + prime;
        while k <= size {
          flags[i] = FALSE;
          k = k + prime
        };
        count = count + 1
      };
    };
  printf("\n "G,count," primes");
}.
*/.

```

Listing 6.9 Manually-edited C program based on the listing emitted by the "translan.c" program

```

/* Sieve test program */
#include <stdio.h>
#define size 7001
#define MAX_ITER 100
#define TRUE 1
#define FALSE 0
main()
{
  int i, prime, k, count, iter;
  unsigned char flags[size];
  printf("START %d iterations\n",MAX_ITER);
  for (iter = 0 ; iter < MAX_ITER; iter++) {
    count = 0;
    for (i = 0; k < size; k++)
      flags[i] = TRUE;
    for (i = 0; k < size; k++) {
      if (flags[i] == 1) {
        prime = k + k + 3;
        k = k + prime;
        while (k <= size) {
          flags[i] = FALSE;
          k += prime;
        }
        count++;
      }
    }
    printf("\n\007%d primes\n",count);
  }
}

```

第七章 高级指针和内存分配技术

本章讨论操作指针和处理内存分配问题的高级技术。概念在实用之处通过开发一个动态串软件包来说明。在这个过程中，也讨论高级内存分配技术、类型分配的里里外外、技巧性指针操作。虽然这里开发的程序是专为动态串的，但许多技术也可用于别的内存分配和指针处理问题。

如果在学习 C 之前你学过 BASIC，则可能最使你迷惑的一件事就是 C 怎样处理串。你也许对它们是如此低效而感到震惊。也许许多时间痛苦地花在学习须先肯定分配了足够的空间给一个串，才能拷入它之上了。在一个复杂的程序之中，这可能很难做到。但除了因不够的或错误的内存分配产生的那些错误难以定位和调试外，它并非如此的糟。

在本章你将找到一些缓和这些问题的方法，而且实际上，你将看到一种超出和离开 BASIC 提供的那种串的一种方法。这里开发的是一种不仅能实现字符型串，而且还能实现任意类型串的方法。另外，如果有必要的话，这些通用串可以变为自动增长的串。

首先让我们只回顾一下标准 C 串怎样工作。C 串只是字符数组。串的结束由 null (空) 字符指示 (具有值 0x00)。这种空终止有三个重要后果。为了找到串长，你须扫描该串寻找空字符。对一种被认为是有效的语言如 C。这显然是低效的。第二个后果是你不能用串存储任意类型的二进制数据 (例如，部分码和数量)，因为 0x00 (空字符) 对整数和浮点类型具有有效字节。随 C 提供的标准串函数也许会误解这样的字节为串尾。第三，也可能是最重要的后果，是当由一个串向另一串拷贝时，无从知道该串中有多少空间。这使得写通用串函数非常困难。

即使有所有这些问题，C 仍有一张王牌。由于在许多方面它是一种低级语言，所以它非常有力，因此建立一些不同于标准串工作的串是较容易的。我们将从别的语言如何处理串开始。一个经典例子是 Turbo Pascal™3.0 中使用的串，它有下列等价的 C 结构：

```
struct turbo_pascal_string {
    unsigned char currlen;
    unsigned char data[MAXSIZE];
};
```

在 Turbo Pascal 中，当前串长由编译程序和运行时系统保存，且与该串一起存储。这使找串长比扫描空字节有效得多。但是，有两个差别：因为长度存储在一个字节中，所以串不能比 255 个字符长。另外，串的定制大小 (即最大值) 必须在编译时间确定。所以当这一办法解决某些问题的同时，它仍有其局限性。

某些 BASIC 使用下列结构的等价结构用于它们的串：

```
struct basic_string {
    unsigned int dimlen;
    unsigned int currlen;
    unsigned char data[CURRDIMSIZE];
};
```


这种串可存储 64K 之多的数据，并且串字符的存储是动态分配的。分配的量可以变化。dimlen 域指示当前分配的字节数，而 currlen 域指示该串当前使用的长度。这成为表示串的最佳方法之一，因为它们实际上可以是任意长度（当然，至少可达 64K），而且你在存储到串中的数据类型上不受限制。

7.1 动态串

在 C 串实现这种串时，你可能会改用下面的结构，而且在此过程中可能会建一新类型：

```
typedef struct dynamic-str-struct {
    unsigned int dimlen;
    unsigned int currlen;
    unsigned char * data;
}-dynamic str;
```

在紧跟 currlen 之后存储的不是串数据，而是到该数据的指针。这允许你独立于串结构本身，能方便地给串数据分配空间并指向之。

要使用这样一个串，你必须先说明 dynamic-str 类型的一个变量，之后再给该数据分配空间。分配的字节数随后被存在 dimlen 中。当前串长也被置为 0。这个数据在每次向串添加或删除数据时都被更新。例如：

```
#include <alloc.h>
#include <string.h>
dynamic_str mystr;
char msg[] = "hello";
main() {
    mystr.data = malloc(25);
    mystr.dimlen = 25;
    mystr.currlen = 0;
    memcpy(mystr.data, msg, sizeof(msg));
    mystr.currlen += sizeof(msg);
}
```

假如没有足够空间存储数据呢？一种好的办法是使该串动态地增长。Turbo C 函数 realloc() 具有这种功能。这个函数允许你改变一个已分配块的大小以使之更大或更小。如果新的大小与旧的至少一样大，则存储的数据保持不变。重新确定为较小的大小则引起数据被截取，结果是丢失信息。

当扩大一个块时，realloc() 试图得到直接在堆上旧数据后的多余的字节。这种情况下，无需移动内存。如果这不可能，则使用堆上第一个足够大小的块。旧数据随后被拷到新位置，并且旧块被放回堆。

这个可能的拷贝有两个后果。第一，这代表了使用这种方法而不使用链接表方案。例如，在链接表方案中，无需拷贝内存；代替之的是指针却可被操作。稍后我们将进一步讨论。

第二个后果是，由于数据可能被移动，指向该数据的指针变量可能会改变它的值，并且在 realloc() 调用之后不能再指望它不变了。例如，在下列程序序列之后使用指针上将是不安全的：

```

#include <alloc.h>
main() {
    char *p, *q;
    p = malloc(200); /* allocate 200 bytes */
    q = p;           /* set pointers equal */
    p = realloc(300); /* enlarge the allocated space */
    /* q may now point to freed memory. Don't use it */
}

```

如果你在一个函数中使用一个动态数组，则这会变得非常巧妙。你永远也不能十分肯定何时指向它是安全的，因为你可能不知道该函数是否使用了 `realloc()`。绕过它的方法是在动态串指针外面包一层别的结构，并且总是从该结构引用该串。这正是用早先给出的动态串结构所能做的事。这样你就能有下面的程序：

```

#include <alloc.h>
void myfunc(dynamic_str *s);
main() {
    dynamic_str mystr;
    mystr.data = malloc(200);
    myfunc(&mystr);
    if (mystr.data[1] == 'Y') /* then do something useful. */
        /* ... */
}
void myfunc(dynamic_str *s)
{
    s->data = realloc(s->data, 600);
}

```

尽管数据指针可能由 `myfunc()` 中的 `realloc()` 改变，但主程序中 `mystr.data[i]` 的使用仍是十分安全的。

7.2 通用串

有了动态串类型（刚定义的）和适当的函数，我们就能创建与大多数 BASIC 中的串作用相似的串。但你也可以更进一步。C 的长处之一是它的类型分配，它是强迫一种数据类型用做另一种的能力。类型分配对指针特别有效，例如：

```

char *p;
int *i;
/* ... */
i = (int *) p;

```

在类型分配之后 `P` 指向的字符数据可以被当作整数数据。上一节显示了一个指向字符串的动态数组结构。用同样的方法你没有原因不能存储别的类型的数据。例如，你可以有整数、浮点数、“widgets”，或任意我们所希望的结构的数据。你必须使编译程序认为数据是正确的类型。可这样做：使用指针分配，存储该串中各元素的大小，并适当改变 `dimlen` 和 `currlen` 的比例。之后你就到了下列通用串结构。

```

typedef struct vstr_struct {
    unsigned int dimlen; /* initial size of allocated space */
    unsigned int currlen; /* current length of string */
    int esize; /* size of string element */
    int inc; /* no. of elements to add on resizing */
    void *data; /* pointer to string data */
}; vstr;

```

新结构是 `vstr` 类型的，它代表变长串。因为这个结构可用于任意类型的串，所以数据指针的类型目前定为 `void` (空)。

我们已加了 `esize` 和 `inc` 两个域到该结构。`esize` 域存储该串的每个元素所占的字节数。无论何时你想扩大该串都要用到 `inc` 域。它告诉你在请求更多的空间时要分配多少附加的元素。由于在 `realloc()` 调用期间可能需拷贝数据，你可能也要尽可能多地加元素以使拷贝值得做。`inc` 域可为你的具体应用而调整以优化执行。浪费的字节是由于使之过大造成的，但如果它过小则会产生过频的拷贝（因为有更多的到 `realloc()` 的调用）。

假如你想建一个整数串。下面的程序序列显示了如何把 `vstr` 定为 25 个整数的大小并把增量大小设置为 10 个整数：

```
#include <alloc.h>
void init_vstr(vstr s) {
    s.dimlen = 25;
    s.currlen = 0;
    s.esize = sizeof(int);
    s.inc = 10;
    s.data = calloc(25, sizeof(int));
}
```

注意用 `sizeof()` 确定每个元素的字节数的用法。尽管在大多数 PC 机上整数是两字节，你也不能肯定别的机器都是这样。最安全的方法是让编译程序决定。推荐用函数 `sizeof()` 用于此用途。

7.3 指针分配

既然你已决定了你的整数数组的大小，那么你怎样在它里面存储和访问数据呢？正确的方法是用指针分配，正如下例所示。它在你的整数串中存储 0—9 这 10 个整数，然后遍历该串，并打印出它们：

```
/* Simple dynamic string example (vstrex1.c) */
#include <stdio.h>
#include <alloc.h>
typedef struct vstr struct {
    unsigned int dimlen; /* initial size of allocated space */
    unsigned int currlen; /* current length of string */
    int esize; /* size of string element */
    int inc; /* no. of elements to add on resizing */
    void *data; /* pointer to string data */
} vstr;
main() {
    vstr s;
    int i, *ip;
    s.dimlen = 25;
    s.currlen = 0;
    s.esize = sizeof(int);
    s.inc = 10;
    s.data = calloc(25, sizeof(int));
    for (i=0; i < 10; i++)
        ((int *) (s.data))[i] = i; /* store data */
    s.currlen = 10; /* good thing to do, even if we don't use it here */
    for (i=0, ip = (int *) (s.data); i < 10; i++, ip++)
        printf("%d\n", *ip);
}
```

这个串有意用两种不同的方法访问。在第一种情形下，s.data 直接被分配成为整数类型的指针。之后尽管它是个整数数组，你仍可标出 s.data 指向的数组的地址。回忆一下 C 怎样进行指针运算：无论何时你加一个数到一指针变量，C 都确定指针变量指向的类型的大小。随后它把这用作比例因子来计算添加到存储于该指针中的地址外的实际数目。如果你有程序片段 $P = P + n$ ，其中 P 是个指针而 n 是个整数，则运算实际上这样发生：

$$P = p + \text{sizeof}(*p) * n$$

第二件事是在 C 中下列语句是等价的：

(1) $x = *(p + n);$

(2) $x = p[n];$

在整数串例子中，s.data 指针先被指定为整数指针以设置适当的比例因子。之后你可用下标访问串中的整数。

例子中的第二个循环显示了访问串中数据的另一方法。在这种情况下，要说明一个辅助整数指针，然后通过类型分配设置它等于 s.data。这样，随着你扫描该数组，这个指针就被增值。使存储在指针中的有效地址增加 2，这是整数的大小。这第 2 种方法通常比第一种更方便。因为一旦你设置了辅助指针，你就不必在每次访问该数组时反复分配类型。但是，如果你用 realloc() 改变串的大小则这种方法是危险的，因为指向数据的实际指针可能被改变。如果你象执行下面这样的程序，则它极可能使你的系统崩溃：

```
vstr s;
int *p;
/* ... initialize s here ... */
p = (int *)s.data; /* then set p to point to string data */
/* ... */
s.data = realloc(500);
*p = 57; /* DON'T DO: p might be a dangling pointer here */
```

存取通用串的最好方法是用宏。对整数串，你可能有一这样的宏：

```
#define int_array ((int *) (s.data))
```

这样你就可以用 int_array[i] 来访问一个串元素，它被扩展到 (int* (s.data)) [i]。一个问题是这个宏假定你的 vstr 名字为 S，你可这样建一更通用的宏：

```
#define int_array(x) ((int *) (x.data))
```

然后写 int_array1 (s) [i] 这样的宏。这些方法中无一完全令人满意，所以最好是选择一个惯例并坚持之。

7.4 VSTR 软件包

既然已介绍了所有的基本概念，你已就绪于接受 VSTR 软件包了。程序在清单 7.1 和

7.2 中给出，且含有下列函数：

| Function | Use |
|-------------|---|
| vstr_init() | Checks for proper initialization |
| dimvstr() | Dimensions a vstr |
| clrvstr() | Either dimensions a vstr if it has not already been done, or sets the current length to 0 |
| redimvstr() | Resizes a vstr |
| delvstr() | Frees up memory allocated for a vstr |
| copyvstr() | Copies one vstr into another |
| vstrdel() | Deletes elements from a vstr |
| vstrins() | Inserts elements into a vstr |

所有这些函数在成功时都返回 1，如果某种错误（如内存不够，或无效的参数集）则返回 0。每个函数都将依次解释。

这种 Vstr 结构我们一直在使用，它有细微的改变，请下列程序：

```
typedef struct vstr_struct {
    char marker[5]; /* special initialization marker */
    unsigned int dimlen; /* initial size of allocated space */
    unsigned int currlen; /* current length of string */
    int esize; /* size of string element */
    int inc; /* no. of elements to add on resizing */
    void *data; /* pointer to string data */
} vstr;
```

增加了一个与字节的初始化标志。当一个 Vstr 首次被确定大小时，这个标志被置为特殊码“(@) %”。无论何时一个 / vstr 被传给上面的函数之一，都要做一检查看此标志字节是否有这个码。如果没有，则 vstr 的初始化不正确。一错误信息被打印出来且程序放弃。这个安全检查在你慢慢习惯于用 vstr 软件包期间主要用于调试用途，之后可删除之。你也可选择把它留下，因为它并不花费开销。

7.5 决定 VSTR 软件包的大小

函数 dimvstr() 用于给 vstr 分配内存和设置 esize 及 inc 参数。假如你想有一个 (x, y) 点的数组，如 Turbo C 多边形绘制和填充例行程序所使用的那些一样。你可创建一个点数据类型然后如下分配一个点串：

```
typedef struct point_struct {
    int x,y;
} point;
vstr pstr;
dimvstr(&pstr, 50, sizeof(point), 5);
```

这将分配 50 个点的空间，带有 5 个点的重定大小增量。请再次注意推荐过的用 sizeof() 确定你的 vstr 元素大小的用法。

初始化一个 vstr 的另一方法是用函数 clrvstr()。这个函数具有与 dimvstr() 相同的参数，并且如下工作：假如你有一 vstrv。如果调用 clrvstr() 时 v.data 是 NULL，则参数被用于调用 dimvstr() 来为此串分配内存。如果不是 NULL，则认为该串已分配了，v.currlen 被简单地置为 0，这意谓着“空串”。无论何时你想在一个循环内部重使用一个 vstr，clrustr() 函数都将是很有用的。当重新使用一个 vstr 时，全部你真正需要做的只是把它的当前长度置为 0。不必回收它的空间，然后又重新为它分配空间，除非你有一个旧的很大的串，而你想释放之做它用，你必须在它第一次被使用时分配内存。下面的程序段显示了一种使用它的方便办法：

```
pstr.data = NULL; /* get things started by doing this */
/* ... */
for (i=0; i<10; i++) {
    /* allocate or reset the string */
    clrvstr(&pstr,50,sizeof(point),5);
    /* ... some code that builds and uses pstr ... */
}
/* ... */
```

```

    delvstr(&pstr);    /* free up memory for good */

typedef struct point_struct {
    int x,y;
} point;
vstr pstr;
int i;

```

说明了一个 vstr 后，将其数据指针置为 Null。这样在此后任何时候你都可用 clrvstr() 恢复或分配该串。虽然你不必这样做，但随着你的程序变得更大而使保持已分配内存的记录非常困难，这种技术是很有用的。遵循这个惯例可节省你许多调试时间。不幸的是，你仍得记住在程序之初把数据指针置为 NULL。你可在编译时间通过初始化整个 Vstr 结构来实现。

```

vstr pstr = {
    {0,0,0,0,0}    /* Set marker to zero */
    0,              /* Set dimlen to zero */
    0,              /* Set currlen to zero */
    0,              /* Set esize to zero */
    0,              /* Set inc to zero */
    NULL           /* Most importantly, set data pointer to NULL */
};

```

定义于 vstr 头文件 (清单 7.1) 中的宏 NULLVSTR 就执行这种功能。使用它的一个例子是:

```

#include <stdlib.h>
#include "vstr.h"
vstr pstr = NULLVSTR;

```

记住，NULL (由 NULLVSTR 引用) 是一个定义于 stdlib.h (也在 stddef.h, stdio.h, mem.h 和 alloc.h 中) 中的一个宏。在使用 NULLVSTR 宏时必须包含这些头文件之一。

无论何时你想改变分配给一个串的内存大小，都要用到函数 vedimvstr()。它主要是供其它 vstr 函数内部使用，虽然你也可直接使用它。redimvstr() 函数调用 Turbo C realloc() 函数做重定大小的工作。如果你想使串变得小一些，则它也调整 currlen 域 0 以反映可能发生的任何截除。要使用这个例行程序，你只需把已分配的 vstr 地址和新的理想大小传给它即可，如下例所示:

```

#include <stdlib.h>
#include "vstr.h"
main() {
    vstr float_str = NULLVSTR;
    clrvstr(&float_str, 20, sizeof(float), 5); /* room for 20 floats */
    /* ... */
    redimvstr(&float_str, 500); /* now there's room for 500 */
    /* .... */
}

```

7.6 用动态串插入和删除

从标准 C 库中遗漏的用于空终止串的两个函数是插入和删除操作。Vstr 包为动态串提供了这样的功能。

`vstrins()` 允许你在一个 `vstr` 的任何位置插入一个元素数组。在插入之前，它检查是否有足够的空间分配给串的新长度。如果不够，则它自动调用 `redimvstr()` 要更多的空间。因为加入的元素数组是做为 `wid` 指针传递的，所以你可以往串中插入任意类型的数据。但你还必须传递元素数目 `n`，所以它认为数组含有 `n * size` 字节的数据。如果 `redimvstr()` 不能分配更多的内存（例如 `inc=0` 时），数据就被插入但串可能得被删节。

插入位置从 0 开始计算，0 是串的开头。如果你给它一个等于或大于当前串长的位置，则数据被有效地联结到尾部。下例显示了插入到一个 `vstr` 的开头，中间和尾部的一些数据：

```
/*
   Example of inserting into a vstr  (vstrcx2.c)
   Must link with: vstr.obj
*/
#include <stdio.h>
#include "vstr.h"
void main() {
    vstr v = NULLVSTR;
    clr_vstr(&v, 10, sizeof(char), 4); /* allocate 10 bytes */
    if (!vstrins(&v, 0, "hello", 5)); /* insert at the beginning */
    if (!vstrins(&v, 3, "p me ", 5)); /* insert in the middle */
    if (!vstrins(&v, v.currlen, "ad my truck", 11)); /* add onto the end */
    if (!vstrins(&v, v.currlen, "", 1)); /* add a null byte to end */
    printf("The vstr data is '%s'.\n", v.data);
    printf("The current length is %d bytes.\n", v.currlen);
}
```

执行该程序则它打印下列文字：

```
The vstr data is 'help me load my truck'.
The current length is 22 bytes.
```

有几件事要注意：一是该串不能由一个空字节自动终止。实际上，这正是开发动态串的主要原因。但如果 `vstr` 含有你想打印出的字符数据，则你必须在尾部加一个空字节以便使 `printf()` 工作正常。你不能简单地给该数据传一个 0；`vstrins()` 期望的是个地址。传递空字节的地址的方便的办法是仅仅传递一个空串。编译程序将给空字节分配内存随后把它的地址传给 `vstrins()`。

要注意的第二件事是即使你原来只给你的串分配了 10 个字节，最后结果所含的也会比这多。你不必显示地重定 `Vstr` 的大小，`vstrins()` 为你执行这种功能。

图 7.1 显示了重定大小的过程。`vstrins()` 重定大小的算法使用加到 `vstr` 的字节数目的最大值 (`numbytesneeded`, `inc`)。这里，`numbytesneeded` 是容纳插入在串中所需的附加字节数目，而 `inc` 是 `vstr` 结构的 `inc` 域。在前一例子中，`inc` 被置为 4 字节。当插入了串 `admytruck` 后，串空间已满，所以 `ustrins()` 知道它不得不加一些字节。由于要加的字节数 (11) 比 `inc` (4) 大，所以刚刚 11 个字节被加到该串的大小中去。这时，该串有 21 个字节的数据，再次满了。当空字节随后被联接尾部时，`ustrins()` 必须再次重定该串的大小，但这次加了 `inc` 个字节，因为 4 比 1 大。最终的串含有 25 个字节的空间。但正如输出所示，当前串长是 22 个字节。

除 `vstr` 不被重定大小，及无内存返回到堆外，从一个 `vstr` 中删除元素本质上是个相反的过程。`currlen` 域确实反映变化。下例做插入例子的相反的事：

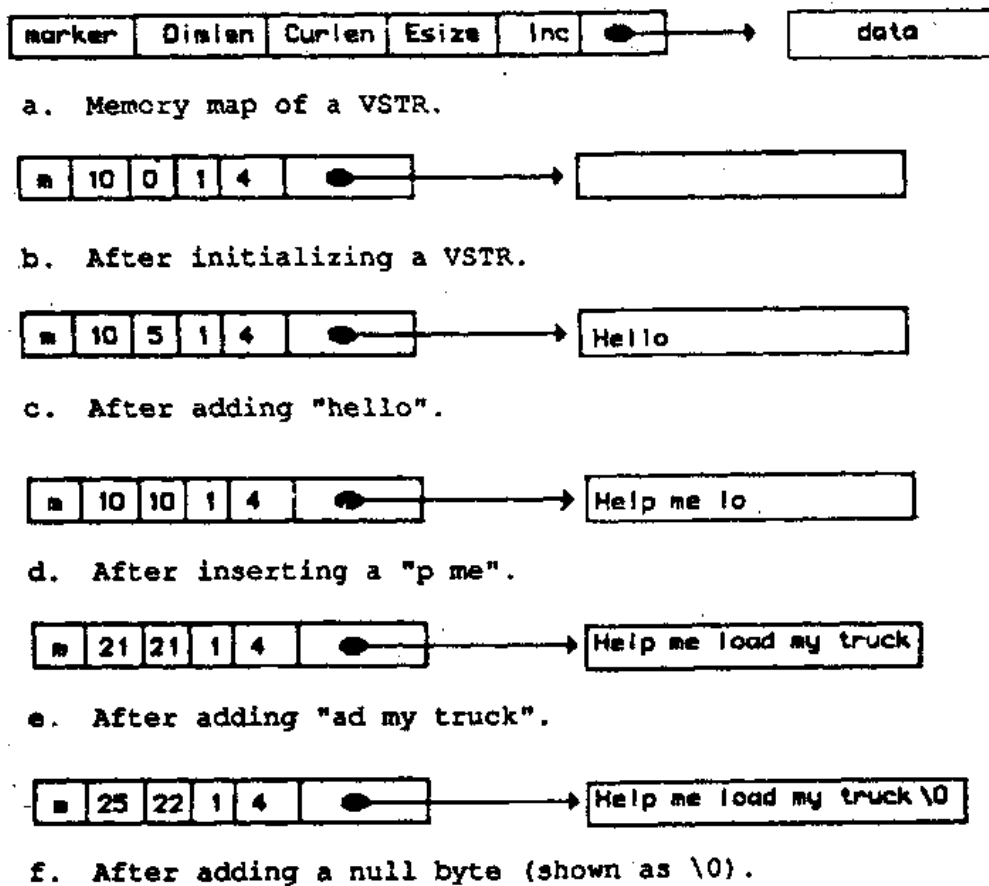


Figure 7.1 Resizing a VSTR

```

/*
   Example of deleting data in a vstr (vstrex3.c)
   Must link with: vstr.obj
*/
#include <stdio.h>
#include "vstr.h"
void main() {
    vstr v = NULLVSTR;
    clrvstr(&v, 30, sizeof(char), 4); /* allocate 30 bytes */
    /* load up the vstr */
    vstrins(&v, 0, "help me load my truck", 21);
    vstrins(&v, v.currlen, "", 1); /* add null byte */
    vstrdel(&v, 3, 5); /* delete "p me " */
    vstrdel(&v, 5, 11); /* delete "ad my truck" */
    printf("The vstr data is '%s'.\n", v.data);
    printf("The current length is %d bytes.\n", v.currlen);
}

```

这个程序在执行时打印下列信息:

```

The vstr data is 'hello'.
The current length is 6 bytes.

```

7.7 动态串与链接表

在使用任何数据结构之前，你应该考虑替换方法和所涉及的交易。这里讲的通用串动态通常在大多数关于数据结构的教材上都没有讲，但对许多应用它们仍是可行的。这些应用是传统上与链接表相联系的那些应用。

一个此类应用是存储一个点链表，它可能在一个 CAD 软件包中用于表示物体。关于这的一个简单例子将在稍后给出，在此，让我们集中于使用链接表和动态串的差别上来。

对两种方法，你可能创建一个 point 结构，例如，它可能含有一个点的 (X, Y) 坐标：

```
typedef struct point_struct {
    int x,y;
} point;
```

使用链接表，你可以创建一个 point-list 类型来表示一个点链表，它是一个含有 (x, y) 坐标和一个指向下一点的链接指针的结构：

```
typedef struct point_link_struct {
    point data;
    point_link_struct *next;
} point_list;
```

让我们检查一下这种表示的花销。对于涉及到内存，每个元素都有存储指向表中下一节点的指针的额外花销。在 Turbo C 中，这种花销对极小、小和中等的内存模式是 2 字节，其它的内存模式是 4 字节。

还要考虑另一类开销：访问该表中每个点的时间。基本上，链接表是个顺序数据类型。你必须顺序扫描该表，寻找下一个指针以访问该表中任一给定的节点。这与使用动态串比较如何呢？

如果你要使用通用串软件包，唯一的花销是 vstr 结构的大小，根据内存模式，是从 15 到 17 个字节（如果我们保留标志字节的话是这样，如果不保留则是 10 到 12）。这是个固定的花销，与串长无关。你可用这种方法非常有效地存储一个节点链表，对任何节点的访问也是非常有效的，因为它涉及的只是一个简单的下标操作（基本上是乘法和加法）。另一好处是你方便地使用 Turbo C 提供的某些内部数组函数，如：qsort(), bsearch() 等。

使用串途径的得失是在插入和删除之中，它要求交换内存。对链接表，只需某些指针操作，其花销极小。另外，如果 realloc() 被调用用来扩展该串，则可能有一些拷贝工作。但移动内存花销如何呢？在 IBM PC 机上，有些特殊的操作能在一个单个指令中移动内存块。这在多数情况下快得不引人注目。唯一的问题（及对通用串的唯一限制）是每个存储块最多只能是 64K 大小。对许多应用，这足够大了。注意没有任何事情能阻止你使用更多的内存与串有效地联系在一起。例如，你可以有一个指针数组，每一个指针指向一个数组。

7.8 一个例子：用动态串表示多边形

我们现在将给出一个为非字符事项使用串的完整例子。在这个例子中，一串圆形，模式下的节点被用鼠标器收集到一个 vstr 中。之后我们调用 Turbo C fill poly() 例行程序来在屏幕上画一封闭的多边形。下面的例子使用了早先开发的鼠标器软件包，还使用了

vstr 软件包和 Turbo C 图形软件包:

```
/*
   Example of storing polygons via vstr's (vstrex4.c)
   Must link with: mouse.obj, vstr.obj, graphics.lib
*/
#include <graphics.h>
#include <stdio.h>
#include "mouse.h"
#include "vstr.h"
/* Convenient macro to access point data */
#define POINT ((point *) (v.data))
typedef struct {
    int x,y;
} point;
void main()
{
    int gd=DETECT, gm=0;
    unsigned int k;
    vstr v = NULLVSTR;
    point p;
    initgraph(&gd,&gm,"");
    init_mouse(MOUSE_NEEDED, gd, gm);
    clrstr(&v,30,sizeof(point),5);
    do {
        while(!(k = mouse_trigger(0)));
        if (k == LEFT_MOUSE_REL) {
            p.x = mouse_grph_x;
            p.y = mouse_grph_y;
            vstrcat(&v,&p);
            /* if we have more than one point, draw a line */
            if (v.currlen > 1) {
                mouse_off(1);
                line(POINT[v.currlen-1].x,POINT[v.currlen-1].y,
                    POINT[v.currlen-2].x,POINT[v.currlen-2].y);
                mouse_on(1);
            }
            else { /* draw starting pixel */
                mouse_off(1);
                putpixel(p.x, p.y, WHITE);
                mouse_on(1);
            }
        }
        while (k != RIGHT_MOUSE_REL);
        vstrcat(&v,v.data); /* make a closed polygon boundary */
        setfillstyle(8,4); /* cross hatch fill, red color */
        mouse_off(1);
        fillpoly(v.currlen, (int far *) (v.data));
        mouse_on(1);
        while(!(k = mouse_trigger(0)));
        mouse_reset();
        closegraph();
        /* print out the points in text mode */
        for (k=0; k < v.currlen; k++) {
            printf("%d %d\n", POINT[k].x, POINT[k].y);
        }
    }
}
```

这个程序工作时等待一系列的左鼠标器按钮的放开动作。鼠标器按钮放开时线就画出来了。线的端点存储在 vstrv 中，它被建立用以保存节点集。这个过程一直进行直到右鼠标器按钮放开为止，这时用线表示的多边形就填充好了。

Turbo C fillpoly() 例行程序要求多边形节点代表一个封闭的边界。你可以把第一个节点拷贝到节点串尾部的办法做到这一点。之后就可把填充式样设置为横直相交平行线阴影，而填充色设置为红色来调用 fillpoly() 例行程序。在另一个按钮放开动作之后（或左或右），屏幕被设置为文本模式且打印出点的坐标。

注意宏 vstrcat() 的用法。这个宏在 vstr.h 中有，且定义如下：

```
#define vstrcat(v,e) vstrins(v,(v)->currlen,e,1)
```

vstrcat() 函数恰允许你加一个元素到串尾。这对象前一例子的那种情况是非常方便的。

这个例子非常明了地说明了一种用串优越于用链接表的情况。这种情况下，数据结构完全适应 fillpoly() 所要求的。但，你仍可以自动分配点数以允许每个多边形有任意个点。

注意，需要类型分配。你必须把串数据指针指定为一个整数指针，因为这是 fillpoly() 所需要的。另外，点线结构中的 (x, y) 顺序非常重要，因为它是 fillpoly() 所要求的顺序。你可利用了解对节点数组 fillpoly() 所用的准确内存表示的优势。

不必花过多的想象就能看出这个简单例子可以扩展成一个实际的 CAD 软件包，你可用它在线段的基础上画和存物体。在后面的章节你将能看到怎样写一系列这样的串到一个文件，以建立一个线图数据库。

Listing 7.1 Source code for file "vstr.h"

```
/* Variable length string toolkit header file (vstr.h) */
/* Convenient macro to concatenate one element to end of string */
#define vstrcat(v,e) vstrins(v,(v)->currlen,e,1)
/* Macro to initialize a vstr at compile time */
#define NULLVSTR {(0,0,0,0,0),0,0,0,0,0,NULL}
/*
 * Generic "virtual" (ie. variable length, dynamic) string package
 */
typedef struct vstr_struct {
    char marker[5];          /* special initialization marker */
    unsigned int dimlen;     /* initial size of allocated space */
    unsigned int currlen;    /* current length of string (in elements) */
    int esize;               /* size of string element */
    int inc;                 /* no. of elements to add on resizing */
    void *data;              /* pointer to string data */
} vstr;
extern int vstr_init(vstr *v);
extern int dimvstr(vstr *v, unsigned int dimlen, int esize, int inc);
extern int redimvstr(vstr *v, unsigned int newdimlen);
extern int clrvstr(vstr *v, unsigned int dimlen, int esize, int inc);
extern int delvstr(vstr *v);
extern int copyvstr(vstr *t, vstr *s);
extern int vstrdel(vstr *v, int p, int n);
extern int vstrins(vstr *v, int p, void *s, int n);
```

Listing 7.2 Source code for file "vstr.c"

```
/******
 * Variable length string toolkit (vstr.c)
 *****/
#include <stddef.h>
#include <stdio.h>
#include <alloc.h>
#include <conio.h>
#include <string.h>
#include <process.h>
#include "vstr.h"
#define MAX(a,b) ((a) > (b) ? (a) : (b))
#define MIN(a,b) ((a) < (b) ? (a) : (b))
static char vinitflag[] = "@!"; /* special internal marker code */
int vstr_init(vstr *v)
/* Checks for proper initialization, if not, it aborts the program */
{
    if (strcmp(v->marker, vinitflag)) {
        printf("Serious error: vstr not init'ed\nPress return ...");
        getch();
        exit(1);
    }
    return 1;
}

int dimvstr(vstr *v, unsigned int dimlen, int esize, int inc)
/*
    Dimension a virtual string to be initially dimlen elements long,
    with each element being esize bytes wide, and adding up to
    inc bytes during a redimension. If inc = 0, it means the string
    will not be allowed to grow beyond its allocated space.
    Returns 1 on success, 0 if any errors occur.
*/
{
    if ((v->data = calloc(dimlen, esize)) != NULL) {
        v->dimlen = dimlen;
        v->currlen = 0;
        v->inc = inc;
        v->esize = esize;
        strcpy(v->marker, vinitflag); /* set special marker flag */
        return 1;
    }
    else return 0; /* calloc failed */
}

int redimvstr(vstr *v, unsigned int newdimlen)
/*
    Redimensions a vstr to the new dimension length newdimlen.
    If v->inc = 0, or a can't get more memory, a 0 is returned,
    else, a 1 is returned. If the new length is shorter than

```

```

the old current length of the string, the string is truncated,
and data is lost.
*/
{
    if (vstr_init(v)) {
        if ((v->inc) && /* check this flag first */
            (v->data = realloc(v->data, newdimlen*v->esize)) != NULL) {
            v->dimlen = newdimlen;
            /* possibly truncate data */
            v->currlen = MIN(v->currlen, newdimlen);
            return 1; /* success */
        }
    }
    return 0; /* failed */
}

int clrvstr(vstr *v, unsigned int dimlen, int esize, int inc)
/*
    If v->data is NULL, the string is dimensioned using the given parms
    by calling dimvstr. If v->data is not NULL, the current length
    of the already allocated data is set to zero.
    An 0 is returned if any errors occur, else a 1 is returned.
*/
{
    if (v->data == NULL) {
        return dimvstr(v, dimlen, esize, inc);
    }
    else {
        if (vstr_init(v)) {
            v->currlen = 0;
            return 1;
        }
        return 0;
    }
}

int delvstr(vstr *v)
/* De-allocates the data portion of a vstr.
   Sets current length to zero.
   If v->data already = NULL, nothing happens.
   An error code is returned if the array has never been
   dimensioned and v->data != NULL.
   Does not free up any dynamic memory pointed to by the data,
   so that better have been done before calling this.
*/
{
    if (v->data != NULL) {
        if (vstr_init(v)) {
            free(v->data);
            v->data = NULL;
            v->currlen = 0;
            return 1;
        }
    }
}

```

```

    }
    else return 0;
}
return 1;
}
int copyvstr(vstr *t, vstr *s)
/*
Copies source virtual string s into target virtual string t.
If t->data comes in NULL, then memory for t will be allocated
and t will take on the same sizing parms as s, else the old
data in t is replaced, and t will be enlarged if necessary.
Errors return 0, else 1 is returned.
*/
{
    if (vstr_initd(s)) {
        if (clrvstr(t, s->dimlen, s->esize, s->inc)) {
            memmove(t->data, s->data, s->currlen*s->esize);
            t->currlen = s->currlen;
            return 1; /* successful */
        }
    }
    return 0; /* default is error */
}
int vstrdel(vstr *v, int p, int n)
/*
Deletes UP TO n elements from virtual string v at pos'n p. If p
is out of range nothing happens. If n <= 0, nothing happens.
An error code of 0 is returned if the string has never been
dimensioned, else a 1 is returned.
*/
{
    int k;
    char *t;
    if (vstr_initd(v)) {
        if ((p >= 0) && (n > 0) && (p < v->currlen)) { /* in range ? */
            if ((k = p+n) >= v->currlen) {
                v->currlen = p; /* chop off end */
            }
            else {
                t = (char *)v->data;
                memmove(t+p*v->esize, t+k*v->esize, (v->currlen-k)*v->esize);
                v->currlen -= n;
            }
            return 1; /* successful */
        }
    }
    return 0; /* error */
}
int vstrins(vstr *v, int p, void *s, int m)
/*

```

Inserts the data pointed to by s into vstr v. It is assumed that the data has the same element size as v. Up to n elements are inserted, starting at position p, (counted by zero). If p >= v->currlen, then the data is concatenated on the end. If we need to allocate more memory, then MAX(n,v->inc) elements are added. However, if v->inc is zero, then the data will be inserted, but data will be truncated off the end of the string. Returns 0 if there's an error, else a 1 is returned.

```

*/
{
    char *t;
    int addsize;
    /* check for initialization and p in range */
    if (vstr_init(v) && p >= 0) {
        addsize = v->currlen + n - v->dimlen;
        if (addsize > 0) {
            /* we need more room */
            if (v->inc != 0) { /* are we allowed to add any? */
                addsize = MAX(addsize, v->inc);
                if (!redimvstr(v, v->dimlen+addsize)) return 0;
            }
            else { /* string not enlarged, so data will be truncated */
                if (p < v->currlen) {
                    t = (char *)v->data; /* point to start of data */
                    if (n > (v->currlen-p)) {
                        n = v->currlen - p; /* we'll be replacing */
                    }
                    else { /* make room for inserted data */
                        memmove(t+(p+n)*v->esize, t+p*v->esize, v->currlen-p-n);
                    }
                    memmove(t+p*v->esize, s, n*v->esize); /* copy in source */
                    return 1; /* success */
                }
                return 0; /* can't concatenate on end, no room */
            }
        }
        t = (char *)v->data; /* point to start of data */
        if (p >= v->currlen) { /* just concatenate */
            p = v->currlen;
        }
        else { /* make room for inserted data */
            memmove(t+(p+n)*v->esize, t+p*v->esize, (v->currlen-p)*v->esize);
        }
        memmove(t+p*v->esize, s, n*v->esize); /* copy in source */
        v->currlen += n;
        return 1;
    }
    else return 0;
}

```

第八章 Turbo C 中的通用编程

Turbo C 支持通用编程并提供一些通用例行程序执行排序和查寻。本章简单地介绍这些例行程序,并讨论写通用函数的方法,和给出一个附加的通用排序和查寻例行程序库。

通用编程是现代软件工程发展起来的技术之一。基本概念是开发一些能对尽可能多的不同类型起作用的例行程序。某些算法可适用于所有数据类型。但其它则仍更被限制于某一类数据。例如,排序和查寻可用于数字和非数字类型,如整数、实数字符和串,相反,返回统计值如最小值、最大值和平均值的例行程序则显然被限制于数字数据类型。在两者中任一情况下,当为某些或全部数据类型开发通用例行程序时,你不必重写已适应接收数据类型的例行程序形式了。通用程序的好处是着实节省大量时间。

8.1 通用例行程序

Turbo C 中的通用排序和查寻函数在 `stdlib.h` 中定原型:

1、`qsort` 用快速排序法排序一个数组。`qsort` 函数的原型是:

```
void qsort(void *base, size_t nelem, size_t width,
           int (*fcmp)(const void*, const void*));
```

`base` 参数是指向数组基地址的(第 0 个元素)的指针。`nelem` 和 `width` 参数分别提供给 `qsort` 例行程序即将排序的元素个数和每个元素的大小。`fcmp` 是指向一个比较任意两个数组元素的函数的指针。`fcmp` 函数的总的形式是:

```
int fcmp(const void *element1, const void *element2)
```

比较函数的结果归于下列三类之一:

| Result of fcmp | Significance |
|----------------|----------------------|
| < 0 | element1 < element2 |
| = 0 | element1 == element2 |
| > 0 | element1 > element2 |

2、`bsearch` 在一个完全排好序的数组中执行一个二进制查寻以寻找一元素。`bsearch` 原型是:

```
void *bsearch(void *key, void *base,
              size_t nelem, size_t width,
              int (*fcmp)(const void*, const void*));
```

`key` 参数是一个指向被查寻元素的指针。`base` 参数是指向数组基地址(即第 0 个元素)的指针。`nelem` 和 `width` 给查寻例行程序分别提供即将扫描的元素的个数和每个元素的大小。`fcmp` 是一个指向比较任意两个数组元素的函数的指针。`bsearch` 函数在找到一个元素时就返回一个指向匹配元素位置的指针;未找到则返回 `NULL` 指针。

3、Lsearch 在数组中执行线性查寻以寻找一元素。Lsearch 函数被说明为:

```
void *lsearch(void *key, void *base,
              size_t nelem, size_t width,
              int (*fcmp)(const void*, const void*));
```

key 参数是指向被寻找的参数的指针。base 参数是指向数组基地址 (即第 0 个元素) 的指针。这个数组不必排序就能使这个查寻例行程序正常工作。nelem 和 width 参数分别提供查寻程序即将检查的元素个数和每个元素的大小。fcmp 是一指向一个比较任两个数组元素的函数的指针。如果找到了, 则 Lsearch 函数返回指向该匹配元素的指针。如果未找到, 则 key 元素被附在数组尾。

4 Lfind 在一个数组中执行线性查寻查找一元素。Lfind 函数被说明为如下:

```
void *lfind(void *key, void *base,
            size_t nelem, size_t width,
            int (*fcmp)(const void*, const void*));
```

key 参数是指向被寻找元素的指针。base 参数是指向数组基地址的指针。该数组不必被排序即能使该查寻例行程序正常工作。nelem 和 width 参数分别提供给该查寻例行程序即将扫描的元素个数和每个元素的大小。fcmp 是指向能比较任意两个数组元素的函数的指针。如果找到, 则该 lfind 函数返回一个指向匹配元素位置的指针; 否则返回一个 NULL 指针。

下面的简单程序显示了用整数数组使用 qsort 和 bsearch 的用法。程序标升序显示整数数组并随后指示含有整数 34 的数组索引。函数 intcmp 被包含用来执行 int 类型元素的比较。源程序如下:

```
#include <stdio.h>
#include <stdlib.h>
main()
{
    int num[] = { 12, 34, 5, 67, 232, 22, 133 };
    int i, n = sizeof(num) / sizeof(int);
    int j = 67;
    int intcmp(const void*, const void*);
    qsort(num, n, sizeof(int), intcmp);
    for (i = 0; i < n; i++)
        printf("%d %3d\n", i, num[i]);
    i = (int*) bsearch(&j, num, n, sizeof(int), intcmp) - (int*) num;
    printf("\nFound %d in array index number %d\n", j, i);
}

int intcmp(const void *i, const void *j)
{
    return (*((int*)i) - *((int*)j));
}
```

下一个程序排序并查寻一个串数组。qsort 例行程序用于把数组排序。Lfind 用于执行线性查寻寻找串 Paul。程序显示含有被寻找的串的数组索引, 然后显示排好序的数组。因为比较串的过程中激活了 strcmp 函数, 所以用了头文件 string.h:

```

#include <stdio.h>
#include <stdlib.h>
#include "string.h"
main()
{
    char str[10][11] = { "Robert", "Bobbi", "Keith", "James", "David",
        "Kim", "Thomas", "Paul", "Peter", "John" };
    char name[11] = "Kim";
    int i, n = 10;
    unsigned found;
    qsort(str, n, 11, strcmp); /* sort array */
    /* search for name */
    found = ((unsigned) lfind(name, str, (size_t*) n, 11, strcmp) -
        (unsigned) str) / sizeof(name);
    printf("found %s in index %u\n\n", name, found);
    for (i = 0; i < n; i++)
        printf("%d %s\n", i, str[i]);
}

```

下一个程序把通用排序和查寻例行程序用于结构。程序定义结构 mail_rec。一个有 5 个元素的结构化数组被初始化、排序、并受到二进制查寻。结构化数组基于其串类型的域排序程序显示匹配数据的数组索引，然后以升序显示该数组：

```

#include <stdio.h>
#include <stdlib.h>
#include "string.h"
struct mail_rec {
    char name[31];
    unsigned int age;
    double wt;
};
typedef struct mail_rec mail;
int struct0_cmp(const void* e1, const void* e2)
{
    return strcmp( ((mail*)e1)->name, ((mail*)e2)->name);
}
main()
{
    mail person[5] = { { "Namir", 34, 180.0 },
        { "Bobbi", 31, 165.0 },
        { "Keith", 29, 155.0 },
        { "James", 35, 190.0 },
        { "David", 41, 190.0 } };
    mail who = { "Keith", 29, 155.0 };
    unsigned i, n = 5;
    unsigned found;
    qsort(person, n, sizeof(mail), struct0_cmp);
    found = ( (unsigned) bsearch(&who, person, n, sizeof(mail),
        struct0_cmp) -

```

```

        (unsigned) person) / sizeof(mail));
printf("Found %s in index %d\n\n", who.name, found);
for (i = 0; i < n; i++)
    printf("%d %s is %2d years old and weighs %lg bounds \n",
        i, person[i].name, person[i].age, person[i].wt);
}

```

8.2 建立通用程序

本节检查建立你自己的通用排序和查寻函数的基本构成成份。正如它所显现的那样，这种软件构成成分的建筑块十分简单。

第一个且是最重要的方面是操作于传输数据数组和单个数据项的 void 类型的指针。因为它们被说明为 Void 类型，所以编译程序不能执行任何必要的指针运算。解决办法是把这些 Void 指针指定为 unsigned char 型的指针，并在以后贯穿于通用例行程序使用。这允许使用只占一个单字节的数据类型。使用类型分配指针后，编译程序就能成功地执行指针运算。通用例行程序中的指针运算只比非通用的精致一点。为访问第三个元素，(给定每个元素的大小)，总的形式是： $\text{address of } i\text{'th element} = \text{ptr} + i * \text{element_size}$

通用例行程序的第二个组成部分处理数组元素的赋值和拷贝。它要求使用 memmove 函数拷贝一或多个元素。

第三个构成成份是局部标量或数组变量。这可由先说明 unsigned char 指针再在动态内存分配中用 malloc 或 calloc (或甚至 realloc) 来使用它们。极力推荐在通用例行程序退出之前用释放函数释放动态内存。

第四个构成成份是用比较函数完成涉及通用数据的决定。这相当简单且以下面的形式出现：

```
(*fcmp)((ptr + i * elmsize), (ptr + j * elmsize))
```

其中 ptr 是类型分配指针，elmsize 是以字节为单位的元素大小，i 和 j 是被比较的通用数组元素的索引。

下面是一个通用外壳排序例行程序的源代码。这个程序比 qsort 慢，但由于无递归发生所以所需堆栈空间极少：

```

void gen_shell_sort(void *base, int nelem, int elmsize,
                    int (*fcmp)(const void*, const void*))
{
    int i, j, jump = nelem;
    unsigned char done;
    unsigned char *tempo, *ptr = (unsigned char *) base;
    tempo = (unsigned char *) malloc(elmsize);
    while (jump > 1) {
        jump /= 2;
        do {
            done = 1;
            for (j = 0; j < (nelem - jump); j++) {
                i = j + jump;
                if ((*fcmp)((ptr+i*elmsize), (ptr+j*elmsize)) < 0) {
                    /* swap i'th and j'th elements */
                    done = 0;
                    memmove(tempo, (ptr+i*elmsize), elmsize);

```

```

        memmove((ptr+i*elmsize), (ptr+j*elmsize), elmsize);
        memmove((ptr+j*elmsize), tempo, elmsize);
    }
}
} while (!done);
}
free(tempo); /* restore dynamic memory */
}

```

8.3 补充的通用排序 / 查寻库

在清单 8.1 和 8.2 中有补充的排序 / 查寻例行程序库 gensort.h 和 gensort.c 的源程序。该库含有下面三类函数：

1. 比较两个元素的函数。这些函数包含有用于 int、unsigned int、long、unsigned long 和 double 等类型的函数。这些函数相当平常，但在通用编程中却是需要的。它们可以作为你想插入的附加的类型比较函数的核心。

2. 通用排序例行程序包含有：

a. gen-shell-sort 用 shell-Metzner 算法排序一个数组。该函数被说明为：

```

void gen_shell_sort(void *base, int nelem, int elmsize,
                    int (*fcmp)(const void*, const void*));

```

base 参数是指向数组基地址（即第 0 个元素）的指针。nelem 和 elmsize 参数分别给该排序程序提供即将排序的元素个数和每个元素的大小。fcmp 是指向比较任意两数组元素的函数的指针。这个排序例行程序的参数表与 qsort 的一样。

b. gen-insert-sor 在数组中插入一个新元素并保持已排的顺序。该函数被说明为：

```

int gen_insert_sort(void *base, void* key,
                   int *nelem, int elmsize,
                   int (*fcmp)(const void*, const void*))

```

base 参数是指向数组基地址的指针。key 参数指向插入的元素。nelem 和 elmsize 参数分别给该排序程序提供即将排序的元素个数和每个元素的大小。fcmp 是指向比较任意两数组元素的函数的指针。

c. gen-merge-sort 把一个排好序的数组与另一个数组合并。函数被说明为如下：

```

int gen_merge_sort(void *base1, void *base2,
                  int *nelem1, int nelem2,
                  int elmsize,
                  int (*fcmp)(const void*, const void*))

```

base1 参数是指向第一个数组的基地址的指针。base2 参数是指向第二个数组的基地址（即第 0 个元素）的指针。nelem1、nelem2 和 elmsize 参数分别给该排序程序即提供即将排序的元素个数和每个元素的大小。

fcmp 是指向一个比较任意两个数组元素的函数的指针。第一个数组在将第二个数组的元素放在适当位置上后返回。

d. gen-reverse-array 把一个数组的元素的顺序反过来。函数被说明为：

```
5int gen_reverse_array(void *base, int nelem, int elmsize)
```

base 参数是指向第一个数组的基地址（即第 0 个元素）指针。nelem 和 elmsize 给该排序程序分别提供即将反序的元素个数和每个元素的大小。

3. 通用查寻例程序，它包含以下内容：

a. **bidir-find** 在一个数组中执行双向线性查寻。这个函数有点类似 Turbo C 的 **lfind** 线性查寻函数。基本差别在于所用的算法不同。**bidir-find** 使用在一个数组中查寻一个元素的统计概率，认为所有的元素都有相同的被找到的可能。统计分析表明这种查寻的索引的平均值指向中间值和它周围的区域。这样数组中中间的元素将是第一个被检查的元素。下一步的查寻步骤使用两个索引使查寻在中间元素以上和以下交替进行。随着每次失败的重复，查寻索引就远离中间值而移向数组的上限和下限。

该函数被说明为：

```
int bidir_find(void *key, void *base,
               int nelem, int elmsize
               int (*fcmp)(const void*, const void*))
```

key 参数指向被插入的元素。base 参数是指向数组的基地址（即第 0 个元素）的指针。nelem 和 elmsize 参数分别给该排序程序提供即将检验的元素个数和每个元素的大小。fcmp 是指向一个比较任意两数组元素的函数的指针。**bidir-find** 函数返回匹配的数组元素的索引或在未找到匹配元素时返回 -1。

b. **bidir-search** 在一数组中执行双向线性查找。该函数使用与 **bidir-find** 相同的算法。差别是如果在数组中未找到该元素，则它被作为数组新的最后元素加到数组中去。

该函数被说明为：

```
int bidir_search(void *key, void *base,
                 int *nelem, int elmsize,
                 int (*fcmp)(const void*, const void*))
```

key 参数指向被插入的元素。base 参数是指向该数组基地址（第 0 个元素）的指针。nelem 和 elmsize 分别给排序程序提供即将检验的元素个数和每个元素的大小，fcmp 是一个指向比较任意两组元素的函数的指针。函数 **bidir-search** 返回匹配元素的索引，或如果无匹配则返回新加的数组元素的索引。

c. **set-index-table** 初始化基于排序的数组的索引查寻表。函数被定义为：

```
int set_index_table(void *table, int *index, void *base,
                    int tbl_size, int nelem, int elmsize)
```

table 参数是指向索引表的指针。index 参数是指向该表使用的索引的数组的指针。base 参数是指向该数组基地址（第 0 个元素）的指针。tbl-size 是指定的表大小。nelem 和 elmsize 参数分别给排序程序提供即将检查的元素的个数和每个元素的大小。如果数组元素太少则该函数返回 0；否则返回 1。

d. **search-index-table** 在一排好序的数组中查寻一给定的元素。使用了一个查寻索引表以加速查寻。索引表的各项是用于缩小即检查的数组索引范围的。在给定的范围内，二分法查寻算法可更进一步提高查寻速度。

函数被说明为：

```

int search_index_table(void *key, void *table,
                      int *index, void *base,
                      int tbl_size, int nelem,
                      int elmsize,
                      int (*fcmp)(const void*, const void*))

```

key 参数指向被寻找的元素。table 参数是指向索引表的指针。index 是指向该表使用的索引的数组的指针。base 参数是指向该数组的基地址（第 0 个元素）的指针。tbl-size 是指定的表大小。nelem 和 elmsize 分别给排序程序提供即将检查的元素的个数和每个元素的大小。失败的查寻返回-1；否则，返回匹配元素的数组索引。

下面有一些使用 gensort.c 库中不同例行程序，使用一系列短的独立程序的例子。这些程序使用内部初始化的各种数据的数组。

下面的程序使用通用外壳排序程序排序一个整数数组。该数组被用 bidir-find 函数检查以确定存储整数 22 的数组元素的位置。匹配数组元素的索引和已排好序的数组的清单被显示：

```

#include <stdio.h>
#include <stdlib.h>
#include "string.h"
#include "gensort.h"
main()
{
    int num[] = { 12, 34, 5, 67, 232, 22, 133 };
    unsigned i, n = sizeof(num) / sizeof(int);
    int found, j = 22;
    /* sort array */
    gen_shell_sort(num, n, sizeof(int), intcmp);
    /* search for array element storing the value in 'j' */
    found = bidir_find(&j, num, n, sizeof(int), intcmp);
    printf("Found %d in array element %d\n\n", j, found);
    puts("The sorted array of integer is :");
    for (i = 0; i < n; i++)
        printf("%u %3d\n", i, num[i]);
}

```

下面的程序也用通用外壳排序程序排序一个串数组。该数组被 bidir-function 函数检查以确定存储串 James 的数组元素的位置。匹配数组元素的索引和已排好序的数组清单被显示：

```

#include <stdio.h>
#include <stdlib.h>
#include "string.h"
#include "gensort.h"
main()
{
    char str[5][11] = { "Namir", "Bobbi", "Keith", "James", "David" };
    unsigned i, n = 5;
    int found;
    char name[11] = "James";
}

```

```

/* sort array */
gen_shell_sort(str, n, 11, strcmp);
/* search for array element storing the string "James" */
found = bidir_find(&name, str, n, 11, strcmp);
printf("Found %s in array element %d\n\n", name, found);
puts("The sorted array of names is:");
for (i = 0; i < n; i++)
    printf("%u %s\n", i, str[i]);
}

```

下面的程序通过每次调用例行程序 `gen-insert-sort` 时插入一个串渐渐建立一个有序的串数组。该数组被 `bidir-search` 函数检查以存储串 `Keith` 的数组元素的位置。匹配的数组元素的索引和已排好序的数组清单被显示:

```

#include <stdio.h>
#include <stdlib.h>
#include "string.h"
#include "gensort.h"
main()
{
    char str[5][11];
    char name[11] = "Keith";
    unsigned i, n = 0;
    int found;
    /* insert names */
    gen_insert_sort(str, "Namir", &n, 11, strcmp);
    gen_insert_sort(str, "Bobbi", &n, 11, strcmp);
    gen_insert_sort(str, "Keith", &n, 11, strcmp);
    gen_insert_sort(str, "James", &n, 11, strcmp);
    gen_insert_sort(str, "David", &n, 11, strcmp);
    /* search for array element storing the string "Keith" */
    found = bidir_search(name, str, &n, 11, strcmp);
    printf("Found %s in array element %d\n\n", name, found);
    puts("The sorted array of names is:");
    for (i = 0; i < n; i++)
        printf("%u %s\n", i, str[i]);
}

```

下一个程序给通用排序程序加了新的一层复杂度。一个结构类型的数组被该通用外壳排序程序排好序并随后显示。该结构被定义为:

```

struct mail_rec {
    char name[31];
    unsigned int age;
    double wt;
};

```

`name` 域用于排序该数组。写了一个特殊的元素比较函数 `structo-cmp` 并把它用在了通用排序例行程序参数表中了:

```

#include <stdio.h>
#include <stdlib.h>
#include "string.h"
#include "gensort.h"
struct mail_rec {
    char name[31];
    unsigned int age;
    double wt;
};
typedef struct mail_rec mail;
int struct0_cmp(void *e1, void *e2)
{
    return strcmp(((mail*)e1)->name, ((mail*)e2)->name);
}
main()
{
    mail person[5] = { { "Namir", 34, 180.0 },
                        { "Bobbi", 31, 165.0 },
                        { "Keith", 29, 155.0 },
                        { "James", 35, 190.0 },
                        { "David", 41, 190.0 } };
    unsigned i, n = 5;
    gen_shell_sort(person, n, sizeof(mail), struct0_cmp);
    for (i = 0; i < n; i++)
        printf("%s is %2d years old and weighs %lg pounds \n",
            person[i].name, person[i].age, person[i].wt);
}

```

下面的程序初始化两个结构化数组。每个数组各自用 qsort 排序，之后这些数组被合并并排序。然后新的大数组的顺序被反序且其内容被显示。该程序显示了对函数 gen-merge-sort 和 gen-reverse-arry 的调用。

```

#include <stdio.h>
#include <stdlib.h>
#include "string.h"
#include "gensort.h"
struct mail_rec {
    char name[31];
    unsigned int age;
    double wt;
};
typedef struct mail_rec mail;
int struct0_cmp(void *e1, void *e2)
{
    return strcmp(((mail*)e1)->name, ((mail*)e2)->name);
}
main()
{
    mail person1[10] = { { "Namir0", 34, 180.0 },
                          { "Bobbi0", 31, 165.0 },
                          { "Keith0", 29, 155.0 },

```



```

        { "James0", 35, 190.0 },
        { "David0", 41, 190.0 } };
mail person2[5] = { { "Namir1", 34, 180.0 },
                    { "Bobbi1", 31, 165.0 },
                    { "Keith1", 29, 155.0 },
                    { "James1", 35, 190.0 },
                    { "David1", 41, 190.0 } };

unsigned i, n = 5, m = 5;
/* sort first array */
qsort(person1, n, sizeof(mail), struct0_cmp);
/* sort second array */
qsort(person2, m, sizeof(mail), struct0_cmp);
/* merge sort both arrays */
gen_merge_sort(person1, person2, &n, m, sizeof(mail), struct0_cmp);
/* reverse the order of the array */
gen_reverse_array(person1, n, sizeof(mail));
/* display structured array in descending order */
for (i = 0; i < n; i++)
    printf("%s is %2d years old and weighs %lg bounds \n",
           person1[i].name, person1[i].age, person1[i].wt);
}

```

最后一个程序说明了通用索引查寻表例行程序的用法。一个 10 个串的数组首先被用 qsort 排序。该有序数组随后被用来建一个有三项的大小适宜的索引表。索引表与查寻存储串 Paul 的数组元素有关。该程序显示存储有被寻找的串的元素索引。然后显示排好序的串的数组：

```

#include <stdio.h>
#include <stdlib.h>
#include "string.h"
#include "gensort.h"
#define TABLE_SIZE 3
#define ARRAY_SIZE 10
#define STRING 11
main()
{
    char str[ARRAY_SIZE][STRING]
        = { "Namir", "Bobbi", "Keith", "James", "David",
            "Kim", "Thomas", "Paul", "Peter", "John" };
    char name[STRING] = "Keith";
    char table[TABLE_SIZE][STRING];
    unsigned index[TABLE_SIZE];
    unsigned i, n = ARRAY_SIZE, tbl_size = TABLE_SIZE;
    int found;
    qsort(str, n, STRING, strcmp); /* sort the array of string */
    /* setup the index search table */
    set_index_table(table, index, str, tbl_size, n, STRING);
    /* use the index search table */
    found = search_index_table(name, table, index, str,
                               tbl_size, n, 11, strcmp);
    printf("found %s in index %d\n\n", name, found);
}

```

```

    for (i = 0; i < n; i++)
        printf("%u %s\n", i, str[i]);
}

```

Listing 8.1 Source code for the generic sort/search header file "gensort.h"

```

int intcmp(int*, int*);
long longcmp(long*, long*);
int uintcmp(unsigned int*, unsigned int*);
long ulongcmp(unsigned long*, unsigned long*);
int doublecmp(double*, double*);
void gen_shell_sort(void*, unsigned, unsigned,
    int (*fcmp)(const void*, const void*));
void gen_insert_sort(void*, void*, unsigned*, unsigned,
    int (*fcmp)(const void*, const void*));
int gen_reverse_array(void*, unsigned, unsigned);
int gen_merge_sort(void*, void*, unsigned*, unsigned, unsigned,
    int (*fcmp)(const void*, const void*));
int bidir_find(void*, void*, unsigned, unsigned,
    int (*fcmp)(const void*, const void*));
int bidir_search(void*, void*, unsigned*, unsigned,
    int (*fcmp)(const void*, const void*));
int set_index_table(void*, unsigned*, void*,
    unsigned, unsigned, unsigned);
int search_index_table(void*, void*, unsigned*,
    void*, unsigned, unsigned, unsigned,
    int (*fcmp)(const void*, const void*));

```

Listing 8.2 Source code for the supplementary generic sort/search library "gensort.c"

```

#include <stdlib.h>
#include <string.h>
int intcmp(int *i, int *j)
{
    return (*i - *j);
}

long longcmp(long *i, long *j)
{
    return (*i - *j);
}

int uintcmp(unsigned int *i, unsigned int *j)
{
    return (int) (*i - *j);
}

long ulongcmp(unsigned long *i, unsigned long *j)
{
    return (long) (*i - *j);
}

int doublecmp(double *x, double *y)
{
    if (*x < *y)
        return -1;
    else if (*x > *y)
        return 1;
    else

```

```

        return 0;
    }
    void gen_shell_sort(void *base, unsigned nelem,
        unsigned elmsize,
        int (*fcmp)(const void*, const void*))
    {
        unsigned i, j, jump = nelem;
        unsigned char done;
        unsigned char *tempo, *ptr = (unsigned char *) base;
        tempo = (unsigned char *) malloc(elmsize);
        while (jump > 1) {
            jump /= 2;
            do {
                done = 1;
                for (j = 0; j < (nelem - jump); j++) {
                    i = j + jump;
                    if ((*fcmp)((ptr+i*elmsize), (ptr+j*elmsize)) < 0) {
                        done = 0;
                        memmove(tempo, (ptr+i*elmsize), elmsize);
                        memmove((ptr+i*elmsize), (ptr+j*elmsize), elmsize);
                        memmove((ptr+j*elmsize), tempo, elmsize);
                    }
                }
                int (*fcmp)(const void*, const void*)
            } while (!done);

            unsigned i, found;
            unsigned char *ptr = (unsigned char *) base;
            unsigned char *kee = (unsigned char *) key;
            if (*nelem > 0) {
                for (i = 0, found = 0; (i < *nelem && !found); i++)
                    if ((*fcmp)((ptr+i*elmsize), kee) > 0) {
                        memmove((ptr+(i+1)*elmsize),
                            (ptr+i*elmsize),
                            (*nelem - i) * elmsize);
                        memmove((ptr+i*elmsize), kee, elmsize);
                        found = 1;
                    }
                (*nelem)++;
                if (!found)
                    memmove((ptr + *nelem * elmsize), kee, elmsize);
            }
            else {
                memmove(ptr, kee, elmsize);
                (*nelem)++;
            }
        }
    }

    int gen_reverse_array(void *base, unsigned nelem, unsigned elmsize)
    {
        unsigned median = nelem / 2, i, j;
        unsigned char *tempo, *ptr = (unsigned char *) base;
        if (nelem < 3)
            return 0;
        tempo = (unsigned char *) malloc(elmsize);
        for (i = 0, j = nelem-1; i < median; i++, j--) {
            memmove(tempo, (ptr+i*elmsize), elmsize);
            memmove((ptr+i*elmsize), (ptr+j*elmsize), elmsize);
            memmove((ptr+j*elmsize), tempo, elmsize);
        }
    }

```

```

    }
    free(tempo);
    return 1;
}

int gen_merge_sort(void *base1, void *base2,
                  unsigned *nelem1, unsigned nelem2,
                  unsigned elmsize,
                  int (*fcmp)(const void*, const void*))
{
    unsigned i, j, k;
    unsigned char *ptr0 = (unsigned char *) base1;
    unsigned char *ptr1 = (unsigned char *) base1;
    unsigned char *ptr2 = (unsigned char *) base2;
    if (*nelem1 < 1 || nelem2 < 1)
        return 0;
    ptr0 = (unsigned char *) calloc(*nelem1, elmsize);
    /* copy main array into temporary array */
    memmove(ptr0, ptr1, *nelem1 * elmsize);
    for (i = 0, j = 0, k = 0; (i < *nelem1 && j < nelem2; k++) {
        if ((*fcmp)((ptr0+i*elmsize), (ptr2+j*elmsize)) < 0) {
            memmove((ptr1+k*elmsize), (ptr0+i*elmsize), elmsize);
            i++;
        }
        else {
            memmove((ptr1+k*elmsize), (ptr2+j*elmsize), elmsize);
            j++;
        }
    }
    if (i < *nelem1) /* copy the rest of array1 */
        memmove((ptr1+k*elmsize),
                (ptr0+i*elmsize),
                (*nelem1-i)*elmsize);
    else /* copy the rest of array2 */
        memmove((ptr1+k*elmsize),
                (ptr2+j*elmsize),
                (nelem2-j)*elmsize);
    *nelem1 += nelem2;
    free(ptr0);
    return 1;
}

int bidir_find(void *key, void *base,
              unsigned nelem, unsigned elmsize,
              int (*fcmp)(const void*, const void*))
/* function that performs linear bidirectional search starting with
the median element and searching towards the ends */
{
    int i, j;
    unsigned char *ptr = (unsigned char *) base;
    unsigned char *kee = (unsigned char *) key;
    if (nelem < 2)
        return -1;
    for(i = nelem / 2 - 1, j = i + 1;
        (i > -1 || j < nelem);
        i--, j++) {
        if (i > -1) {
            if ((*fcmp)((ptr+i*elmsize), kee) == 0)

```

```

        return i;
    }
    if (j < nelelem) {
        if ( (*fcmp)((ptr+j*elmsize), kee) == 0)
            return j;
    }
    return -1;
}

int bidir_search(void *key, void *base,
                unsigned *nelem, unsigned elmsize,
                int (*fcmp)(const void*, const void*))
/* function that performs linear bidirectional search starting with
the median element and searching towards the ends.
If the element is not found it is added to the end of the array.
*/
{
    unsigned i, j;
    unsigned char *ptr = (unsigned char *) base;
    unsigned char *kee = (unsigned char *) key;
    if (*nelem > 1) {
        for(i = *nelem / 2 - 1, j = i + 1;
            (i > -1 || j < *nelem);
            i--, j++) {
            if (i > -1) {
                if ( (*fcmp)((ptr+i*elmsize), kee) == 0)
                    return i;
            }
            if (j < *nelem) {
                if ( (*fcmp)((ptr+j*elmsize), kee) == 0)
                    return j;
            }
        }
    }
    /* add new element */
    memmove((ptr + *nelem * elmsize), kee, elmsize);
    (*nelem)++;
    return (*nelem - 1);
}

int set_index_table(void *table, unsigned *index, void *base,
                  unsigned tbl_size, unsigned nelelem, unsigned elmsize)
{
    unsigned i, j, offset = nelelem / tbl_size;
    unsigned char *ptr = (unsigned char *) base;
    unsigned char *tbl = (unsigned char *) table;
    if (offset < 1)
        return 0;
    for (i = 0, j = 0; j < tbl_size; j++, i += offset) {
        memmove((tbl+j*elmsize), (ptr+i*elmsize), elmsize);
        *(index+j) = i;
    }
    return 1;
}

int search_index_table(void *key,

```

```

        void *table,
        unsigned *index,
        void *base,
        unsigned tbl_size,
        unsigned nelem,
        unsigned elmsize,
        int (*fcmp)(const void*, const void*))
(
    unsigned i, first, last, found = 0;
    unsigned median;
    unsigned char *ptr = (unsigned char *) base;
    unsigned char *tbl = (unsigned char *) table;
    unsigned char *kee = (unsigned char *) key;
    for (i = 1; (i < tbl_size && found == 0); i++) {
        if ( (*fcmp)(kee, (tbl+i*elmsize)) <= 0 ) {
            found = 1;
            first = *(index+i-1);
            if (i < (tbl_size - 1))
                last = *(index+i);
            else
                last = nelem - 1;
        }
    }
    if (found == 0) {
        first = *(index+tbl_size-1);
        last = nelem - 1;
    }
    found = 0; /* reset found flag */
    while ( (first+1) < last && found == 0 ) {
        /* calculate median index */
        median = (first + last) / 2;
        if ( (*fcmp)(kee, (ptr+median*elmsize)) < 0 )
            last = median;
        else if ( (*fcmp)(kee, (ptr+median*elmsize)) > 0 )
            first = median;
        else
            found = 1;
    }
    if (found == 0) {
        if ( (*fcmp)(kee, (ptr+last*elmsize)) == 0 )
            return last;
        else if ( (*fcmp)(kee, (ptr+first*elmsize)) == 0 )
            return first;
        else
            return -1;
    }
    return median;
)

```

第九章 目录实用程序

本章看一下扩展常用 DOS 命令的建筑函数。这些程序是以能很方便地并入你的用户应用程序的函数形式写的。给出的四个实用程序是：

1. 扩展的目录列表产生器函数和应用 它通过允许多至 10 个可指定的通配符程序扩展 MS-DOS 的 DIR 命令 (并且也改进 OS/2 的 DIR 命令)。

2. 支持多个通配符的文件复制函数及应用 它通过支持多个通配符, 和可选择的冗长而安全的拷贝模式来扩展 DOS 的 copy 命令。该安全拷贝模式检查目的目录看每个被拷贝的文件是否存在。如果一个文件在当前和目标目录都存在, 则用户被提示授权拷贝该文件。

3. 实现多文件列表的应用 该应用使用扩展的目录列表函数调用一个实际处理源程序列表的子程序。

4. 灵活的目录跳转函数应用 这个函数扫描目录树并使你的程序只调用一个目录的名字就能移入一个目录, 而不必用全路径名!

9.1 扩展的目录函数和应用

上面的实用程序是建立在对函数 fn-edir 的调用之上的, 该函数可在清单 9.1 和 9.2 中找到。这个函数使用两个 Turbo C 的涉及文件名查询的基本函数, 它们名为 findfirst 和 findnext。findfirst 和 findnext 函数的头部被说明为:

```
struct ffbk* fn_edir(char wildcard[] (MAX_WILDCARD_LEN),
                    int num_wildcard,
                    struct ffbk *files,
                    int *num_files)

int findfirst(const char *pathname, struct ffbk *ffblk, int attrib)
int findnext(struct ffbk *ffblk)
```

pathname 参数定义准确的目录路径并包含明确的文件名或通配符。attrib 参数定义即将选择的目录项的类型。属性可以是一个归档文件、只读文件、系统文件、隐藏文件、目录名, 或磁盘卷标名。结构 ffbk 被定义为:

```
struct ffbk {
    char ff_reserved[21]; /* reserved area */
    int ff_ftime; /* file time stamp */
    int ff_fdate; /* file date stamp */
    long ff_fsize; /* file size */
    char ff_name[13]; /* complete filename */
}
```

到 findfirst 的调用应指定路径名和属性。如果在指定的路径名下有至少一个与要求的

属性匹配的文件被找到，则该函数返回一个非零整数。被找到的文件项的数据通过文件块指针参数 * fblk 返回。findfirst 返回的非零结果也意味着可能有更多的匹配文件项。通常，用一个 WHILE 循环来包含到函数 findnext 的调用。与 findfirst 类似，如果 findnext 找到了由 findfirst 的一个调用指定的更多的匹配文件则它返回一个非零整数。findnext 被重复调用直到它返回 0 为止，0 表明已到了匹配文件的终点。

函数 fnedir 用一个按序排好的保存文件名的动态数组建立匹配文件的清单。因为无法提前预测（甚至大概估计）实际的数组大小，所以使用动态分配和重分配。文件名数组被动态地分配为由宏 DYN-ARRAY-SIZE 指定的某些任意的大小。当数组满时，它被 realloc 函数动态地再分配 INC-ARRAY-SIZE 这个元素。realloc 函数使函数动态再分配易于实现。如果没有它，则程序员有两个其它选择。第一个是开发他或她自己的 realloc。第二个选择是用有序单链表。

保持一个有序数组可减少涉及到查寻重复的名字所用的时间。这是函数 fnedir 的重要方面。假如指定了“*.*”和“*.C”这样的通配符，则这将从相同的目录下取出许多重复的文件名。为防止这种冗余，fnedir 函数捕捉重复的文件名。这也解决了本章给出的调用函数 fnedir 的别的实用程序的潜在的重复文件名的问题。

fnedir 返回一个指向 fblk 结构的指针。结果应重新赋给指针型变量参数 files 以更新动态数组的地址。重要的是一旦动态组的数据不再需要，则直接激活 fnedir 的应用程序或函数就释放动态数组。

清单 9.3 含有 edir, c 的源程序，它是一个列出文件名和文件大小的应用程序。当编译后的该程序不带参数运行时，该应用程序就象你打过参数 *.* 一样运行，并显示出当前目录中全部文件清单。该应用程序能从不同目录和或驱动器显示文件名。但是，不推荐这样做，因为识别准确的文件位置将变得非常令人混淆。当多个通配符连同它们的路径名被指定时，后面的将全是一样的。图 9.1 显示了同已编译的“edir.c”程序的一个样例对话。输出由从 DOS 命令级敲入下面的行指定 *.c 和 *.BAT 通配符来产生：

```
> EDIR *.C *.BAT
```

| Filename | Size | Time | Date |
|------------|-------|----------|------------|
| BASTAT.C | 882 | 02:56:13 | 01-27-1987 |
| BGIDEMO.C | 42530 | 01:05:00 | 12-10-1987 |
| BJEASTAT.C | 882 | 20:34:15 | 03-09-1988 |
| BJBGIDEM.C | 42530 | 20:34:16 | 03-09-1988 |
| CHILDLIS.C | 3137 | 20:38:22 | 03-10-1988 |
| CHILDMN1.C | 1456 | 17:10:15 | 03-10-1988 |
| CHILDMN2.C | 1917 | 17:07:00 | 03-10-1988 |
| CHILDMN3.C | 1949 | 17:14:02 | 03-10-1988 |
| CHILDPRN.C | 1647 | 15:26:24 | 03-10-1988 |
| CHILDWS2.C | 1969 | 16:00:06 | 03-10-1988 |
| CPASDEMO.C | 1682 | 01:05:00 | 12-10-1987 |
| DIROPS.C | 7797 | 22:03:02 | 03-17-1988 |
| FILECOMP.C | 11073 | 01:00:00 | 05-13-1987 |
| GENSORT.C | 7009 | 22:26:00 | 03-15-1988 |
| GETOPT.C | 4228 | 01:05:00 | 12-10-1987 |

| | | | |
|-----------|------|----------|------------|
| LIST.C | 2948 | 11:28:21 | 03-16-1988 |
| MAIN.C | 1102 | 01:05:00 | 12-10-1987 |
| MATHERR.C | 3850 | 01:05:00 | 12-10-1987 |
| OPEN.C | 478 | 14:27:11 | 01-19-1988 |
| STROPS1.C | 1675 | 11:41:12 | 03-16-1988 |
| STROPS2.C | 6349 | 17:45:08 | 03-08-1988 |

press any key to continue

| Filename | Size | Time | Date |
|------------|------|----------|------------|
| T.C | 8891 | 16:01:05 | 03-07-1988 |
| TRANSLAN.C | 7848 | 15:40:06 | 03-10-1988 |

23 files matching in 163829 bytes

26 files matching in 168124 bytes

Figure 9.1 Sample output of a session with application `edir.c`. The command line arguments supplied is `*.C *.BAT`.

9.2 扩展的文件拷贝函数和应用

第二个文件/目录管理函数是 `copyto`，它显示于清单 9.4 和 9.5 中。函数头的说明如下：

```
int fn_copyto(char* destination,
               char wildcard[] (MAX_WILDCARD_LEN),
               int num_wildcard,
               int verbose,
               int safe_mode)
```

`destination` 参数是目标目录的路径名。`wildcard` 参数传递含有通配符的串的数组。`num-wildcard` 参数指定通配符的个数。`verbose` 参数指示该函数在拷贝文件时是否显示文件名。一个非 0 的 `verbose` 参数值打开 `verbose` 模式，而一个零值关闭之。`safe-mode` 参数用于确保在文件拷贝过程中目的目录不会盲目地覆盖旧文件。非 0 的 `safe-mode` 参数值打开安全拷贝模式。如果被拷贝的文件已存在于目的目录，则提示用户授权此拷贝。

`copyto` 函数调用 `fn-edir` 函数访问文件名数组。该函数认为文件名都在当前目录下。低级文件 I/O 函数 `open`, `write`, `read` 和 `close` 用于执行快速缓冲 I/O。缓冲区当前被设置为大约 4K。另外，该函数使用预定义的 Turbo C 标志 `O-CREATE`, `O-RDONLY`, `O-WRONLY` 来控制已打开的文件的状态。用这些标志，该函数能够判别在当前和目的目录的相同文件的出现。`copyto` 函数在无文件匹配任一指定的通配符时返回 0 值；否则返回 1 值。

清单 9.6 含有驱动 `copyto` 函数的一个应用程序。该程序扫描命令行参数，并提示你用 `verbose` 和 `safe` 拷贝模式。如果未提供参数，则程序显示一个联机帮助信息并退出。例如，已编译的应用程序是从 DOS 下被调用来拷贝 `.C` 和 `.EXE` 文件到 `-RAM` 盘的根目录的，如下：

```
>COPYTO E:\ *.C *.EXE
```

9.3 多文件列表实用程序

清单 9.7 含有 `lister.c`，它是把 `fn-edir` 函数的功能同系统调用的多用途性结合在一起的

一个短 C 程序。程序本身可被看作是一种读命令行参数以获得文件通配符的外壳。如果未找到，则程序用 *.* 通配符。一个到 fn-edir 的调用返回一个指向含有即将观察的文件名的数组的指针（假定文件名在当前目录）。一个 FSR 循环用于建立一个调用 list.c 程序（见清单 9.8）并供它一个单个明确的文件名的 DOS 命令。一旦 list.c 退出，你就返回到 lister.c 程序，它提示以看你是否想停止继续观察剩下的文件。例如，已编译的应用程序从 DOS 下被调用以观察 RAM 盘根目录下的 .C 和 .BAT 文件，如下：

list.c 程序是一单独的、独立的用系统函数激活的程序。你可以用另一能读取做为命令行参数提供的文件名的程序名来替换该程序名。list.c 程序用长指针直接写屏幕。使用长指针是因为视频的物理地址很可能落在程序的数据段之外。你可以用 PgUp、PgDn、Home、End 和上下光标键在文本中移动。按 Q 键将退出 list.c 程序。list.c 程序读多至 700 行文本文件并忽略多余的行。

9.4 目录跳转

下一个实用程序对于在目录间来回移动尤为方便。在把文件集中到一起的过程中十分有用的目录树结构在从一个目录移到另一目录时将变得十分麻烦，因为在此过程中你必须在目录树上上下下移动。显示于清单 9.7 的跳转目录程序为此过程提供了一个捷径。它也显示了使用 Turbo C 函数 findfirst() findnext() 和 chdir() 的更多的例子。>LISTER *.C *.BAT

该跳转程序允许你只指定其名字就跳到一不同的子目录中去，而不必指定其余路径名。名字可以是部分的，而该程序将揭示你解决任何混淆。之后你就能直接跳到该子目录，不必考虑你在树中何处。如果名字是唯一的，则无任何提示，且你马上就到该子目录了。

该程序这样工作：创建当前驱动器中完整的目录树的“拉平的”形式。“拉平的”树被放入你的根目录下一特殊的文件中。该文件存储关于每个子目录的名字及其在树中层次的信息。然后当你使用跳转命令时该文件就被咨询以最快速产生子目录的全路径名。这个名字随后被用作 Turbo C 函数 chdir() 的输入，它允许你改变目录。

该跳转程序由两种不同方法调用：>jump -s

这用于创建一个目录图：>jump <partial_dir_name>

它用于移到另一目录。

这样，跳转程序就有两种状态：一个种用于创建特殊的目录文件，另一种是提供跳转功能。跳转程序中的主程序是 savedirc() 和 jumpdir() 它们分别处理这两种状态。

savedir() 程序从根目录开始递归遍历目录途径。Turbo C 函数 findfirst() 和 findnext() 用于返回目录中每个文件的名字和属性。那些是目录而非普通文件的都被清除。然后，这些目录中的每一个都被访问，而且此过程递归进行。大体上，目录树是以深度优先方式被遍历的。

为分辨目录和普通文件。该程序检查结构 struct fblk 中的一个标志，它由 findfirst() 和 findnext() 两者返回。这个结构是：

```
struct fblk {
    char ff_reserved[21];
    char ff_attrib;
```

```

int ff_ftime;
int ff_fdate;
int ff_fsize;
char ff_name[13];

```

标志 ff-attrib 给出 DOS 文件属性，它可取下列码。这些码定义于头文件 dos.L 中：

```

FA_RDONLY      Read only file
FA_HIDDEN      Hidden file
FA_SYSTEM      System file
FA_LABEL       Volume Label
FA_DIREC       Directory
FA_ARCH        Archive

```

你所寻找的那个是 FA-DIRECT。注意你必须特别检查目录“.”和“..”。这些特殊目录分别是当前目录和父目录的名字。为防止一个无限循环，你必须肯定不要重复访问它们。

当一个目录被找到后，它的名字和层次被加到特殊文件 \save.dir，它创建于你的根目录。存储的名字不是全路径名，而只是子目录的名字。层次是一个指示你在目录结构中多深的一个整数，其中 0 意味着子目录是从根目录访问的。 \save.dir 的内容的例子如下：

| | | | |
|--------|----------|---------|--------|
| 0 DOS | 1 QUICKY | 0 TC | 2 V2 |
| 0 MSC | 2 TST | 1 WORK | 3 KEYS |
| 1 LIB | 2 CQ | 1 TSR | 0 CG |
| 1 RMRL | 0 MOUSE | 1 GUIDE | 0 TALK |

在这个例子中，子目录 DOS、MSC、Mouse、TC、CG 和 TALK 都驻留在根目录下。

7. 下面的清单显示了所有目录的全路径名：

| Directory | Pathname |
|-----------|-------------------|
| 0 DOS | \dos |
| 0 MSC | \msc |
| 1 LIB | \msc\lib |
| 1 RMRL | \msc\rmrl |
| 1 QUICKY | \msc\quicky |
| 2 TST | \msc\quicky\tst |
| 2 CQ | \msc\quicky\cq |
| 0 MOUSE | \mouse |
| 0 TC | \tc |
| 1 WORK | \tc\work |
| 1 TSR | \tc\work\tsr |
| 1 GUIDE | \tc\guide |
| 2 V2 | \tc\guide\v2 |
| 3 KEYS | \tc\guide\v2\keys |
| 0 CG | \cg |
| 0 TALK | \talk |
| 1 CG | \talk\cg |

一旦 \save.dir 已建立, 就能用该程序跳到不同目录了。函数 jumpdir() 做所有的工作。基本上, \save.dir 被作为层次一名字对的一个数组被装入内存。然后, 给定一个子目录名, 这个程序顺序扫描该数组, 寻找部分匹配。在这个过程中, 候选的目录的全路径名被构造出来, 使用存储于该数组的层次信息。

当找到匹配后, 该程序看它是否是唯一的。如果是, 则 chdir() 函数就被调用来做跳转, 且给该函数以路径名。如果名字不是唯一的, 则提示用户候选的路径名。通过按 y 或 Y 进行选择。别的任何键都将导致带有下一个匹配路径名提示。

例如, 对早先给出的目录样例, 可用下面的跳转:

```
>jump tst           jumps to \msc\quicky\tst
>jump gu            jumps to \tc\guide
>jump ke            jumps to \tc\guide\v2\keys
>jump cg            not unique, so it will prompt for \cg, and then \talk\cg
>jump \tc\guide     NOT allowed, can't give pathnames!!
```

该跳转程序的一个问题是每次的添加或删除一个目录时, 你都必须运行 jump-s 以重建 \save.dir。这有点讨厌, 通过使该程序驻留内存和监视所有建目录和删目录命令到 DOS 可以缓解之。

\save.dir 被设计为较小, 以便执行跳转命令时读此文件能尽快完成, 当然, 如果你正在用一软磁盘, 则它将相当慢, 所以最好是用硬盘。

另一限制是路径名不能用, 即使是部分的也不行。困难将是改变程序以允许如上例所给出的 ">jump work \ts" 跳转到 \tc\work\str 去。

Listing 9.1 Contents of the header file "fn_edir.h"

```
#define MAX_WILDCARD_LEN 31
#define DYN_ARRAY_SIZE 50
#define INC_ARRAY_SIZE 25
struct fblk* fn_edir(char[][] , int, struct fblk*, int*);
```

Listing 9.2 Source code for function "fn_edir.c"

```
#include "stdlib.h"
#include "dos.h"
#include "dir.h"
#include "string.h"
#include "fn_edir.h"
struct fblk* fn_edir(char wildcard[][MAX_WILDCARD_LEN]
                    int num_wildcard,
                    struct fblk *files,
                    int *num_files)
{
    int n = DYN_ARRAY_SIZE;
    unsigned int blk_size = sizeof(struct fblk);
    int i, j, k, nomatch, not_unique, found;
    struct fblk *ptr;
    /* allocate new block for file structures */
    files = (struct fblk *) calloc(n, blk_size);
```

```

ptr = (struct fblk*) calloc(n,blk_size);
*num_files = 0; /* initialize matching file count */
/* loop to look for the specified wildcards */
for (i = 0; i < num_wildcard; i++) {
    /* search for the first matching entry */
    nomatch = findfirst((wildcard+i),ptr,0);
    /* while a match is found */
    while (!nomatch) {
        not_unique = 0; /* assume filename is unique */
        /* search in the available array of filenames */
        if (*num_files > 0)
            for (j = 0; (j < *num_files) && (!not_unique); j++)
                if (!strcmp(ptr->ff_name, (files+j)->ff_name))
                    not_unique = 1; /* found a match */
        /* if filename is unique process it */
        if (!not_unique) {
            (*num_files)++; /* increment file count */
            /* need more dynamic memory? */
            if (*num_files > n) {
                n += INC_ARRAY_SIZE;
                files = (struct fblk *) realloc(files, n * blk_size);
            }
            /* insert in order */
            if (*num_files > 1) {
                found = 0;
                k = *num_files - 1;
                /* search for proper insertion location */
                for (j = 0; (j < k) && (!found); j++) {
                    if (strcmp(ptr->ff_name, (files+j)->ff_name) < 0) {
                        found = 1;
                        /* move rest of array upward */
                        memmove((files+j+1), (files+j), (k-j)*blk_size);
                        /* insert new element */
                        *(files+j) = *ptr;
                    }
                }
                if (!found) /* insert as last array element */
                    *(files + k) = *ptr;
            }
            else /* assign first entry */
                *files = *ptr;
        }
        /* search for the next matching entry */
        nomatch = findnext(ptr);
    }
}
return files; /* return pointer to dynamic array */
}

```

Listing 9.3 Source code for utility program "edir.c"

```

#include <stdio.h>
#include "conio.h"
#include "stdlib.h"
#include "dos.h"
#include "dir.h"
#include "errno.h"

```

```

#include "string.h"
#include "fcntl.h"
#include "sys\stat.h"
#include "fn_dir.h"
main(int argc, char* argv[])
{
    char wildcard[10][MAX_WILDCARD_LEN];
    int num_wildcard, num_files, i, count;
    struct ffbk *files = NULL;
    long sum = 0;
    clrscr();
    directvideo = 1;
    if (argc < 2) {
        num_wildcard = 1;
        strcpy(wildcard[0], "*.*");
    }
    else {
        num_wildcard = argc - 1;
        for (i = 0; i < num_wildcard; i++)
            strcpy(wildcard[i], argv[i+1]);
    }
    files = fn_dir(wildcard, num_wildcard, files, &num_files);
    if (num_files > 0) {
        puts("  Filename      Size");
        puts("-----");
        for (i = 0, count = 0; i < num_files; i++, count++) {
            if (count > 20) {
                printf("\npress any key to continue");
                getch();
                clrscr();
                puts("  Filename      Size");
                puts("-----");
                count = 0;
            }
            printf("%-13s %10ld\n",
                (files+i)->ff_name,
                (files+i)->ff_fsize);
            sum += (files+i)->ff_fsize;
        }
        printf("\n%d files matching in %ld bytes\n", num_files, sum);
        free(files);
    }
}

```

Listing 9.4 Contents of the header file "fn_copy.h"

```
int fn_copyto(char*, char [][], int, int, int);
```

Listing 9.5 Source code for function "fn_copy.c"

```

#include "stdlib.h"
#include "dos.h"
#include "dir.h"
#include "string.h"
#include "fcntl.h"
#include "fn_dir.h"
int fn_copyto(char* destination,

```

```

        char wildcard[MAX_WILDCARD_LEN],
        int num_wildcard,
        int verbose,
        int safe_mode)
{
    struct fblk *files;
    int num_files, num_bytes, i;
    char ch, path[65];
    char buffer[4096];
    int infile, outfile;
    unsigned int const BUF_SIZE = 4096;
    int line_num;
    files = fn_dir(wildcard, num_wildcard, files, &num_files);
    if (num_files > 0) {
        for (i = 0; i < num_files; i++) {
            strcpy(path, destination);
            strcat(path, (files+i)->ff_name);
            infile = open((files+i)->ff_name, O_RDONLY | O_BINARY);
            outfile = open(path, O_WRONLY | O_BINARY);
            if ((safe_mode) && (outfile > 0)) {
                line_num = wherey(); /* store cursor row number */
                do {
                    gotoxy(1, line_num);
                    clrscr();
                    printf("overwrite %s in destination ? Y/N -> ",
                        (files+i)->ff_name);
                    ch = getch(); printf("\n");
                    if (ch >= 'a' && ch <= 'z') ch += 'A' - 'a';
                } while (ch != 'Y' && ch != 'N');
                if (ch == 'N') {
                    close(infile);
                    close(outfile);
                    continue;
                }
            }
            close(outfile);
            /* reopen file handle to allow new file creation */
            outfile = open(path, O_CREAT | O_WRONLY | O_BINARY);
            /* are both files are opened ok */
            if ((infile > 0) && (outfile > 0)) {
                if (verbose)
                    printf("%s -> %s\n",
                        (files+i)->ff_name, path);
                while ( (num_bytes = read(infile, buffer, BUF_SIZE)) > 0)
                    write(outfile, buffer, num_bytes);
                close(infile);
                close(outfile);
            }
            else {
                if (infile > 0) close(infile);
                if (outfile > 0) close(outfile);
            }
        }
        free(files);
        return 1;
    }
}

```

```

else
    return 0;
}
int backup(char* destination,
           char wildcard[][MAX_WILDCARD_LEN],
           int num_wildcard,
           int verbose)
{
    struct ffbk *files, *ptr;
    int num_files, num_bytes, i, nomatch;
    char ch, path[65];
    char buffer[4096];
    int infile, outfile;
    unsigned int const BUF_SIZE = 4096;
    int t1, t2, d1, d2;
    long int s1, s2;
    long sum_size;
    files = fn_dir(wildcard, num_wildcard, files, &num_files);
    if (num_files > 0) {
        for (i = 0; i < num_files; i++) {
            strcpy(path, destination);
            strcat(path, (files+i)->ff_name);
            nomatch = findfirst(path, ptr, 0);
            if (!nomatch) {
                /* get source file size, time and date stamps */
                s1 = (files+i)->ff_fsize;
                t1 = (files+i)->ff_ftime;
                d1 = (files+i)->ff_fdate;
                /* get destination file size, time and date stamps */
                s2 = ptr->ff_fsize;
                t2 = ptr->ff_ftime;
                d2 = ptr->ff_fdate;
                /* do files match in size, time and date */
                if ( (s1 != s2) || (t1 != t2) || (d1 != d2) )
                    continue;
            }
            infile = open((files+i)->ff_name, O_RDONLY | O_BINARY);
            outfile = open(path, O_CREAT | O_WRONLY | O_BINARY);
            /* are both files are opened ok */
            if ((infile > 0) && (outfile > 0)) {
                if (verbose)
                    printf("%s -> %s\n",
                           (files+i)->ff_name, path);
                while ( (num_bytes = read(infile, buffer, BUF_SIZE)) > 0 )
                    write(outfile, buffer, num_bytes);
                close(infile);
                close(outfile);
            }
            else {
                if (infile > 0) close(infile);
                if (outfile > 0) close(outfile);
            }
        }
        free(files);
        return 1;
    }
    else

```



```
    return 0;
```

```
}
```

Listing 9.6 Source code for program "copyto.c"

```
#include <stdio.h>
#include "stdlib.h"
#include "dir.h"
#include "errno.h"
#include "conio.h"
#include <io.h>
#include "string.h"
#include "fcntl.h"
#include "sys\stat.h"
#include "fn_dir.h"
#include "fn_copy.h"
main(int argc, char* argv[])
{
    char wildcard[10] (MAX_WILDCARD_LEN);
    unsigned int num_wildcard, num_files, i;
    char ch, destination[65];
    int verbose, safe_mode;
    clrscr();
    directvideo = 1;
    if (argc < 2) {
        puts("proper usage is: copyto <destination> <wildcard 1>...\n\n");
        exit(0);
    }
    else {
        strcpy(destination, argv[1]);
        num_wildcard = argc - 2;
        for (i = 0; i < num_wildcard; i++)
            strcpy(wildcard[i], argv[i+2]);
    }
    printf("\n\nUse verbose mode ? (Y/N) ");
    ch = getche(); printf("\n\n");
    verbose = (ch == 'y' || ch == 'Y') ? 1 : 0;
    printf("Use safe copy mode ? (Y/N) ");
    ch = getche(); printf("\n\n");

    safe_mode = (ch == 'y' || ch == 'Y') ? 1 : 0;
    fn_copyto(destination, wildcard, num_wildcard, verbose, safe_mode);
}
```

Listing 9.7 Source code for program "lister.c"

```
#include <stdio.h>
#include "stdlib.h"
#include "dir.h"
#include "errno.h"
#include "conio.h"
#include "string.h"
#include "fcntl.h"
#include "sys\stat.h"
#include "fn_dir.h"
main(int argc, char* argv[])
```

```

char wildcard[10][MAX_WILDCARD_LEN];
unsigned int num_wildcard, num_files, i, count;
struct fblk *files = NULL;
char ch;
char dos_cmd[81];
clrscr();
directvideo = 1;
if (argc < 2) {
    num_wildcard = 1;
    strcpy(wildcard[0], "**.*");
}
else {
    num_wildcard = argc - 1;
    for (i = 0; i < num_wildcard; i++)
        strcpy(wildcard[i], argv[i+1]);
}
files = fn_edir(wildcard, num_wildcard, files, &num_files);
if (num_files > 0) {
    for (i = 0; i < num_files; i++) {
        strcpy(dos_cmd, "LIST ");
        strcat(dos_cmd, (files+i)->ff_name);
        system(dos_cmd);
        printf("press the [X] key to exit or any other key to continue");
        ch = getch();
        if (ch == 'X' || ch == 'x') exit(0);
    }
}
free(files);

```

Listing 9.8 Source code for program "list.c"

```

/* C program that lists small programs. The cursor
   and screen control keys are used to navigate in the listing.
   The listings are protected from any modification. */
#include <stdio.h>
#include "conio.h"
enum booleans { FALSE, TRUE };
typedef enum booleans boolean;
#define MAX_LINES 750
#define LINES_PER_SCREEN 23
main(int argc, char* argv[])
{
    char filename[65];
    FILE *filevar;
    char line[MAX_LINES][81];
    int i, j, k, n, m, count, offset;
    char ch;
    int far *screen = (int far*) 0xB0000000L; /* for color */
    /* int far *screen = (int far*) 0xB0000000L; for mono */
    clrscr();
    gotoxy(33,1);
    puts("PROGRAM LISTER");
    gotoxy(33,2);
    puts("_____ \n\n\n");
    do {

```

```

if (argc < 2) {
    printf("Enter filename -> ");
    gets(filename); putchar('\n');
}
else
    strcpy(filename,argv[1]);
/* set argc to 1 so that subsequent filenames are queried */
argc = 1;
if ( (filevar = fopen(filename,"rt")) == NULL)
    printf("\nCannot open file %s\n\n",filename);
} while (filevar == NULL);
printf("\nReading ");
n = 0;
m = 0;
while (n < MAX_LINES && !feof(filevar)) {
    if ((n % 50) == 0) putchar('.');
    fgets(line[n], 80, filevar);
    n++;
}
fclose(filevar);
n--;
do {
    clrscr();
    k = 80;
    puts(filename);
    for (count = 1; count < LINES_PER_SCREEN; count++) {
        j = m + count;
        offset = 0;
        for (i = 0; (line[j][i] != '\0' && line[j][i] != 0x0A); i++) {
            if (line[j][i] == 0x09)
                offset += 7;
            else
                *(screen + i + k + offset) = line[j][i] | 0x700;
        }
        k += 80;
    }
}
gotoxy(1,25);
printf("use cursor/page control keys, [Q] to quit");
ch = getch();
if (ch == 0) {
    ch = getch(); /* get second byte */
    switch (ch) {
        case 71 :
            m = 0; /* Home */
            break;
        case 79 :
            m = n - LINES_PER_SCREEN; /*End*/
            break;
        case 73 : /* PageUp */
            if (m > 1) m -= LINES_PER_SCREEN;
            if (m < 0) m = 0;
            break;
        case 81 : /* PageDown */
            if (m < n) m += LINES_PER_SCREEN;
            if (m >= (n - LINES_PER_SCREEN))
                m = n - LINES_PER_SCREEN;
    }
}

```

```

        break;
    case 72 : /* Arrow Up */
        if (m > 0) m--;
        break;
    case 80 : /* Arrow Down */
        if (m < n) m++;
        if (m >= (n - LINES_PER_SCREEN))
            m = n - LINES_PER_SCREEN;
        break;
    } /* switch */
    ch = ' ';
}
} while (ch != 'Q' && ch != 'q');
clrscr();
}

```

Listing 9.9 The source code for the jump directory program "jump.c".

```

/*****
 *
 *          Jump Directory Program
 *
 * Calling sequence: >j partial_dir_name % jumps to directory
 *
 *          or
 *
 *          >j -s % save directory tree
 *
 * This program uses a fast lookup file created by the "-s" option to
 * find a match on a directory name, and then jumps there. Since the
 * match may be a partial one, it prompts the user if the match is not
 * unique and verifies that it is the one wanted. If no match is
 * found, or the user says no to each prompt, then no jump takes place.
 *
 * This program reads/creates the file \save.dir in your root directory.
 *
 *****/
#include <stdio.h>
#include <string.h>
#include <dos.h>
#include <dir.h>
/* change this constant if you have more directories than this */
#define MAXNUMDIRS 256
void savedir(void); *
int scandir(int level, char *path, FILE *f, struct fblk fblk);
void jumpdir(char *dirname);
int srch_tree(int level, char *path, char *dir);
int verify(char *dir, char *newpath);
struct dirstruct {
    int level;
    char dir[MAXFILE]; /* MAXFILE = maximum size of file name + null */
} table[MAXNUMDIRS];
FILE *f;
int numdirs, index;
char path[MAXPATH]; /* MAXPATH = maximum size of path name + null */

```

```

main(int argc, char *argv[]) {
    int i,n;
    if (argc > 1) {
        if (!strcmp(argv[1], "-s")) {
            savedir(); /* -s option: he wants to save directory tree */
        }
        else {
            jumpdir(argv[1]); /* else jump to a directory */
        }
    }
    else {
        printf("Usage: j partial_dir_name or j -s\n");
    }
}

void savedir(void)
/*
 * This routine scans the current drive and collects the path names
 * of all directories. The names are stored in a special file
 * called "save.dir" in your root directory. This file can then
 * be used to quickly find the directory to jump to.
 */
{
    struct fblk fblk;
    FILE *f;
    printf("Creating directory table \save.dir\n");
    f = fopen("\\save.dir", "wt"); /* open save.dir in root directory */
    strcpy(path, "\\*."); /* start with root path */
    scandir(0, path, f, fblk);
    fclose(f);
}

int scandir(int level, char *path, FILE *f, struct fblk fblk)
/*
 * This is a recursive routine that walks the directories in a
 * depth-first fashion and accumulates the full path name of
 * each subdirectory, then computes its depth in the directory tree.
 * The subdirectory name and its level are then written to file f.
 *
 * NOTE: since char *path is modified, it should NOT come in as a
 * string literal.
 */
{
    int done, oldlen;
    oldlen = strlen(path); /* keep track of "base" path length */
    done = findfirst(path, &fblk, FA_DIRC); /* grab first directory */
    while(!done) {
        path[oldlen] = 0; /* reset to base path name */
        /* check to see if we found a directory, and that it is not the
         special directories "." (current directory) or ".." (parent
         directory). These last two checks prevent infinite recursion.
        */
        if ((fblk.ff_attrib == FA_DIRC) && strcmp(fblk.ff_name, ".")
            && strcmp(fblk.ff_name, "..")) {
            printf("%d ", level); /* feedback to user */
            fprintf(f, "%d %s\n", level, fblk.ff_name); /* and store */
            path[strlen(path)-3] = 0; /* remove "*. " search pattern */
            strcat(path, fblk.ff_name); /* add new directory name */
            strcat(path, "\\*."); /* and search pattern */
        }
    }
}

```

```

        scandir(level+1, path, f, ffbk);
    }
    if (findnext(ffblk) == -1) done = 1; /* no more files */
}
return;
}

void jumpdir(char *dirname)
/* Opens the directory tree table \save.dir and then searches for
 * the directory node char *dirname. This name can be a partial one.
 */
{
    int i,n;
    if ((f = fopen("\\save.dir", "rt")) == NULL) {
        printf("Directory table file \\save.dir not found\n");
        exit(0);
    }
    numdirs = -1;
    while(!feof(f)) { /* read in the number of directories */
        if (numdirs == MAXNUMDIRS - 1) {
            printf("WARNING: out of room in directory table, ");
            printf("only first %d read\n", MAXNUMDIRS);
            break;
        }
        numdirs++;
        fscanf(f, "%d %s\n", &(table[numdirs].level), table[numdirs].dir);
    }
    fclose(f);
    index = 0; /* initialize directory table index */
    strcpy(path, "\\"); /* start with root directory */
    if (!srch_tree(0, path, dirname)) printf("sub-directory not found\n");
}

int srch_tree(int level, char *path, char *dir)
/*
 * Searches our directory tree for a match, and changes directories
 * and returns a 1 if we do find one, else 0 is returned.
 *
 * "path" is the full path name of the directory we are currently
 * trying. "dir" is the partial or full name of the directory
 * we would like to jump to.
 *
 * NOTE: This is a recursive routine, and uses index and numdirs as
 * global variables. They must be initialized before this
 * routine is called the first time.
 * Also, since char *path is modified, NEVER pass it as a
 * string literal.
 */
{
    int oldlen;
    oldlen = strlen(path); /* keep track of base path */
    while(index < numdirs) {
        path[oldlen] = 0; /* reset to base path name */
        strcat(path, table[index].dir); /* tack on our target dir */
        /* check for match, if found, verify it is the desired one */
        if (!strnicmp(dir, table[index].dir, strlen(dir))) {
            if (verify(dir, path)) {
                chdir(path); /* yep, so jump there */
            }
        }
    }
}

```

```

        return 1;      /* signal we've done so */
    }
}
index++;
if (index < numdirs) {
    if (table[index].level != table[index-1].level) {
        if (table[index].level > table[index-1].level) {
            strcat(path, "\\"); /* get ready to go down the tree */
            if (srch_tree(level+1, path, dir)) return 1;
            /* might be several levels deep, so keep popping */
            if (level > table[index].level) return 0;
        }
        else { /* new level < old level */
            return 0; /* so go back up to parent */
        }
    }
}
return 0;
}
int verify(char *dir, char *newpath)
/*
 * Sees if the directory name is unique. If it's not,
 * the user is prompted with the full path name to verify
 * it's the one he wants. If it is unique, no prompting
 * is done.
 * Returns 1 if it's the right directory, else 0.
 */
{
    int i, cnt, c;
    for (i=0, cnt=0; i <= numdirs; i++) /* count number of matches */
        if (!strcmp(dir, table[i].dir, strlen(dir))) cnt++;
    if (cnt > 1) {
        printf("\n%s ?", newpath);
        c = getche();
        if (c == 'y' || c == 'Y') return 1;
        return 0;
    }
    return 1;
}

```

第十章 高级文件 I/O

本章补充第五章丢下的内容并讨论更高级的文件应用程序。这是通过开发在 FILE I/O 软件包之上建立的另一面向应用的软件包来完成的。

前面讨论过的面向记录的文件 I/O 只适用于定长记录。有许多应用在每个记录中需不同的数据量。某些例子是 CAD 数据库，其中图形可能会被用一节点链表或可变大小的位映像图画并存储。另一例子是联机注释程序，其中每个注释有不同的文本量。正日益变得流行的新的 hypertext (多维文本) 应用程序也需某种存储不同量数据的方法。虽然你可以用定长记录用于这些应用，但使用变长记录可使文件变得小得多。

本章介绍一个变长记录 (VLR) 软件包。并随之介绍许多文件 I/O 的实用函数，你也将看到弹出窗口和动态串软件包在起作用。本章以一个非常简单的“幻灯片”程序结束，它存储并恢复图形目标，而且使用变长记录。在稍后的一章，你将看到一个也使用这些变长记录的多维文本应用。

10.1 变长记录文件

处理变长记录文件比定长记录文件复杂得多。基本上涉及到三个方面。

- 怎样找到文件中的记录？
- 怎样插入和删除记录？
- 怎样缓解文件中记录的碎片？

我们将讨论这些问题中的每一个，还将讨论我们的 VLR 软件包怎样处理它们：

10.2 在文件中找 VLRS

在定长记录文件中找记录是容易的。用记录大小乘以记录号即可找到一个记录的字节偏移量。对 VLRS，必须用某种形式的索引。

一般地，一个 VLR 文件的索引含有一个识别每个记录的关键字和该记录在文件中的起始地址。可以有多于一组的索引，或把记录归为几类或提供多个到数据库的关键字。这些索引可以独立于数据文件本身单独存储，在 B 树数据库中常这样做，或存储在数据文件内。每种方法都有其好坏。

使索引独立于数据文件本身允许你独立于数据文件来组织和重新组织关键字。它也可以只通过创建另一个索引文件使得加多个索引集变得容易。但是，这就意味着你必须对任一给定的数据库保持更多的文件的记录，而且更重要的是一次要打开多个文件。一般，把打开文件的数目压缩到最少是个好的办法，因为从 DOS 只能得到有限数目的文件把柄。对于单独的索引途径，如果你需同时打开多个数据库，则可能很快用完全部文件把柄。

把索引与数据本身存储在一起，你就可绕过这个问题，但是在此过程中会出现别的问题。由于文件中的记录个数可以得到，所以索引的大小也可得。你可只设置一个在该文件

中可得的记录数的上限。然后索引就是定长的，且你可以在文件中（可能在开头）为索引保留一些空间。但最好不要设置这种任意的上限，那么你可以做什么呢？

一种处理这种情形的方法是把索引本身当作一个 VLR 存储。然后索引被读入即将使用的内存。

这可能加了一个索引必须能放入内存的限制，所以可处理的记录的最大数目是固定的。但是在许多应用中，这个限制实际上不成为问题。

虽然索引不严格是 VLR 软件包的一部分，但我们仍将在我们的例子中显示内部索引的办法。

10.3 插入和删除 VLRS

对定长和变长文件插入和删除记录都是个问题。许多定长记录数据库保持一个自由或未用的记录的链表。当一个记录被删除时，它就被放到“自由”链上了。当加一个新记录时，则使用自由链上的一个记录。同样的方法也用于变长记录。

VLR 文件把每个变长记录作为在链接表中联结在一起的数据块存储。（这样，我们贯穿于这个讨论将交替使用链表和记录这两个）术语。块本身是变长的且可长达 256 字节。

有一个特殊的记录，称为自由空间链表，它含有所有曾经用过但已被删除的块。当需要更多的空间时，自由空间链表的块首先被使用。如果自由空间表空了，则加字节到文件尾。当一个链表被删除，它的所有的块都被加到自由空间链表。

10.4 记录碎片

当重新使用一个变长记录时，新数据可能比原数据大或小。使记录大小增长相对容易些，只须向自由空间链表请求新块即可。但假如新数据较小，尤其是小许多呢？为不致浪费，你可能会想把尽可能多的空间送回自由空间链表。但你必须小心以免在文件中引起过度的碎片，（也即，产生大量短块）。碎片化的文件会导致许多空间浪费，因为有过多的的指针被存储，与数据本身相关。这样的文件的访问速度也慢。这个碎片问题在每个存储管理系统中都有，而不论该存储是在 RAM 中，还是存在磁盘上。甚至 DOS 文件系统也有此问题。对于我们的 VLR 文件，碎片问题以下面的方式处理。

只要可能，VLR 算法总是试图使用满 256 字节的块。当加一个新链表时，尽可能多地使用满 256 字节的块。为不浪费一个完整的 256 一字节的块，任何剩下的字节都被存储在一个较短的块中，我们只使最后一个块恰为所需的大小；另外，也可用多用的字节填充这最后一块。这给记录了一点增长的空间，而不必分配新块。填充的大小由应用决定。使用适当的大小，你可以使碎片最少。

任何时候一个记录被更新，其旧链表都被重新使用。如果新数据的大小较小，则你须从剩余块中还回一些字节到自由空间链表。为保持碎片最少，一旦一个块被创建，则它永远也不被分开。也即，你不部分地重新使用一个块，而把剩余的部分还回自由链表。只有剩下的完全未用的块才被还回。虽然这种方法浪费了一些空间，但避免了使一些块只有几个字节在它们里面。另外，如果记录被更新和变大，则最后一块里的空的空间就可马上用于扩展，并且你也许根本就不必创建任何新块。

填充最后一块有一个不幸的后果：仅仅扫描到最后一块的尾部不能判别实际数据在何

处结束。这样，数据长度必须以某种形式记录。如果你在存储 ASCII 文本，则你可简单地用 NULL 字节指示结束。甚至对二进制数据，你可能也有一个特殊的标志字节序列来指示结束。一个更通用的方法是只把记录的长度存在开头。这就是 VLR 软件包中使用的办法：VLR 记录的第一块中含有数据的实际字节数目。

10.5 VLR 文件格式

图 10.1 显示了 VLR 软件包中使用的 VLR 文件格式。

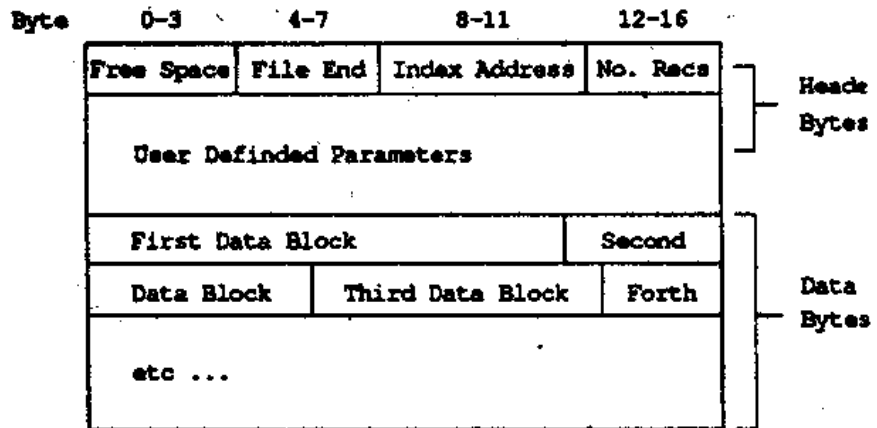


Figure 10.1 VLR file format

10.6 VLR 文件头

文件的第一部分含有一个头字节块。头部中有四个参数。free space 参数指向自由空间的开头。file end 参数指向文件的最后一个字节。这两个参数相等时，这就是个文件中无自由空间的信号，即，这个文件已满，并且如果需增加字节，则文件大小必须增大。

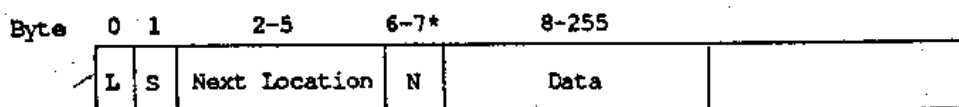
index address 参数指向文件索引的开头。这个参数是可选择的，且在你希望把索引与数据存储在同一文件中时提供。free space 和 file end 参数只用于内部使用，但 index address 参数不是 VLR 软件包的一部分，且它完全由用户处理。

最后一个参数是当前存储在文件里的记录数，主要是为方便而提供的。它完全由 VLR 软件包维持。

跟在这四个参数之后的是一片保留空间。这一空间的大小在创建一个新 VLR 文件时被指定。它是作为一个保存你想存储于文件中的附加参数的区域被提供的，并且它完全处于用户控制之下。例如，你可能想让多个索引与文件存储在一起。这个空间就可用于存储这些索引的起始地址。

10.7 VLR 记录格式

在头字节的后面是变长记录的开头。每个记录都作为一个数据块单链表存储。每个块可以是 6 字节到 256 字节长。块的结构如图 10.2 所示。



* For the first block, this is the number of actual bytes of data in the list. For all other blocks, these bytes are data.

Figure 10.2 Format of variable-length record data blocks

第一个字节含有长度 L，它表示块中的字节数，不含长度字节本身。例如，如果 L=15，则它意味着块是 16 字节长。L 的范围是从 5 到 255。如果 L=5，则意味着块中无数据；块只含有一个到下一块的链接，空块表示被浪费的空间，如果该程序使用得当，这极少在 VLR 文件中发生。

第二个字节叫 sync byte，总是具有码 0xfd。这个字节用于完整性检查。当扫描这个链表时，这个字节被检查以确保它有码 0xfd。如果它不具备此码，则它意味着文件在某种程度上被损坏了。

在 sync byte 之后是一个 4 字节整数，它是链表中下一块的地址。例如，如果它有值 0x0100，则下一块从离开文件开头 256 字节 (0x100) 处开始。如果下一位置是 0x0000，则这意味着已到了链表尾部。

在 next location 域之后是数据本身。如果这个块是链表中的第一个，则前两个字节存储整个链表的数据字节的实际数目。

10.8 VLR 软件包

既然我们已讨论了与变长记录有关的问题并已看过 VLR 文件的格式，那么就可以介绍 VLR 软件包了。程序在清单 10.1 和 10.2 中给出。软件包包含有下列例行程序：

| Routine | Use |
|-----------|-------------------------------|
| openvlr | Open/create a VLR file. |
| readhdr | Read the file header bytes. |
| writehdr | Write the file header bytes. |
| getvlr | Read in a VLR record. |
| addvlr | Add a VLR record. |
| delvlr | Delete a VLR record. |
| reuse_vlr | Reuse or update a VLR record. |

这些函数广泛应用了早先给出的错误报告和文件 I/O 软件包。VLR 软件包与 VSTR 软件包携手工作。VSTR 软件包处理的动态串对在 RAM 中存储不同数量的数据很有用，VLR 记录对在文件中存同一类型的数据很有用。

10.9 打开及创建 VLR 文件

openvlr() 函数用于打开或创建一个 VLR 文件，其原型是：

```
int openvlr(char *fname, char *access type, long rs);
```

第一个参数是文件名而第二个是存取模式，及为适用于任一标准 I/O 文件的串给出。第三个参数指明数据记录即将开始的字节位置。

文件把柄被返回。与所有标准文件 I/O 函数一样，一个 -1 值返回表明有错。注意，返回的文件把柄不是 DOS 文件把柄，而是文件 I/O 软件包使用的伪把柄。

一个创建 VLR 文件的典型调用如下：

```
/*
  Opening VLR files example (vlrex1.c)
  Must link with: popup.obj, mouse.obj, sayerr.obj,
                  fileio.obj, vstr.obj, vlr.obj
*/
#include <stdio.h> /* must include for std I/O routines */
#include "popup.h" /* all of these must be included as well */
#include "fileio.h"
#include "vstr.h"
#include "vlr.h"
void main() {
    int fh;
    init_win(); /* always, always, always do */
    init_files();
    if ((fh = openvlr("mydata.vlr", "w+b", 32L)) != -1) {
        mprintf("mydata.vlr file created successfully\r\n");
        closefile(fh);
    }
    else {
        mprintf("mydata.vlr file not created successfully\r\n");
    }
}
```

这里，“w+b”用于存取模式，它指明你想创建文件并使用二进制模式。永远也不要附加模式和总是使用二进制存取很重要：否则的话，VLR 例行程序可能不正常工作。

在这个例子中，我们已告诉 openvlr() 从字节位置 32 处开始数据记录。这基本上告诉了 VLR 软件包文件的头部大小。你可以使用你认为适合的比 16 大的任一值。四个参数 Fs（自由空间地址）、FE（文件和地址）、IA（索引地址）和 NR（记录号）存储在前 16 个字节中。在这些与数据记录开始之间的区域是未初始化的，所以如果你通过十六进制来检查该文件，则将看到一些即无害又无用的字节在那儿。由用户来决定是否初始化该区。

当最初创建一个 VLR 文件时，FS、FE、IA 和 NR 参数都被初始化了。FS 和 FE 参数都被设置为数据记录的开始。通过把它们设置为相等，它指明当前无自由空间。IA 参数被设置为 0。（它由用户在适当的时候更新），NR 参数也被设置为 0。现在 VLR 文件就可以用了。

10.10 访问头部

例行程序 readhdr() 和 writehdr() 用于访问头部的 4 个参数 FS、FE、IA 和 NR。它们有下面的原型：int readhdr(int fh, long *fs, long *fe, long *ia, long *nr);
int writehdr(int fh, long fs, long fe, long ia, long nr);

FS 和 FE 参数仅作内部使用且不应挪做它用。实际上, 在应用中使用 readhdr() 和 writehdr() 的唯一原因是存取索引地址和文件中的记录号。任何别的用户定义的在头部的参数必须直接由你自己的例行程序访问

由于这 4 个参数对文件完整性很重要, 所以任何时候 writehdr() 被调用, 文件缓冲区都要被填满。

10.11 添加和删除记录

例行程序 addvlr() 和 getvlr() 用于向文件添加一个新记录和恢复一个旧记录的数据。即将添加和恢复的数据通过一个动态串传递。对于 addvlr() 记录的新位置被返回。对于 getvlr(), 你必须传给它这个位置。这两个函数的原型是:

```
int addvlr(int fh, vstr *s, long *locn);
int getvlr(int fh, vstr *s, long locn);
```

下面的程序显示了一个存储和恢复一个记录的例子。假定记录从字节 32 处开始。

```
/*
   Example of accessing a vlr file (vlrex2.c)
   Must link with: popup.obj, mouse.obj, sayerr.obj,
                  fileio.obj, vstr.obj, vlr.obj
*/
#include <stdio.h>
#include "popup.h"
#include "fileio.h"
#include "vstr.h"
#include "vlr.h"
void main() {
    int fh;
    vstr mydata = NULLVSTR;
    long locn;
    long fs, fe, ia, nr; /* four parms in header section */
    init_win(); /* always, always, always do */
    init_files();
    /* Start a character string, add a null
       terminated message to it */
    clrvstr(&mydata, 20, sizeof(char), 5);
    vstrins(&mydata, 0, "hello world\0", 12); /* notice null byte */
    /* Open file. If successful, write, and then read the data */
    /* Also, retrieve number of records in the file */
    if ((fh = openvlr("mydata.vlr", "w+b", 32L)) != -1) {
        mprintf("mydata.vlr created successfully\r\n");
        addvlr(fh, &mydata, &locn); /* add record */
        getvlr(fh, &mydata, locn); /* read it back in */
        mprintf("First record at locn %ld contains: '%s'\r\n",
                locn, mydata.data);
        readhdr(fh, &fs, &fe, &ia, &nr);
        mprintf("Free space at %ld\r\n", fs);
    }
}
```

```

        mprintf("File ends at %ld\r\n", fe);
        mprintf("Number of records is %ld\r\n", nr);
        closefile(fh);
    }
    else {
        mprintf("mydata.vlr not created successfully\r\n");
    }
}

```

函数 `addvlr()` 取动态串中的数据, 并且如果有自由空间的话, 就用这些自由空间链表存储数据。如果无自由空间, 则数据被附在文件后面。在两种情形下, 记录的起始地址都返回。当用 `getvlr()` 读一个访问记录时, 必须指定起始地址。典型地, 地址被保存在与文件相关联的某索引表中。在这个简单例子中, 我们只是把它保存在一个局部变量中。

10.12 确定 VLRS 的类型

当一个动态串被写入一个 VLR 文件时, 该串当前长度就被乘以元素的大小以确定串中字节的实际数目。这个数与数据存储在一起。当读取一个变长记录时, 相关联的串先被复位 (把当前长度设置为 0), 之后数据被读入。随后存储的数据长度被元素大小除, 以确定串长。

它的含意是存储的数据的“类型”由所传递的动态串的“类型”决定。所用的确定类型的信息仅仅是串中一个元素的大小。这样, 用同一“类型”的串用于读写一个变长记录就很重要。具体地讲, 如果元素的大小不是所写的字节个数的倍数, 则将返回一个错误的长度, 并且数据将丢失。如果你想以原始的二进制形式访问这些数据, 则使用具有元素大小为一个字符的串。

```

int reuse_vlr(int fh, long locn, vstr *w, int offset, int use_fs);
int delvlr(int fh, long locn);

```

10.13 更新 VLRS

VLR 文件的记录由 `reuse_vlr()` 函数更新, 由 `delvlr()` 函数删除。它们的原型是:

删除一个记录是简单的。只须把该记录的块加到自由空间链表的前面, 并把记录计数器减 1 即可。

`reuse_vlr()` 函数是例行程序中最长最复杂的。它先取存储于 `w` 中的动态串并在位置 `locn` 扫描旧的变长记录, 填满它的块。如果有任何剩下的完全未使用的块, 则被返回到自由空间链表。最后一个部分块的剩余字节保持原样。

如果旧记录不够大, 则使用自由空间链表。这由在自由空间链表被“重使用”时递归调用 `reuse_vlr()` 来实现。当递归时, 参数 `offset` 指向将写的串的剩余字节。参数 `use_fs` 指示该例行程序重新使用自由空间。

如果自由空间链表用完了, 则调用内部例行程序 `extend_file()` 把字节附于文件尾。这个例行程序填充所需的 256 个字节并用多余的字节填充最后一块。外部变量 `padsize` 可以设置为任意理想的值。它的系统设置值为 10 字节。

下例显示了一个 VLR 文件被创建及一个记录被添加、更改、最后被删除:

```

/*
More examples of vlr file I/O (vlrex3.c)
Must link with: popup.obj, mouse.obj, sayerr.obj,
               fileio.obj, vstr.obj, vlr.obj
*/

#include <stdio.h>
#include "popup.h"
#include "fileio.h"
#include "vstr.h"
#include "vlr.h"

void main() {
    int fh;
    vstr mydata = NULLVSTR;
    long locn;
    long fs, fe, ia, nr; /* four parms in header section */
    init_win();          /* always, always, always do */
    init_files();
    /* start a character string, add a
       null terminated message to it */
    clrvstr(&mydata, 20, sizeof(char), 5);
    vstrins(&mydata, 0, "hello world\0", 12); /* notice null byte */
    /* open file. If successful, write, and then read the data */
    /* Also, retrieve number of records in the file */
    if ((fh = openvlr("mydata.vlr", "w+b", 32L)) != -1) {
        mprintf("mydata.vlr created successfully\r\n");
        addvlr(fh, &mydata, &locn); /* add record */
        getvlr(fh, &mydata, locn); /* read it back in */
        mprintf("First record at locn %ld contains: '%s'\r\n",
                locn, mydata.data);
        readhdr(fh, &fs, &fe, &ia, &nr);
        mprintf("Free space at %ld\r\n", fs);
        mprintf("File ends at %ld\r\n", fe);
        mprintf("Number of records is %ld\r\n", nr);
        /* update record, then read it back in */
        vstrins(&mydata, 6, "cruel ", 6);
        mprintf("adding %s\r\n", mydata.data);
        reuse_vlr(fh, locn, &mydata, 0, 0);
        getvlr(fh, &mydata, locn);
        mprintf("Record now contains: '%s'\r\n", mydata.data);
        readhdr(fh, &fs, &fe, &ia, &nr);
        mprintf("Free space at %ld\r\n", fs);
        mprintf("File ends at %ld\r\n", fe);
        mprintf("Number of records is %ld\r\n", nr);
        /* delete the record */
        delvlr(fh, locn);
        mprintf("Record now deleted\r\n");
        readhdr(fh, &fs, &fe, &ia, &nr);
        mprintf("Free space at %ld\r\n", fs);
    }
}

```

```

    nprintf("File ends at %ld\r\n", fe);
    nprintf("Number of records is %ld\r\n", nr);
    closefile(fh);
}
else {
    nprintf("mydata.vlr not created successfully\r\n");
}
}

```

当执行时，这个程序将打印出：

```

New vlr file created successfully
First record at locn 32 contains: 'hello world'
Free space at 62
File ends at 62
Number of records is 1
adding hello cruel world
Record now contains: 'hello cruel world'
Free space at 62
File ends at 62
Number of records is 1
Record now deleted
Free space at 32
File ends at 62
Number of records is 0

```

注意最后两个参数 offset 和 use-fs 是怎样处理的。对一个到 reuse-vmr() 的外部调用，这两个参数应总是 0。只有内部递归调用才能把这些参数设置为非零值。

10.14 建立一个内部 VLR 索引

现在我们将告诉你如何为一组变长记录建一索引及随后如何把该索引作为一个记录本身存储。在这个例子中，索引将不存储任何关键字，只存储记录地址。最简单的方法是把索引建成一个长整数的动态数组。所有这些记录都被加到该 VLR 文件，而它们的起始地址被记录在该整数串中。然后该串被写到该文件，而索引地址本身被加到头块里。

```

/*
Example of an internal vlr file index (vlrex4.c)
Must link with: popup.obj, mouse.obj, sayerr.obj,
               fileio.obj, vstr.obj, vlr.obj
*/
#include <stdio.h>
#include "popup.h"
#include "fileio.h"
#include "vstr.h"
#include "vlr.h"
void main() {
    int fh;
    vstr rec1data = NULLVSTR;
    vstr rec2data = NULLVSTR;
    vstr rec3data = NULLVSTR;
    vstr buffer   = NULLVSTR;

```



```

vstr index    = NULLVSTR;
long locn;
long fs, fe, ia, nr;    /* four parms in header section */
init_win();    /* always, always, always do */
init_files();
/* make the null terminated character strings */
clrvtstr(&rec1data, 20, sizeof(char), 5);
vstrins(&rec1data, 0, "This is record one\0",19);
clrvtstr(&rec2data, 20, sizeof(char), 5);
vstrins(&rec2data, 0, "This is record two\0",19);
clrvtstr(&rec3data, 20, sizeof(char), 5);
vstrins(&rec3data, 0, "This is the third record\0",25);
clrvtstr(&buffer, 20, sizeof(char), 5); /* temp buffer */
/* and then the index itself */
clrvtstr(&index, 10, sizeof(long), 5);
/* open file.  If successful, add the records, and record the
addresses in the index table.
*/
if ((fh = openvlr("mydata.vlr","w+b", 32L)) != -1) {
    mprintf("New vlr file created successfully\n");
    addvlr(fh, &rec1data, &locn);
    vstrcat(&index, &locn);
    addvlr(fh, &rec2data, &locn);
    vstrcat(&index, &locn);
    addvlr(fh, &rec3data, &locn);
    vstrcat(&index, &locn);
    /* Add the index itself to the file, then update the
header bytes to point to index
*/
    addvlr(fh, &index, &locn);
    readhdr(fh, &fs, &fe, &ia, &nr);
    writehdr(fh, fs, fe, locn, nr);
    /* Now, read in the index, and then each record */
    readhdr(fh, &fs, &fe, &ia, &nr); /* get index address */
    getvlr(fh, &index, ia);
    locn = ((long *) (index.data))[0];
    getvlr(fh, &buffer, locn);
    mprintf("First record at locn %ld contains: '%s'\n",
        locn, buffer.data);
    locn = ((long *) (index.data))[1];
    getvlr(fh, &buffer, locn);
    mprintf("Second record at locn %ld contains: '%s'\n",
        locn, buffer.data);
    locn = ((long *) (index.data))[2];
    getvlr(fh, &buffer, locn);
    mprintf("Third record at locn %ld contains: '%s'\n",
        locn, buffer.data);
}

```

```

    mprintf("Free space at %ld\r\n", fs);
    mprintf("File ends at %ld\r\n", fe);
    mprintf("Number of records is %ld\r\n", nr);
    closefile(fh);
}
else {
    mprintf("File not created successfully\r\n");
}
}

```

当执行时，这程序打印出：

```

New vlr file created successfully
First record at locn 32 contains: 'This is record one'
Second record at locn 69 contains: 'This is record two'
Third record at locn 106 contains: 'This is the third record'
Free space at 179
File ends at 179
Number of records is 4

```

注意索引记录在记录计数器中是怎样被计算的。

10.15 例子：一个简单的幻灯片程序

这一节给出一个使用 VLR 文件的扩展的例子。给出的是两个程序，一个创建一串图形物体的“幻灯片”，一个放映这些幻灯片。幻灯片创建程序在清单 10.3 中给出，幻灯片放映程序在清单 10.4 中给出。检查它们时，你将会发现这两个程序非常相似，使用基本相同的程序。

幻灯创建程序允许你在一个框架里创建图形多边形物体。然后你可以存储这些物体并创建别的物体，产生一系列框架。程序使用鼠标器（如果你没有的话我们向你道歉）画图 and 选择命令。图 10.3 显示了一个样例屏幕。通过建立一系列的线段在长方形区域内画出这些物体（打开修剪功能）。在框架内任何时候按一下左鼠标器按钮，都将画出一条线段。

命令是通过指向理想的项，按一下左鼠标器按钮进行选择。这些命令有：

| Command | Use |
|---------|--|
| Store | Stores the current slide |
| Fill | Makes a closed polygon by connecting the first point with the last, and then filling the polygon with red crosshatches |
| Clear | Clears the slide, and resets the object's point list |
| Quit | Closes the slide file and quits the program |

为了画一物体，只需把鼠标移到理想的位置并按一下左鼠标器按钮即可。一条线将从前一个点画到该新点。继续此过程直到你希望完成这个多边形为止。然后选择 FILL 命令。你可以在一个幻灯片上画多个多边形。在完成之后，选 STORE 命令存储该幻灯片。如果在制幻灯片时出了错，可选 CLEAR 清除屏幕和复位多边形链表。当所有的都完成后，选择 QUIT 关闭文件并退出程序。现在你的幻灯片系列都存入一个 VLR 文件了。

"Store Fill Clear Quit"

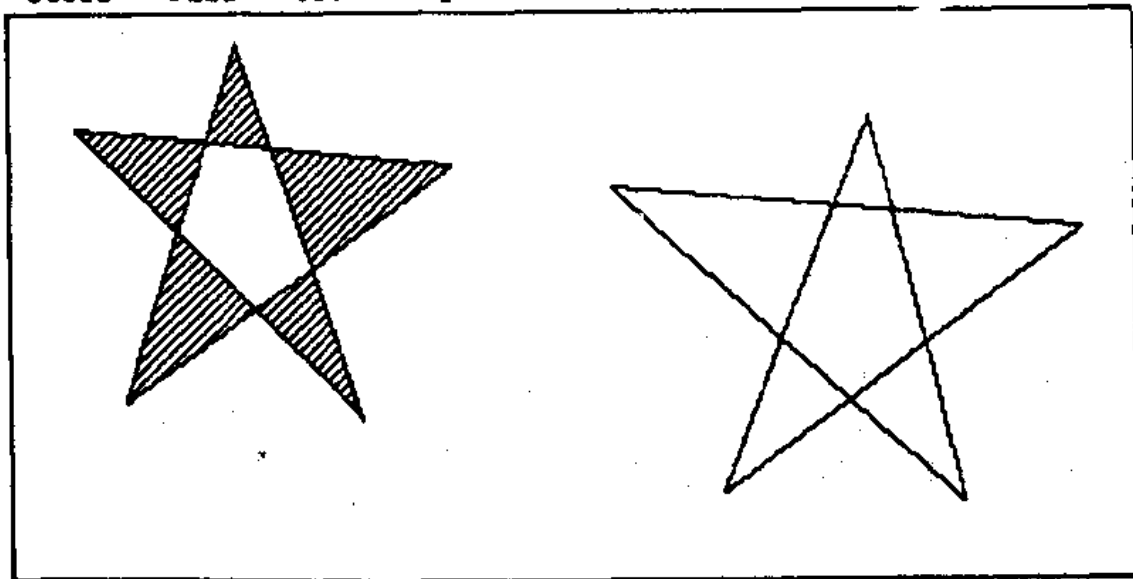


Figure 10.3 Using the polygon slide show creation program

"First Last Next Prev Quit"

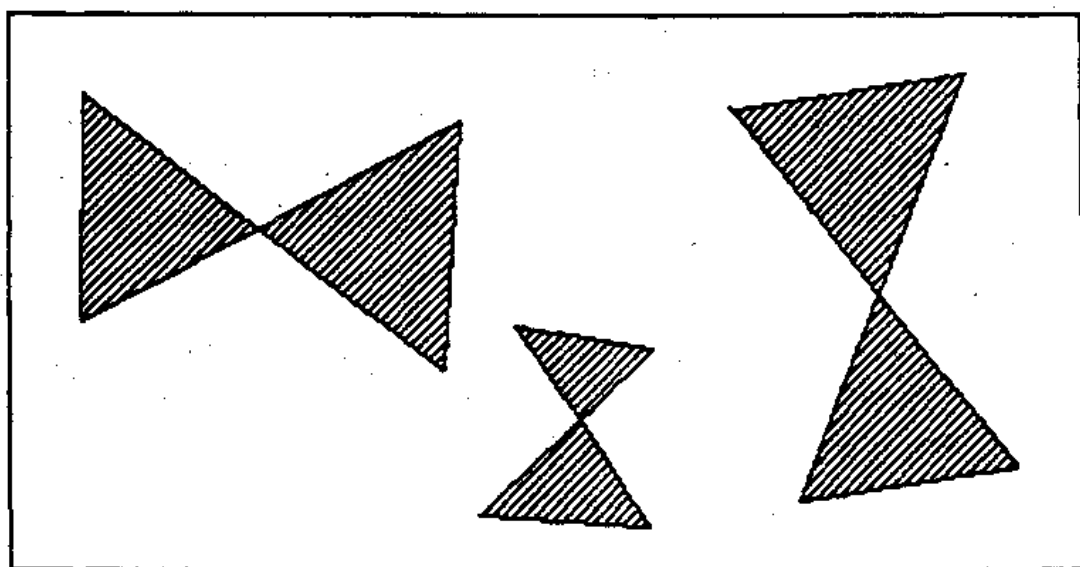


Figure 10.4 Using the polygon slide show browser program

为观察这些幻灯片，可执行清单 10.4 中的程序。图 10.4 给出了一个样例屏幕。FIRST、LAST、NEXT 和 PREVIOUS 命令全都移到了相应的幻灯片上。右手角上是当前幻灯片的号及幻灯片的总数。

两个程序都使用一个名为 objects.vlr 的在当前目录中的文件。幻灯片创建程序创建此文件。幻灯放映程序为只读存取打开它。

这些样例程序中有一个缺点。VLR 软件包使用文件 I/O 软件包，而后者又依次使用错误报告软件包。而错误报告软件包又依次调用弹出窗口软件包。弹出窗口只用于文本模式，所以如果在图形模式下出现了文件错误，则你只能得到一些无用的信息而非一个错

误窗口。由于软件包的模块化，你所需做的全部只是创建一个用于图形模式的不同的 sayerr() 函数，适当改动头文件，并连接入 sayerr() 的图形版本而非文本版本。这留给你做为一个练习。

Listing 10.1 Source code for file "vlr.h"

```
/* Variable length record toolkit header file (vlr.h) */
extern int openvlr(char *fname, char *access_type, long rs);
extern int readhdr(int fh, long *fs, long *fe, long *ita, long *nl);
extern int writehdr(int fh, long fs, long fe, long ita, long nl);
extern int getvlr(int fh, vstr *s, long locn);
extern int reuse_vlr(int fh, long locn, vstr *w, int offset, int use_fe);
extern int delvlr(int fh, long locn);
extern int advlr(int fh, vstr *w, long *locn);
extern int padsize;
```

Listing 10.2 Source code for file "vlr.c"

```
/* *****
 * Variable length record toolkit (vlr.c)
 * ***** */
#include <stddef.h>
#include <io.h>
#include <stdio.h>
#include <fcntl.h>
#include <string.h>
#include "popup.h"
#include "vstr.h"
#include "fileio.h"
#include "vlr.h"
/* our synchronization code for integrity checking */
#define SYNC_BYTE 253
/* "NULL" link code, and data padding constant */
#define NOLINK 0
/* Each block has 6 bytes of header information, can hold 250 data bytes */
#define BLK_SIZE 256
#define DATA_SIZE 250
int padsize = 10; /* default padding to 10 bytes */
static int getvltrelem(int fh, long pos, long *nxtlocn,
                      unsigned char *datablk);
static int extend_file(int fh, vstr *s, long *locn, int offset);
int openvlr(char *fname, char *access_type, long rs)
/*
 * Opens a variable length record file, using access_type to control
 * the file access mode (see openfile function in fileio.c).
 * NOTE: DO NOT use any append mode, and be sure to use binary mode !!!
 * If the file is created, then it sets the records to start at rs.
 * If just opening, then this parm is ignored.
 * Returns new file entry, or -1 if an error occurred.
 */
{
    int fh;
    long zero = 0L;
    /* Open for byte level access */
    if ((fh = openfile(fname, access_type, 1)) != -1) {
```

```

        if (*access_type == 'w') { /* ie. we're creating */
            outbytes(fh, 0L, (unsigned char *)&rs, 4); /* freespace */
            outbytes(fh, 4L, (unsigned char *)&rs, 4); /* file_end */
            outbytes(fh, 8L, (unsigned char *)&zero, 4); /* index_rec */
            outbytes(fh, 12L, (unsigned char *)&zero, 4); /* nrecs */
        }
    }
    return fh;
}

int readhdr(int fh, long *fs, long *fe, long *ita, long *nl)
/*
    Reads the header info: FreeSpace, FileEnd, IndxTabAddr,
    and number of Lists.

    Returns = 0 if successful, else, returns a -1
*/
{
    unsigned char datablk[16];
    int rval;
    rval = inbytes(fh, 0L, datablk, 16);
    memcpy((char *)fs, datablk, 4);
    memcpy((char *)fe, datablk+4, 4);
    memcpy((char *)ita, datablk+8, 4);
    memcpy((char *)nl, datablk+12, 4);
    if (rval < 0) return -1; return 0;
}

int writehdr(int fh, long fs, long fe, long ita, long nl)
/*
    Writes the header info: FileStart, FileEnd, IndxTabAddr,
    and number of Lists.

    Returns = 0 if successful, else, returns a -1
*/
{
    unsigned char datablk[16];
    int rval;
    memcpy(datablk, (char *)&fs, 4);
    memcpy(datablk+4, (char *)&fe, 4);
    memcpy(datablk+8, (char *)&ita, 4);
    memcpy(datablk+12, (char *)&nl, 4);
    rval = outbytes(fh, 0L, datablk, 16);
    if (rval < 0) return -1;
    /* Flush the buffer to insure critical data is written.
       Note we must access stream pointer as stored in file table.
    */
    if (fflush(ft[fh].fp)) {
        sayerr(FERR, "Writing vlr header data in %s\r\n", ft[fh].name);
        return -1;
    }
    return 0;
}

static int getvlrelem(int fh, long pos, long *nxtlocln,
    unsigned char *datablk)
/*
    * Reads up to 256 bytes of a block of a VLR. Thus, datablk should have
    * at least 256 bytes of room in it. Returns number of bytes read,

```

```

* or returns -1 if an error has occurred.
*
* It determines the number of bytes to read from the first byte of block,
* which gives the length of the block, excluding the byte itself.
*/
{
    int bytesread;
    inbytes(fh, pos, datablk, 1); /* read length byte */
    bytesread = inbytes(fh, -1L, datablk+1, *datablk);
    if (datablk[1] != SYNC_BYTE) {
        sayerr(SERR, "Sync byte error at %ld\r\n", pos);
        return -1;
    }
    else {
        memcpy((char *)nextlocn, datablk+2, 4);
    }
    return bytesread;
}

int getvlnr(int fh, vstr *s, long pos)
/*
* Reads in a variable-length record into virtual string s from file fh,
* at pos'n pos. Returns number of bytes in the record, else returns
* -1 if an error occurred.
*/
{
    unsigned char datablk[256];
    unsigned int offset, actual_len, data_size, data_indx;
    int nblks;
    s->currlen = 0; /* reset string length */
    offset = 0;
    nblks = 0;
    while(pos != 0L) {
        if ((getvirelem(fh, pos, &pos, datablk)) == -1) {
            /* give up and return what you have */
            s->currlen = offset / s->esize;
            return -1;
        }
        else { /* add data to end of virtual string */
            if (++nblks == 1) { /* if first block, then get actual data length */
                memcpy(&actual_len, datablk+6, 2);
                if (actual_len % s->esize) { /* data types wrong !! */
                    sayerr(SERRF, "Data length doesn't jive with vstr type.\r\n"
                        "Data len = %u, element size = %u\r\n",
                        actual_len, s->esize);
                    return -1;
                }
            }
            /* Might have to extend the string. */
            if ((actual_len / s->esize) > s->dimlen) {
                if (!redimvstr(s, actual_len / s->esize)) return -1;
            }
            data_indx = 8;
            data_size = *datablk-7; /* compute amt of data in blk */
        }
        else {
            data_indx = 4;
            data_size = *datablk-5;
        }
    }
}

```

```

    }
    memcpy((char *) (s->data)+offset, datablk+data_idx, data_size);
    offset += data_size;
}

/* Set current length (in units of elem_size) to the length given
   in the data itself. Hopefully it divides equally.
*/
s->currlen = actual_len / s->esize;
return actual_len;
}

static int extend_file(int fh, vstr *s, long *locn, int offset)
/* Extends the file by adding the data in variable string s to the file,
   * starting with the bytes at the specified offset. The locn of the
   * start of the data is returned in *locn.
   * Function returns 0 if successful, or -1 otherwise.
*/
{
    long fs, fe, ita, nrecs, link;
    unsigned int i, nb, nl, len;
    unsigned char datablk[256];
    /* Convert length to no. of chars, and subtract offset */
    /* but if on first block, leave room for two length bytes */
    len = s->currlen * s->esize - offset;
    if (offset == 0) len += 2;
    if (readhdr(fh, &fs, &fe, &ita, &nrecs) == -1) return -1;
    nb = len / DATA_SIZE;
    nl = len - nb*DATA_SIZE;
    link = fe; *locn = fe;
    /* Fill up full blocks first. If we exhaust the data then link the
       * last block to nowhere.
    */
    if (nb) {
        for (i = 1; i <= nb; i++) {
            if (i == nb && nl == 0) link = 0; else link += BLK_SIZE;
            memcpy(datablk+2, (char *)&link, 4);
            datablk[0] = BLK_SIZE-1;
        }
        datablk[1] = SYNC_BYTE;
        if (offset == 0) {
            /* stuff in data length bytes on first blk */
            len -= 2;
            memcpy(datablk+6, &len, 2);
            memcpy(datablk+8, ((char *) (s->data)), DATA_SIZE-2);
            offset += DATA_SIZE - 2;
        }
        else {
            memcpy(datablk+6, ((char *) (s->data))+offset, DATA_SIZE);
            offset += DATA_SIZE;
        }
        if (outbytes(fh, fe, datablk, BLK_SIZE) == -1) return -1;
        fe += BLK_SIZE;
    }

    /* For the last non-full block, link it to nowhere, and add up to
       * padsize extra bytes to it, (to help minimize fragmentation when
       * reusing the block).
    */

```

```

*/
if (nl) {
    link = NOLINK;
    nl += padsize;
    if (nl > DATA_SIZE) nl = DATA_SIZE;
    datablk[0] = nl+5;
    datablk[1] = SYNC_BYTE;
    memcpy(datablk+2, (char *)&link, 4);
    if (offset == 0) {
        /* stuff in data length bytes on first blk */
        len = 2;
        memcpy(datablk+6, &len, 2);
        memcpy(datablk+8, ((char *) (s->data)), nl-2);
    }
    else {
        memcpy(datablk+6, ((char *) (s->data))+offset, nl);
    }
    if (outbytes(fh, fe, datablk, nl+6) == -1) return -1;
    fe += nl + 6;
}
/* Set freespace = file_end, which means no freespace */
if (writehdr(fh, fe, fe, ita, nrecs) == -1) return -1;
return 0; /* <<< success >>> */
}

int reuse_vlr(int fh, long locn, vstr *w, int offset, int use_fs)
/*
 * Reuses the vlr starting at byte pos'n "locn" in file fh, by storing the
 * data in variable string w at the specified offset. If we run out of
 * room the vlr list, then the list is extended by first using up
 * freespace, and if we run out of that, bytes are appended to the file.
 * If we do have to use freespace, then this routine is called recursively
 * with "locn" set to the freespace itself, the new vstr offset, and the
 * "use_fs" flag set to 1. This flag tells us whether to try using
 * freespace or to append to the file when we need more room.
 *
 * NOTE: Never call reuse_vlr directly with offset != 0. This condition
 * is meant for recursive calls only. Always use offset = 0.
 *
 * If we have any blocks left over from the old vlr, they are put on the
 * freespace list. Note however, that the blocks themselves are never split
 * up once they are made.
 *
 * Returns 0 if successful, -1 otherwise.
 */
{
    unsigned int nb, nu, ls, len;
    unsigned char datablk[256];
    long oldfs, nextlocn, fs, fe, ita, nrecs, dmy;
    /* Convert to number of chars. If on first block, */
    /* then add room for two bytes of length code */
    len = w->currlen * w->esize;
    if (offset == 0) len += 2;
    if (readhdr(fh, &fs, &fe, &ita, &nrecs) == -1) return -1;
    /******
     * Walk thru each block of vlr, reusing it's space until
     * we either run out of room, or have added all the data.
     * *****/

```



```

do {
    if (getvirelem(fh, locn, &nxtlocn, datablk) == -1) return -1;
    nb = *datablk;
    if (nb <= 5) {
        locn = nxtlocn; /* we have block with no room in it, so move on */
    }
    else {
        /* we have block with data in it, so reuse it */
        nu = nb-5;
        ls = len - offset;
        if (nu >= ls) { /* what's left of our string fits in block */

            /* what's left of our string fits in block, so null at nxtlocn
            since at end, and then copy the data into the block. If
            we're just starting the vlr, then add the length bytes first.
            */
            memset(datablk+2, 0, 4); /* null out nxtlocn cause at end */
            if (offset == 0) { /* if at start of block, add length bytes */
                len -= 2;
                memcpy(datablk+6, &len, 2); /* add data length code */
                memcpy(datablk+8, ((char *) (w->data)), ls-2); /* and data */
            }
            else {
                memcpy(datablk+6, ((char *) (w->data))+offset, ls);
            }
        }
        if (outbytes(fh, locn, datablk, *datablk+1) == -1) return -1;
        /* Now, save the old free space pointer, and point freespace
        to the start of any remaining blocks in the chain. If
        we were reusing freespace and we're at it's end, well,
        it's out of space now, so set it equal to the file end.
        */
        oldfs = fs;
        if (nxtlocn) {
            fs = nxtlocn;
        }
        else {
            /* if we were using freespace, it's */
            if (use_fs) fs = fe; /* out of room now */
        }
        /* Now, if not already using freespace, and there's links
        left in the chain, scan down the chain to the end and
        link the end to the old freespace. If fs == fe, then there
        really was no freespace, so just leave the end link's
        nxtlocn null. Then, break out of loop
        */
        if (!use_fs && (nxtlocn != 0)) {
            do {
                locn = nxtlocn;
                if (getvirelem(fh, locn, &nxtlocn, datablk) == -1) return -1;
            } while (nxtlocn);
            if (oldfs != fe) { /* hook up end of chain with fs */
                memcpy(datablk+2, (char *)&oldfs, 4);
                if (outbytes(fh, locn, datablk, *datablk+1) == -1) return -1;
            }
        }
        writehdr(fh, fs, fe, ita, nrecs);
        break; /* <<< exit loop >>> */
    }
}
else {

```

```

/* String doesn't fit into block. So add what you can.
   If at the start of the vlr, then add the length
   bytes to the front
*/
if (offset == 0) { /* add len bytes to first block */
    len = 2;
    memcpy(datablk+6, &len, 2);
    memcpy(datablk+8, ((char *) (w->data)), nu-2);
    offset += nu-2;
}
else {
    memcpy(datablk+6, ((char *) (w->data))+offset, nu);
    offset += nu;
}
}
if (nxtlocln) { /* write out new block */
    if (outbytes(fh, locln, datablk, *datablk+1) == -1) return -1;
    locln = nxtlocln;
}
} while (nxtlocln);
/***** end of block walk loop *****/
if (nu < 16) {
    /* We still have some data left, so we need to extend the
     * vlr. Test to see whether to use freespace or append to
     * file, and set our link to the file-end, or to freespace.
     * Note that even if it says use freespace, there may not
     * be any freespace, but this is checked later.
     */
    if (use_fs) memcpy(datablk+2, (char *)&fe, 4);
    else memcpy(datablk+2, (char *)&fs, 4);
    if (outbytes(fh, locln, datablk, *datablk+1) == -1) return -1;
    /* We may want to append to file, or may have to cause there
     * really isn't any freespace. Otherwise, recursively reuse the
     * freespace vlr.
     */
    if (use_fs || (fs==fe)) {
        if (extend_file(fh, w, &dmy, offset) == -1) return -1;
    }
    else {
        if (reuse_vlr(fh, fs, w, offset, 1) == -1) return -1;
    }
}
return 0;
}

int delvlr(int fh, long locln)
/*
 * Deletes the vlr which starts at byte pos'n "locln" in file fh.
 * The record is deleted by placing it on the freespace list.
 * Returns 0 if successful, -1 otherwise.
*/
{
    unsigned char datablk[256];
    long newfs, fs, fe, ita, nrecs, nxtlocln;
    /* get freespace and file end pointer and number of records */
    if (readhdr(fh, &fs, &fe, &ita, &nrecs) == -1) return -1;

```

```

newfs = locn; /* vlr's address will be new freespace address */
/* scan the vlr's blocks until last one found or there's an error */
nxtlocn = locn;
do {
    locn = nxtlocn;
    if ((getvirelem(fh, locn, &nxtlocn, datablk)) == -1) return -1;

    } while (nxtlocn);
/* If we have freespace, hook up last block of our vlr to it */
if (fs != fe) {
    memcpy(datablk+2, (char *)&fs, 4);
    if (outbytes(fh, locn, datablk, *datablk+1) == -1) return -1;
}
/* write out new freespace pointer and number of records */
if (writehdr(fh, newfs, fe, ita, nrecs-1) == -1) return -1;
return 0;
}

int addvlr(int fh, vstr *w, long *locn)
/*
 * Takes the data stored in virtual string w and adds it as a
 * vlr to the file fh, and returns the locn of the vlr
 * in *locn. If there is freespace, that is used before appending
 * to the file. Returns 0 if successful, -1 otherwise.
 */
{
    long oldfs, fs, fe, ita, nrecs;
    if (readhdr(fh, &oldfs, &fs, &ita, &nrecs) == -1) return -1;
    /* either append bytes to the file, or use up freespace first */
    if (oldfs == fe) {
        if (extend_file(fh, w, &fs, 0) == -1) return -1;
    }
    else {
        if (reuse_vlr(fh, oldfs, w, 0, 1) == -1) return -1;
    }
    /* update number of records in file */
    if (readhdr(fh, &fs, &fe, &ita, &nrecs) == -1) return -1;
    if (writehdr(fh, fs, fe, ita, nrecs+1) == -1) return -1;
    *locn = oldfs; /* our vlr's addr. was the old freespace addr. */
    return 0;      /* successful completion */
}

```

Listing 10.3 Source code for file "vlrex5.c"

```

/*****
 * Polygon slide show creation example (vlrex5.c)
 *
 * Must link with: popup.obj, mouse.obj, sayerr.obj,
 *                fileio.obj, sayerr.obj, vstr.obj,
 *                vlr.obj, graphics.lib
 *****/
#include <graphics.h>
#include <stdio.h>
#include <string.h>
#include <process.h>
#include "mouse.h"
#include "vstr.h"

```

```

#include "popup.h"
#include "fileio.h"
#include "vlr.h"
#define rectxul 0
#define rectyul 32
typedef struct {
    int x,y;
} point;
typedef struct {
    int x, y;
    char label[10];
} button;
button draw_menu[] = {
    { rectxul,      rectyul-16, "Store" },
    { rectxul+80,   rectyul-16, "Fill"  },
    { rectxul+160,  rectyul-16, "Clear" },
    { rectxul+240,  rectyul-16, "Quit " }
};
int num_draw_menu_items = sizeof(draw_menu) / sizeof(button);
int mouse_on_entry(button *menu, int nitems);
void main()
{
    int gd=DETECT, gm=0, rectxlr, rectylr;
    unsigned int k;
    vstr v = NULLVSTR, index = NULLVSTR;
    point p = {0,0};
    int first_time = 1, i, command, done, starting_point;
    int fh;
    long locn, fs, fa, ita, nr;
    init_win();
    init_files();
    if ((fh = openvlr("objects.vlr", "w+b", 32L)) == -1) {
        printf("Error creating objects.vlr\n");
        exit(1);
    }
    initgraph(&gd, &gm, "");
    init_mouse(MOUSE_NEEDED, gd, gm);
    rectxlr = getraxx(); rectylr = getmaxy();
    clrstr(&v, 30, sizeof(point), 5);
    clrstr(&index, 20, sizeof(long), 5);
    /* draw menu */
    mouse_off(1);
    for(i=0; i < num_draw_menu_items; i++) {
        outtextxy( draw_menu[i].x, draw_menu[i].y, draw_menu[i].label );
    }
    mouse_on(1);
    /* draw window, and setup window coordinates */
    mouse_off(1);
    rectangle(rectxul, rectyul, rectxlr, rectylr);
    mouse_on(1);
    setviewport(rectxul+1, rectyul+1, rectxlr-1, rectylr-1, 1);
    setfillstyle(8, 4); /* cross hatch fill, red color */
    done = 0;
    starting_point = 0;
    do {
        while(! (k = mouse_trigger(0)));
        if (k == LEFT_MOUSE_REL) {

```

```

command = mouse_on_entry(draw_menu, num_draw_menu_items);
switch(command) {
case 0:
    /* store object, and remember its address */
    addv1r(fh, &v, &locn);
    vstrcat(&index, &locn);
    /* clear object */
    mouse_off(1);
    clearviewport();
    mouse_on(1);
    v.currlen = 0;
    first_time = 1;
    break;
case 1:
    /* make a closed polygon boundary */
    /* by adding starting point to end */
    /* then filling in the polygon */
    vstrcat(&v, ((point *)v.data) + starting_point);
    mouse_off(1);
    /* note: this redraws ALL of the polygons in the string */
    fillpoly(v.currlen, (int far *) (v.data));
    mouse_on(1);
    first_time = 1;
    break;
case 2: /* clear window, reset point string */
    mouse_off(1);
    clearviewport();
    mouse_on(1);
    v.currlen = 0;
    first_time = 1;
    break;
case 3: /* finished drawing, so write out index, close file */
    done = 1;
    addv1r(fh, &index, &locn);
    readhdr(fh, &fs, &fe, &ita, &nrr);
    writehdr(fh, fs, fe, locn, nrr);
    closefile(fh);
    break;
default: /* draw line segments */
    if (first_time) {
        mouse_off(1);
        putpixel(mouse_grph_x - rectxul,
                 mouse_grph_y - rectyul, 15);
        mouse_on(1);
    }
    else {
        mouse_off(0);
        line(p.x, p.y,
             mouse_grph_x - rectxul, mouse_grph_y - rectyul);
        mouse_on(0);
    }
    p.x = mouse_grph_x - rectxul;
    p.y = mouse_grph_y - rectyul;
    vstrcat(&v, &p);
    if (first_time) starting_point = v.currlen - 1;
    first_time = 0;
}

```

```

    }
} while (!done):
mouse_reset();
closegraph();
}
int mouse_on_entry(button *menu, int nitems)
/* Returns button number that mouse is currently on, or returns -1 */
{
    int i;
    for(i=0; i < nitems; i++) {
        if( mouse_in_box(1,
            menu[i].x, menu[i].y,
            menu[i].x + strlen(menu[i].label)*8 - 1, menu[i].y + 7) )
            return i;
    }
    return -1;
}

```

Listing 10.4 Source code for file "vlrex6.c"

```

/*****
 * Polygon slide show browsing program (vlrex6.c)
 *
 * Must link with: popup.obj, mouse.obj, sayerr.obj,
 *                fileio.obj, sayerr.obj, vstr.obj,
 *                vlr.obj, graphics.lib
 *****/
#include <graphics.h>
#include <stdio.h>
#include <string.h>
#include <process.h>
#include "mouse.h"
#include "vstr.h"
#include "popup.h"
#include "fileio.h"
#include "vlr.h"
#define rectxul 0
#define rectyul 32
typedef struct {
    int x,y;
} point;
typedef struct {
    int x, y;
    char label[10];
} button;
button show_menu[] = {
    { rectxul,    rectyul-16, "First" },
    { rectxul+50, rectyul-16, "Last" },
    { rectxul+100, rectyul-16, "Next" },
    { rectxul+150, rectyul-16, "Prev" },
    { rectxul+200, rectyul-16, "Quit" }
};
int num_show_menu_items = sizeof(show_menu) / sizeof(button);
vstr v = NULLVSTR, index = NULLVSTR;

```

```

char buffer[20];
int mouse_on_entry(button *menu, int nitems);
char far *bios_video_area = (char far *)0x00400049L;
void main()
{
    int gd=DETECT, gm=0, i, command, obj, fh, rectxlr, rectylr;
    unsigned int k;
    point p;
    long fs, fe, ita, nr;
    init_win();
    init_files();
    if ((fh = openvtr("objects.vtr","rb",32L)) == -1) {
        printf("Error opening file objects.vtr\n");
        exit(1);
    }
    initgraph(&gd,&gm,"");
    init_mouse(MOUSE_NEEDED, gd, gm);
    rectxlr = getmaxx(); rectylr = getmaxy();
    clrscr(&v,30,sizeof(point),5);
    clrscr(&index,20,sizeof(long),5);
    /* read in index */
    readhdr(fh, &fs, &fe, &ita, &nr);
    getvtr(fh, &index, ita);
    /* draw menu */
    mouse_off(1);
    for(i=0; i < num_show_menu_items; i++) {
        outtextxy(show_menu[i].x, show_menu[i].y, show_menu[i].label);
    }
    mouse_on(1);
    /* draw window, and setup window coordinates */
    mouse_off(1);
    rectangle(rectxul, rectyul, rectxlr, rectylr);
    mouse_on(1);
    setviewport(rectxul+1,rectyul+1,rectxlr-1,rectylr-1,1);
    setfillstyle(8,4); /* cross hatch fill, red color */
    command = 0; obj = 0;
    do {
        /* change viewports temporarily, so we can clear status field */
        setviewport(rectxlr-60,rectyul-16,rectxlr,rectylr-8,1);
        mouse_off(1);
        clearviewport();
        sprintf(buffer,"%d of %d",obj+1, index.currlen);
        outtextxy(0,0,buffer);
        mouse_on(1);
        /* back to normal viewport */
        setviewport(rectxul+1,rectyul+1,rectxlr-1,rectylr-1,1);
        while(!(k = mouse_trigger(0)));
        if (k == LEFT_MOUSE_REL) {
            command = mouse_on_entry(show_menu, num_show_menu_items);
            if (command != 4) {
                switch(command) {
                    case 0: /* first object */
                        obj = 0;
                        break;
                    case 1: /* last object */
                        obj = index.currlen-1;

```

```

break;
case 2: /* next object, wrap around to beginning */
    if (++obj == index.curren) obj = 0;
break;
case 3: /* previous object, wrap around to end */
    if (--obj < 0) obj = index.curren - 1;
break;
default: /* default to first object */
    obj = 0;
}
if (command >= 0 && command <= 3) {
    /* get object */
    getv1r(fh, &v, ((long *) (index.data))[obj]);
    /* clear window, and show object */
    mouse_off(1);
    clearviewport();
    fillpoly(v.curren, (int far *) (v.data));
    mouse_on(1);
}
}

} while (command != 4);
mouse_reset();
closegraph();
closefile(fh);
}

int mouse_on_entry(button *menu, int nitems)
/* Returns button number that mouse is currently on, or returns -1 */
{
    int i;
    for(i=0; i < nitems; i++) {
        if( mouse_in_box(1,
            menu[i].x, menu[i].y,
            menu[i].x + strlen(menu[i].label)*8 - 1, menu[i].y + 7) )
            return i;
    }
    return -1;
}

```


第十一章 Turbo C 图形函数的使用

本章的目的不是重写《Turbo C Additions & Enhancements Guide》的图形部分，而是指出使用图形例行程序过程中的技巧和陷阱。在简短的介绍之后，我们将转入使用鼠标器并随后给出一个样例图形弹出窗口软件包。这含有克服 Turbo C 图形的某些限制的方法。这将给我们一个机会以访问许多严格组织、面向应用的图形例行程序。虽然现在我们将很快看一下图形程序，但它假定你已熟悉这些图形例行程序了。

11.1 对 Turbo C 图形的快速浏览

Turbo C 软件包提供对许多不同图形适配器的支持并允许你写相当独于模式的程序。随着库文件 graphics.Lib，还有几个别的文件为图形软件包所用。这包括一组用于不同图形模式的驱动程序文件和一组笔划字体文件：

| File | Use |
|-------------|--|
| ATT.BGI | Driver file for AT&T Graphics |
| CGA.BGI | Driver file for CGA Graphics |
| EGAVGA.BGI | Driver file for EGA and VGA Graphics |
| HERC.BGI | Driver file for Hercules Graphics |
| IBM8514.BGI | Driver file for IBM 8514 Graphics |
| PC3270.BGI | Driver file for IBM 3270 Graphics |
| GOTH.CHR | File for gothic style stroked fonts |
| LITT.CHR | File for small stroked fonts |
| SANS.CHR | File for sansserif style stroked fonts |
| TRIP.CHR | File for triplex style stroked fonts |

注意：在运行本章的任何例子之前，一定要把所有必须的驱动程序和字体文件装入你的工作目录。

下面的程序显示了访问 Turbo C 图形例行程序所必需的程序。

```
/*
Quick tour of Turbo C graphics (grphex1.c)
Must link with: graphics.lib
*/
#include <graphics.h> /* must always include when using graphics */
#include <conio.h>
void main() {
    int gd = DETECT, gm = 0; /* set for auto-detect, highest mode */
    initgraph(&gd, &gm, ""); /* initialize the graphics system */
                                /* drivers should be in current dir */
    circle(100, 100, 50);      /* sample graphics routine */
    outtextxy(88, 100, "abc"); /* sample text output */
}
```

```

    getch();
    closegraph();          /* go back to text mode      */
}

```

程序 `initgraph()` 用于初始化图形系统。这是通过在运行时间动态装入适当的图形驱动程序来完成。驱动程序由参数 `gd` 决定。当设置为 `DETECT (0)` 时，它告诉 `initgraph()` 自动判别正在使用的硬件的类型，并装入适当的驱动程序。`gm` 参数告诉它用哪种图形模式。如果在自动判别驱动程序时，这个参数是 0，则它使用可能的最高分辨率。传递两个都设置为 0 的参数是初始化图形系统的最常见方法。

第三个参数确定找到图形驱动程序和字体表的目录。Turbo C 在运行时间查看该目录装入这些驱动程序。也有可能在连接时间用例行程序 `registerbgidriver()` 和 `registerbgifont()` 来装入它们。(请见 Turbo C 使用手册)。在本章的例子中，我们将假定正确的驱动程序和字体文件在当前目录中。

到 `circle()` 的调用显示了一个被使用的典型图形函数。圆的坐标也同任何图形物体的一样，总是相对于当前视见区指定的。视见区是屏幕上所有图形输出都画在其上的一个区域。它与基于文本的窗口类似，并且这里我们将交替使用这两个术语。系统设置的视见区是整个屏幕。

到 `Dut textxy()` 的调用显示了一种在图形模式下输出文本的方法。Turbo C 提供一种位映像的字体，也提供 4 种笔划字体。字体非常通用，因为你可以改变它们的形状、大小、颜色、并调整排字。

虽然提供了许多函数，但下面这些可能是最常用的

Basic Initialization and viewport functions.

| Function | Use |
|------------------------------|--|
| <code>initgraph()</code> | Initializes the graphics system |
| <code>closegraph()</code> | Closes the graphics system, goes back to text mode |
| <code>setviewport()</code> | Sets up a viewport |
| <code>clearviewport()</code> | Clears the current viewport |
| <code>cleardevice()</code> | Clears the whole screen |
| <code>setcolor()</code> | Sets the drawing color |
| <code>setbkcolor()</code> | Sets the background color |

Basic text output functions

| Function | Use |
|-----------------------------|--|
| <code>outtextxy()</code> | Outputs text at a certain pixel location |
| <code>outtext()</code> | Outputs text at the current pixel location |
| <code>settextstyle()</code> | Sets text font, size, and direction |

Basic drawing functions

| Function | Use |
|----------|-----|
|----------|-----|

| | |
|-----------------------------|-----------------------------------|
| <code>drawpixel()</code> | Draws a pixel |
| <code>circle()</code> | Draws a circle |
| <code>line()</code> | Draws a line |
| <code>rectangle()</code> | Draws a rectangle |
| <code>bar()</code> | Draws a bar for bar graphs |
| <code>bar3d()</code> | Draws a bar for 3D bar graphs |
| <code>setfillstyle()</code> | Sets the fill pattern and color |
| <code>setlinestyle()</code> | Sets the line thickness and style |

Animation Support Routines

| Function | Use |
|-------------------------|--|
| <code>getimage()</code> | Saves a bit mapped image into memory |
| <code>putimage()</code> | Puts a bit mapped memory image onto the screen |

你将有机会看到这些例行程序中的多数起作用，还有几个这里未列出的别的例行程序。所以卷起你的袖子开始工作吧。

11.2 使用鼠标器

正如你在文本状态下使用鼠标器一样，在图形状态下你也可以使用鼠标器。同鼠标器一起提供的驱动程序几乎为你办所有的事。只须保证在设置屏幕为图形方式之后初始化鼠标器，这样可使鼠标器正确建立。早先给出的鼠标器工具箱设计为既可在图形方式下工作也可在文本方式下工作。

在图形方式下使用鼠标器有两个问题：一是当使用一个 Hercules 图形卡时，另一个是在使用低分模式时。

Hercules 图形卡通过在文本状态下仿真 IBM 单色显示适配器来工作。另外，它有一种特殊的图形模式，不幸的是它不被 BIOS 认可，则为它不是 IBM 标准的，因为这就无法使鼠标器驱动程序判别何时正在使用 Hercules 图形方式。纠正办法是使 BIOS 误认为你在图形状态下。

BIOS 使用位于 `0x0040:0x0049` 的一块特殊内存区域为它的显示例行程序存储各种各样的参数。该区域 (`0x0040:0x0049`) 开始处的那一个用于存储当前显示状态。当在 Hercules 文本状态时，这里存的是 7。不幸的是，如果你转换到 Hercules 图形方式，它仍是 7，因为 BIOS 不知道 Hercules 图形。存储一个 6 在那儿，鼠标器驱动程序将知道一个图形方式正在使用中（具体地讲，是 Hercules 图形）。这个值假定你正在使用 Hercules 图形存储区的第 0 页。如果你在使用图形存储区的第一页则应在 `0x0040:0x0049` 处存一个 5。

另一问题与恢复鼠标器坐标有关。鼠标器驱动程序假定除 Hercules 外的所有状态下实际屏幕宽是 640 个像素，Hercules 使用 720 个像素。但有几个低分状态仅有 320 个像素宽。这些情况下，鼠标器驱动程序返回的 X 坐标必须折半才对。所用的值是 Turbo C 函数 `initgraph()` 返回的值。函数 `initmouse()` 取这些值并判定是使用的 Hercules 图形还是低分图形。对于 Hercules 图形，它设置内存位置 `0x0040:0x0049` 为值 6，(0 页图

形)。对低分图形，它设置一个内部标志以便在请求鼠标图形坐标时，它们能被正确计算。下面的程序显示了如何正确建立鼠标器：

```
/*
    Using the mouse in graphics mode (grphex2.c)
    Must link with: mouse.obj, graphics.lib
*/
#include <graphics.h>
#include "mouse.h"
main() {
    int gd = DETECT, gm = 0; /* default modes */
    initgraph(&gd, &gm, "");
    init_mouse(MOUSE_NEEDED, gd, gm);
    while (!mouse_trigger(1));
    mouse_reset();
    closegraph();
}
```

11.3 由鼠标器驱动程序改变鼠标器光标

与文本状态下使用的长方形鼠标器光标不同，图形状态下的系统设置的形状是一个指向左上的箭头。别担心，有办法产生你自己的光标。基本上有两种方法：通过调用鼠标器驱动程序的一个特殊函数或通过涂改产生自己的光标。我们将教你两种方法。

在大多数图形状态下鼠标器光标都被定义为 16×16 象素的块。虽然你可以通过正确设置其形状而使之变小一些，但光标的大小不能超过它。为改变鼠标器光标，你可以以功能号 9 调用鼠标器驱动程序，并传三个参数到它。

前两个参数定义光标的“热点”坐标。热点确定光标的原点；即从该点确定鼠标器坐标。例如，在系统设置的光标中，热点是箭头的尖。位置是相对于 16×16 块的左上角给出的，且两个参数都必须在 -16 到 16 的范围内。使用负坐标，你可以把热点定义在实际光标的左上方。

最后一个参数指定两个 16×16 位的用于创建光标形状的屏蔽的地址：屏幕屏蔽和光标屏蔽。当鼠标器光标画好后，屏幕屏蔽象素与屏幕上的象素进行 AND 运算。结果再与光标屏蔽进行 XOR 运算。这种有趣的组合可使你把光标制成背景色白色、透明的、或是背景色的相反色。适当设置屏蔽，你可以在同一光标内得到这些颜色的任一组合。图 11.1 表明了该 AND 和 XOR 的过程。表 11.1 向你表明位组合如何起作用。

Table 11.1 Mouse cursor bit combinations

| Screen Mask Bit | Cursor Mask Bit | Resulting Color |
|-----------------|-----------------|---------------------------|
| 0 | 0 | Background color |
| 0 | 1 | White |
| 1 | 0 | Transparent |
| 1 | 1 | Foreground color inverted |
| | | Background becomes white |

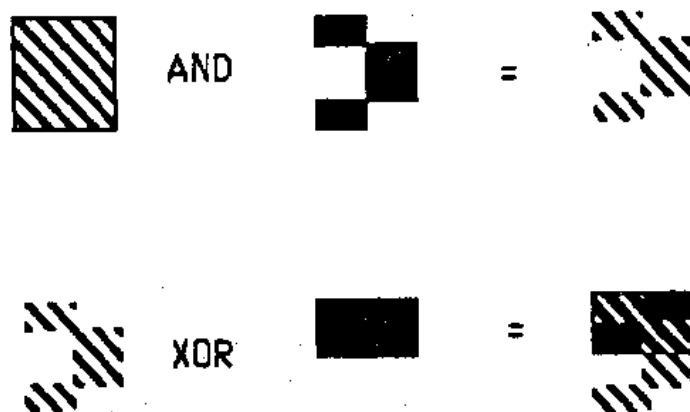


Figure 11.1 The two step cursor masking process (colors inverted)

一个改变鼠标器光标的例子在下面的程序中给出:

```

/*****
 * Example of changing the mouse cursor (grphex3.c)
 * Must link with: mouse.obj, graphics.lib
 *****/
#include <dos.h>
#include <conio.h>
#include <graphics.h>
#include <alloc.h>
#include "mouse.h"
unsigned int test_pattern[] = {
    /* Screen Mask */
    0xff00, /* 1111 1111 0000 0000 */
    0xff00, /* 1111 1111 0000 0000 */
    0xff00, /* 1111 1111 0000 0000 */
    0xff00, /* 1111 1111 0000 0000 */
    0x00ff, /* 0000 0000 1111 1111 */
    0x00ff, /* 0000 0000 1111 1111 */
    0x00ff, /* 0000 0000 1111 1111 */
    0x00ff, /* 0000 0000 1111 1111 */
    0x00ff, /* 0000 0000 1111 1111 */
    0x00ff, /* 0000 0000 1111 1111 */
    0x00ff, /* 0000 0000 1111 1111 */
    0x00ff, /* 0000 0000 1111 1111 */
    0x00ff, /* 0000 0000 1111 1111 */
    0x00ff, /* 0000 0000 1111 1111 */
    0x00ff, /* 0000 0000 1111 1111 */
    0xff00, /* 1111 1111 0000 0000 */
    0xff00, /* 1111 1111 0000 0000 */
    0xff00, /* 1111 1111 0000 0000 */
    0xff00, /* 1111 1111 0000 0000 */
    /* Cursor mask */
    0xffff, /* 1111 1111 1111 1111 */
    0xffff, /* 1111 1111 1111 1111 */
    0xffff, /* 1111 1111 1111 1111 */
}

```

```

0xffff,      /* 1111 1111 1111 1111 */
0xffff,      /* 1111 1111 1111 1111 */
0xffff,      /* 1111 1111 1111 1111 */
0xffff,      /* 1111 1111 1111 1111 */
0xffff,      /* 1111 1111 1111 1111 */
0x0000,      /* 0000 0000 0000 0000 */
0x0000,      /* 0000 0000 0000 0000 */
0x0000,      /* 0000 0000 0000 0000 */
0x0000,      /* 0000 0000 0000 0000 */
0x0000,      /* 0000 0000 0000 0000 */
0x0000,      /* 0000 0000 0000 0000 */
0x0000,      /* 0000 0000 0000 0000 */
0x0000,      /* 0000 0000 0000 0000 */
0x0000,      /* 0000 0000 0000 0000 */
};

void setcur(int hotspotx, int hotspoty, void *shape);
void main() {
    int gd = DETECT, gm = 0;
    initgraph(&gd, &gm, "");
    init_mouse(MOUSE_NEEDED, gd, gm);
    /*
       Set background to red, draw a yellow fill patterned box
       to see effects of cursor masks
    */
    mouse_off(1);
    setbkcolor(RED);
    setfillstyle(BKSLASH_FILL, YELLOW);
    bar(100, 50, 200, 75);
    mouse_on(1);
    while(!mouse_trigger(1));
    /* Change to test pattern cursor */
    setcur(0, 0, test_pattern);
    while(!mouse_trigger(1));
    mouse_reset();
    closegraph();
}

void setcur(int hotspotx, int hotspoty, void *shape)
/*
    Routine to set the graphics mouse cursor to
    specified shape with specified hot spot coordinates.
*/
{
    union REGS inregs, outregs;

```

这个程序先画一个黄色斜纹填充图形方框和一个系统设置的形状的光标。按下任一鼠标器按钮则光标就改变到一个测试图形形状，它里边有所有的可能的位组合。把光标移到填充图形之上可以看到屏蔽的效果。按另一鼠标器按钮则退出该程序。

函数 `setcur()` 通过 `int86x()` 调用鼠标器驱动程序来改变光标形状。你也许会想把该例行程序加到你的鼠标器库中。注意屏蔽地址是如何传递的。屏蔽被安排在一个单整数数组中，屏幕屏蔽后随光标屏蔽。该数组的地址是通过寄存器 `DX` 和 `ES` 传递的。

这个程序可用于任一图形状态，但在 `EGA` 和 `VGA` 彩色状态下最引人注目。在 `CGA` 高分状态下屏蔽效果也相当好。

11.4 建自己的光标

用 Turbo C 例行程序 `getimage()` 和 `putimage()`，你可以创建你自己的图形光标。这有几个优势。你对光标的颜色有更多的控制，并且你可以使用任意希望的大小。可能主要的优势还是即使你没有鼠标器也能有光标。这个光标可由光标键移动。这着实意味着你必须明确地写一个驱动程序来移动此光标，而对鼠标器光标来说这却是由鼠标器驱动程序为你做了。

虽然这不是最简单的办法，但下面的程序表明了如何用鼠标器驱动程序光标使用的同样两个屏蔽来创建一个光标，主要是为了表明鼠标驱动程序为其光标所进行的操作。

```

    struct SREGS sereg;
    inregs.x.ax = 9; /* Cursor shape function code */
    inregs.x.bx = hotspotx;
    inregs.x.cx = hotspoty;
    inregs.x.dx = FP_OFF(shape);
    sereg.es = FP_SEG(shape);
    int86x(0x33, &inregs, &outregs, &sereg);
/*****
 * Rolling your own mouse cursor (grphex4.c)
 * Must link with: graphics.lib
 *****/
#include <dos.h>
#include <bios.h>
#include <conio.h>
#include <graphics.h>
#include <alloc.h>
#define UPKEY      0x4800
#define DOWNKEY    0x5000
#define LEFTKEY    0x4b00
#define RIGHTKEY   0x4d00
#define ESCKEY     0x011b
unsigned int test_pattern[] = {
    /* Screen Mask */
    0xff00, /* 1111 1111 0000 0000 */
    0xff00, /* 1111 1111 0000 0000 */
    0xff00, /* 1111 1111 0000 0000 */
    0xff00, /* 1111 1111 0000 0000 */
    0x00ff, /* 0000 0000 1111 1111 */

```

```

0x00ff,      /* 0000 0000 1111 1111 */
0x00ff,      /* 0000 0000 1111 1111 */
0x00ff,      /* 0000 0000 1111 1111 */
0x00ff,      /* 0000 0000 1111 1111 */
0x00ff,      /* 0000 0000 1111 1111 */
0x00ff,      /* 0000 0000 1111 1111 */
0x00ff,      /* 0000 0000 1111 1111 */
0x00ff,      /* 0000 0000 1111 1111 */
0xff00,      /* 1111 1111 0000 0000 */
0xff00,      /* 1111 1111 0000 0000 */
0xff00,      /* 1111 1111 0000 0000 */
0xff00,      /* 1111 1111 0000 0000 */
/* Cursor mask */
0xffff,      /* 1111 1111 1111 1111 */
0xffff,      /* 1111 1111 1111 1111 */
0xffff,      /* 1111 1111 1111 1111 */
0xffff,      /* 1111 1111 1111 1111 */
0xffff,      /* 1111 1111 1111 1111 */
0xffff,      /* 1111 1111 1111 1111 */
0xffff,      /* 1111 1111 1111 1111 */
0xffff,      /* 1111 1111 1111 1111 */
0xffff,      /* 1111 1111 1111 1111 */
0x0000,      /* 0000 0000 0000 0000 */
0x0000,      /* 0000 0000 0000 0000 */
0x0000,      /* 0000 0000 0000 0000 */
0x0000,      /* 0000 0000 0000 0000 */
0x0000,      /* 0000 0000 0000 0000 */
0x0000,      /* 0000 0000 0000 0000 */
0x0000,      /* 0000 0000 0000 0000 */
0x0000,      /* 0000 0000 0000 0000 */
0x0000,      /* 0000 0000 0000 0000 */
};

void make_cursor(unsigned int *curs, void *scrnmsk, void *cursmsk);
void draw_cursor(int x, int y);
void erase_cursor(int oldx, int oldy);
void *scrnmsk, *cursmsk, *save_image;
void main() {
    int gd = DETECT, gm = 0, k, x, y, mx, my;
    initgraph(&gd, &gm, "");
    save_image = malloc(imagesize(0,0,15,15));
    scrnmsk = malloc(imagesize(0,0,15,15));
    cursmsk = malloc(imagesize(0,0,15,15));
    make_cursor(test_pattern, scrnmsk, cursmsk);
    outtextxy(0,40,"Bit masks made. Press a key to continue ...");
    getch();
    cleardevice(); /* erase cursor construction */
    /*
       Set background to red, draw a yellow fill patterned box
       to see effects of cursor masks
    */
    setbkcolor(RED);
    setfillstyle(BKSLASH_FILL, YELLOW);
    bar(100, 50, 200, 75);

```



```

x = 100; y = 50;
mx = getmaxx()-15; my = getmaxy()-15;
draw_cursor(x,y);
do {
    if (bioskey(1)) {
        k = bioskey(0);
        erase_cursor(x,y);
        switch(k) {
            case UPKEY:
                if (-y < 0) y = 0;
                break;
            case DOWNKEY:
                if (++y > my) y = my;
                break;
            case LEFTKEY:
                if (-x < 0) x = 0;
                break;
            case RIGHTKEY:
                if (++x > mx) x = mx;
                break;
            default: ;
        }
        draw_cursor(x,y);
    }
} while(k != ESCKEY);
closegraph();

void make_cursor(unsigned int *curs, void *scrnmsk, void *cursmsk)
/*
    Makes the cursor masks by forming bit images right on the screen.
    The masks should be allocated before coming in.
*/
{
    int i, j;
    setbkcolor(0); /* should be black for this to work properly */
    /* create the screen mask right on the screen */
    for (i = 0; i<16; i++) {
        for (j=0; j<16; j++) {
            if ((curs[i] >> (15-j)) & 0x0001)
                putpixel(j,i,WHITE); /* important to use white !! */
        }
    }
    /* now store bit image of it in a buffer */
    getimage(0,0,15,15,scrnmsk);
    /* same for cursor mask */
    for (i = 0; i<16; i++) {
        for (j=0; j<16; j++) {
            if ((curs[i+16] >> (15-j)) & 0x0001)
                putpixel(j,i+20,WHITE); /* important to use white !! */
        }
    }
}

```

```

    }
    getimage(0,20,15,35,cursmsk);
}
void erase_cursor(int oldx, int oldy)
/* Erase cursor by putting back old image */
{
    putimage(oldx,oldy,save_image,COPY_PUT);
}
void draw_cursor(int x, int y)
/* Draw cursor at new position, but save image underneath first */
{
    getimage(x,y,x+15,y+15,save_image); /* save image */
    putimage(x,y,scrnmsk,AND_PUT);      /* draw the cursor */
    putimage(x,y,cursmsk,XOR_PUT);
}

```

除了你必须明确地创建、移动和擦除自己的光标外，这个程序与在鼠标器例子中所用的在本质上是一样。键盘上的箭头用手四处移动光标，ESC 退出程序。

函数 `make_cursor()` 负责取屏幕、光标屏蔽和创建光标图形。它使用屏蔽中的位模式在黑色背景上画白色像素以完成这一功能。一旦图形形成，就调用 `getimage()` 来存储光标的实际内存图形。然后可调用 `putimage()` 在任何位置画此光标。

认识到有位屏蔽和有实际图形本身之间的差别是很重要的。在 EGA 和 VGA 中，每个像素是由 * 安排在四个面上的四位形成的。这四个面确定像素的颜色。对每一像素都须设置全部四位，以便 AND 和 XOR 操作能正确执行。这可用黑上加白的颜色方案来完成。

为以后的使用而先在屏幕上画出图形的这种创建图形的技术是一种典型的创建自己的光标、图像等的方法。除了分辨率和颜色上的差异外，使用这种方法可使你的图形适用于任一图形状态。如果先看到它们画出来使你感到麻烦，则如果你的图形适配器支持的话，你可先在一个不同的隐含的图形页上画它们。Turbo C 函数 `setactivepage()` 和 `setvisualpage()` 可用于此目的。

函数 `draw_cursor()` 和 `erase_cursor()` 用于画和擦除光标。为允许移动光标，下面的图形在画光标之前就被存储起来，而在擦除光标后又恢复之。为移动光标用了四个内存变换：先恢复旧位置图形，再存储新位置图形，与屏幕屏蔽做 AND 运算，再与光标屏蔽做 XOR 运算。这不是移动图形的最佳方法。

最有效的方法是把图形与下面的屏蔽进行 XOR 运算。两个连续的 XOR 运算可导致图形被画出，而后又被擦除。这样只用 `putimage()` 和一个 XOR 运算即可移动图形。不必要存储下面的图形。用它情形将怎样呢？由于 XOR 起作用的方式，屏幕的状态是通过改变像素颜色存储的。例如，如果一个屏幕像素是黄色，又与一个蓝色像素做 XOR 运算，则产生一个黄色 * 像素。另一个 XOR 把像素恢复为黄色。这意味着光标的颜色与其下面的颜色相互影响，这对某些应用可能不太理想。

鼠标驱动程序使用的多个存储、AND 和 XOR 操作允许你有创建光标外观的更大的灵活性。你可使之部分为白色、翻转的、透明的、或为背景色。只有对 XOR 运算，你才完全受它的控制。但是，XOR 技术有效得多，而且与往常一样，有得有失。

11.5 一个样例图形弹出窗口软件包

本节我们将给出在图形状态下做弹出窗口的程序。程序在清单 11.1 和 11.2 中给出。它是借用文本状态下的弹出窗口例行程序并稍加修改以适宜在图形状态下工作而写成的。虽然小心地采取了步骤使这些调用尽量与文本状态例行程序接近，但不幸的是这些想法并不总是能够被满足。虽然这里我们不做，但写一个软件包而能支持文本和图形两种状态将是一个有趣的练习。

该例行程序中有许多都与文本状态软件包使用的相同或非常相似。所缺的只是支持写文本到窗口的那些程序。你得自己提供这些程序。这里的意图不是给你一个完整的软件包，而是一个介绍许多 Turbo C 图形功能特性的程序。

由于这个软件包使用了与文本状态形式下建立的相同的概念，所以我们将主要集中于这些程序的差别上。关于文本状态弹出窗口的详细情况见第四章。

11.6 窗口状态

Turbo C 为图形状态提供比文本状态更多的设置。文本弹出窗口软件包中定义的 `wincolors` 结构被修改以处理这些附加的设置，该结构给出如下：

```
typedef struct wincolors_struct {
    struct linesettingstype lineinfo; /* line style */
    struct fillsettingstype fillinfo; /* fill style */
    struct textsettingstype textinfo; /* font style */
    struct linesettingstype brdinfo; /* border line style */
    struct viewporttype viewinfo; /* window coordinates */
    char border_type;
    unsigned char border_color, text_color, hilite_color, back_color;
} wincolors;
```

附加的结构存储当前颜色，视见区坐标，和行及填充格式的状态。用于获取这些设置和它们的相关结构的例行程序是：

| Routine | Associated Structure |
|--------------------------------|--|
| <code>getfillsettings()</code> | <pre>struct fillsettingstype { int pattern; int color; }</pre> |
| <code>getlinesettings()</code> | <pre>struct linesettingstype { int linestyle; unsigned upattern; int thickness; }</pre> |
| <code>gettextsettings()</code> | <pre>struct textsettingstype { int font; int direction; int charsize; int horiz; int vert; }</pre> |

```

getviewsettings()    struct viewporttype {
                      int left, top, right, bottom;
                      int clip;
                      }

```

Turbo C 有相应的程序改变这些设置:

| Function | Use |
|------------------|---|
| setfillpattern() | Selects a user-defined fill pattern |
| setfillstyle() | Selects a fill pattern and color |
| setlinestyle() | Changes the line width and style |
| settextjustify() | Sets the text justification |
| settextstyle() | Sets the text font, direction, and size |
| setviewport() | Sets the viewport coordinates |

对 wincolors 结构所做的另一改动是没有标题颜色 (对图形窗口不画标题), 但加了一个背景颜色。这里提到的背景颜色不是屏幕的总的背景色, (它由 setbkcolor() 设置), 而是清除该窗口时所用的颜色。

border-type 域基本上与其原来的作用一样。把它设置为 0 将不画边界。把它设置为 1 将导致 NORM-WIDTH 行 (1 个像素宽) 用做边界。把它设置为 2 将导致使用 THICK-WIDTH 行 (3 像素宽)。

有了这些对窗口颜色软件包的改动, 窗口结构本身将是这样的:

```

typedef struct winstruct {
    char *name;                /* window title */
    void *image;               /* ptr to image save area */
    struct winstruct *under,*over; /* ptrs to window below and
                                   above on pop-up stack */
    wincolors wc;              /* current drawing settings */
    int wd, ht;                /* computed width & height */
    int xsave,ysave;           /* saved cursor posn */
    enum windowtype wtype;     /* window type */
} windesc;

```

这与文本状态下的结构几乎一样。一个变化是虽然存储了一个标题, 但它永远也不被画出来。它在这里主要是为了兼容性。另一个变化是坐标 Xul、Yul、Xlr 和 ylr 不再出现了。这些坐标被定义在存储于 wincolor 结构中的 viewport (视见区) 结构中。为使把文本状态程序容易地移植到图形状态, 在头文件中定义了下面的宏:

```

#define xul wc.viewinfo.left
#define yul wc.viewinfo.top
#define xlr wc.viewinfo.right
#define ylr wc.viewinfo.bottom

```

11.7 初始化窗口软件包

函数 init-win() 用于初始化窗口软件包。与它的文本状态对等部分相似, 它取当前屏幕设置并用基本窗口存储它们。

与文本状态的形式不同, 图形 init-win() 使用三个附加的全程变量。它们被定义

为:

```
int graphdriver = 0;
int graphmode   = 0;
char *pathdriver = "";
```

前两个全局变量是在初始化图形系统时所用的图形驱动程序和图形模式。如果给定系统设置值则会使图形系统被初始化为可能的最高状态（即发生自动判别）。如果你愿意，你可在调用 `init-win()` 之前把它们设置为其他的值。

最后一个参数是到存储图形驱动程序和字体目录的路径。它的系统设置值是“”，这意味着是当前目录。正如所写的那样，图形弹出窗口软件均在运行时间装入图形驱动程序和字体。如果你想在连接时间装入它们，则你需相应改变 `init-win()` 函数。

11.8 画窗口

例行程序 `draw-win()` 用于弹出一个图形窗口，它与文本状态形式中使用的那个非常相似。

注意图形存储缓冲区是怎样被初始化的。由于写 `draw-win()` 的方式，到 `putimage()` 的调用最终是在图形被存储之前做的（通过 `swap-image()`）。与 `gettext()` 和 `puttext()` 使用的缓冲区不同，图形缓冲区不仅存储图形本身，还存储其大小。如果 `putimage()` 是用一个未初始化的缓冲区调用的，则不仅会出现无用图形，而且可能超出原定区域成为任意大小。为了防止它，图形缓冲区用 `calloc()` 分配，它把整个缓冲区初始化为 0，也有效地把图形大小设置为 0。这个图形大小在 `getimage()` 一被调用就被更新。

11.9 交换图形

在函数 `swap-image()` 中，函数 `getimage()` 和 `putimage()` 用于代替它们的文本状态对等部分 `gettext()` 和 `puttext()`。一个细微的差别是 `getimage()` 和 `putimage()` 使用视觉区相对坐标，而文本状态则使用绝对坐标。为存取窗口的边界（它在视觉区以外），你必须在调用 `getimage()` 和 `putimage()` 之前转换到绝对坐标。`chgviewport(base-win)` 调用就执行这一功能。

11.10 清除窗口

例行程序 `clr-win()` 不仅清除当前窗口，还设置背景颜色并重画边界。它调用函数 `fill-win()` 做所有的工作。

注意，你不能用 `setbkcolor()` 设置窗口的背景颜色，因为这个程序为整个屏幕设置颜色。代替之的是用 `bar3d()`。使用这个程序是因为它一次为你做两件事。它不仅能画一个长方形边界，而且还能用指定的颜色填充该长方形。在调用 `bar3d()` 之前填充格式被设置为 `SOLID-FILL`，而颜色被设置为窗口的背景色。由于在 `swap-image()` 中，你必须使用绝对坐标。

一个细微的问题是，如果用的是 `THICK-WIDTH` 行，因为它们宽于 1 个象素，你必须使用比 `NORM-WIDTH` 行的更小的横坐标，以便边界保持在为窗口保留的屏幕区域内。

11.11 改变窗口

例程序 `view-win()`, `chg-win()`, 和 `chgviewport()` 都与选择新窗口有关。`view-win()` 函数与用于文本状态下的情形一样。`chg-win()` 函数被修改为存储旧颜色、行和填充格式, 并转换到新的去。函数 `chgviewport()` 是增加的, 并且在计算新窗口坐标时处理不同边界行宽。

11.12 移动窗口例子

为说明图形弹出窗口软件包的作用, 下面是用于基于文本的弹出窗口的移动窗口例子(见第四章), 被修改为用于图形状态了。

```
/* ****  
 * Moving Windows Example in Graphics Mode (grphex5.c)  
 * Must link with: gpopup.obj, mouse.obj, graphics.lib  
 **** */  
#include <stdlib.h>  
#include <conio.h>  
#include <bios.h>  
#include <graphics.h>  
#include "gpopup.h"  
#include "mouse.h"  
#define MAX(a,b) ((a) > (b) ? (a) : (b))  
#define MIN(a,b) ((a) < (b) ? (a) : (b))  
wincolors acolors, bcolors, ccolors;  
int mouse_on_border(windesc **w, int *i, int *xofs, int *yofs);  
void move_curr_win(int x, int y);  
void main() {  
    windesc *w[3];  
    int x, y, xofs, yofs, i;  
    unsigned int k;  
    init_win();  
    init_mouse(MOUSE_OPTIONAL, graphdriver, graphmode);  
    acolors = defcolors; acolors.back_color = BLACK;  
    bcolors = defcolors; bcolors.back_color = BLACK;  
    ccolors = defcolors; ccolors.back_color = BLACK;  
    w[0] = draw_win(50, 60, 200, 50, "", popup, &acolors);  
    rectangle(50, 15, 150, 35);  
    outtextxy(100, 25, "1");  
    w[1] = draw_win(70, 90, 200, 60, "", popup, &bcolors);  
    circle(100, 25, 20);  
    outtextxy(100, 25, "2");  
    w[2] = draw_win(90, 110, 200, 50, "", popup, &ccolors);  
    ellipse(100, 25, 0, 360, 40, 15);  
    outtextxy(100, 25, "3");  
    do {  
        while (!(k = mouse_trigger(1))); /* get event trigger */  
        if (k == LEFT_MOUSE_PRESS) {
```

```

    if (mouse_on_border(w, &i, &xofs, &yofs)) {
        slct_win(w[i]); /* mouse on border of window, so select it */
        while(button_state()) { /* move till button release */
            mouse_grph_posn(&x, &y);
            move_curr_win(x-xofs, y-yofs);
        }
    }
}

else { /* possible keyboard press */
    xofs = 0; yofs = 0; /* reset mouse/window corner offsets */
    x = curr_win->xul; /* default to current position */
    y = curr_win->yul;
    switch(k) {
        case UPKEY:
            y -= 5;
            move_curr_win(x, y); /* move_curr_win will do bounds checking */
            break;
        case DOWNKEY:
            y += 5;
            move_curr_win(x, y);
            break;
        case LEFTKEY:
            x -= 5;
            move_curr_win(x, y);
            break;
        case RIGHTKEY:
            x += 5;
            move_curr_win(x, y);
            break;
        case 0x0231: /* the "1" key */
            slct_win(w[0]);
            break;
        case 0x0332: /* the "2" key */
            slct_win(w[1]);
            break;
        case 0x0433: /* the "3" key */
            slct_win(w[2]);
            break;
        default: /* send character to current window */
            if (k != RIGHT_MOUSE_PRESS) /*mprintf("%c", lo(k))*/;
    }
}

} while (k != ESCKEY && k != RIGHT_MOUSE_PRESS);
rmv_win(w[2]);
rmv_win(w[1]);
rmv_win(w[0]);
mouse_reset(0);
closegraph();

```

```

int mouse_on_border(windesc **w, int *i, int *xofs, int *yofs)
/* Given an array of windows, determines which border the mouse
   is currently on, and returns the window number and the offset
   of the mouse from the top left hand corner
*/
(
    int x, y, j;
    mouse_grph_posn(&x, &y);
    for (j = 0; j < 3; j++) {
        if (x >= w[j]->xul && x <= w[j]->xlr &&
            y >= w[j]->yul && y <= w[j]->yul) { /* in window region */
            if (abs(x - w[j]->xul) < 4 ||
                abs(x - w[j]->xlr) < 4 ||
                abs(y - w[j]->yul) < 4 ||
                abs(y - w[j]->yul) < 4) { /* within range of border */
                *i = j;
                *xofs = x - w[j]->xul;
                *yofs = y - w[j]->yul;
                return 1;
            }
        }
    }
    return 0;
}

void move_curr_win(int x, int y)
/*
   Moves the current window. If coordinates haven't changed,
   then nothing happens. Does bounds checking on the new position.
*/
(
    int xsave, ysave;
    if (x != curr_win->xul || y != curr_win->yul) {
        xsave = getx();
        ysave = gety();
        swap_image(curr_win); /* hide window */
        curr_win->xul = x;
        curr_win->yul = y;
        curr_win->xul = MAX(curr_win->xul, 0);
        curr_win->xul = MIN(curr_win->xul, getmaxx() - curr_win->wd);
        curr_win->yul = MAX(curr_win->yul, 0);
        curr_win->yul = MIN(curr_win->yul, getmaxy() - curr_win->ht);
        curr_win->xlr = curr_win->xul + curr_win->wd - 1;
        curr_win->yul = curr_win->yul + curr_win->ht - 1;
        swap_image(curr_win); /* show window */
        chgviewport(curr_win); /* change window coordinates */
        moveto(xsave, ysave);
    }
}

```


该程序如下工作：如果你按数 1, 2 和 3, 则相应的窗口就被选中。如果你使用箭头键, 则你就能移动所选的窗口了；或者如果你愿意, 当它在窗口的边界上时按左鼠标器按钮。窗口被选中了, 然后你按着左鼠标器按钮拖动鼠标器就可移动窗口了。按 ESC 或右鼠标器按钮则退出, 图 11.2 显示了一个用此程序画的样例屏幕。

如果你在使用 EGA 或 VGA 高分彩色状态, 则你将注意到甚至在快速的 286 为基础的 AT 机上, 交换和移动窗口也是个相当慢的过程。这向你表明了除非更快的图形卡广泛地应用于 PC 机上基于文本的窗口可能是唯一可行的方法的原因。另外, 弹出图形窗口比它们的基于文本的对等部分占有多得多的内存。例如, 存储 EGA640×350 的 16 色状态下的整个屏幕大约要用 112K, 而文本方式只需 4K。

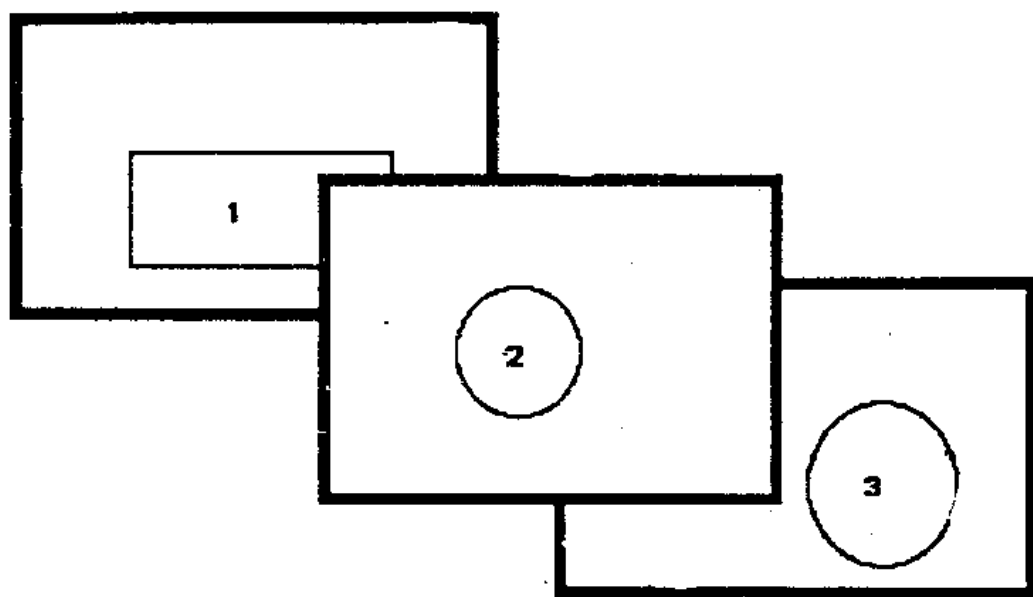


Figure 11.2 Moving windows in graphic mode

11.13 图形状态下的文本

本节我们将看一下写基于图形的文本在图形状态下输出文本的最简单的方法, 是调用 `putchar()`、`puts()` 或 `printf()`。这些标准 I/O 例行程序最终要调用 BIOS, 它支持在所有的图形状态 (除 Hercules) 下打印出文本。但是, 这个支持是很小的。一是它输出文本时用电传打字卷动状态, 并且它不支持彩色。二是不同字体和大小的使用受到严重限制。因此, Turbo C 提供了一套函数用于做更复杂的文本输出。

应该注意的是, 使用这些标准 I/O 例行程序并非全都不好。使用它们, 你可以不改动任何东西而使你的程序既能在文本状态又能在图形状态下工作。一个好的例子是 DOS 的 DIR 命令, 无论你在哪个状态下它都正常工作。你也保留做 I/O 重定向的能力, 而使用 `printf()` 给你以内部格式化的能力。(不久你将看到一种用 Turbo C 文本例行程序做某些类型的格式化的方法)。

Turbo C 为在图形状态下输出文本提供函数 `outtext()` 和 `outtextxy()`。还有用于改变颜色、字体、字体大小和调整文本的函数。这些函数是：

| Function | Use |
|--------------------------------|---|
| <code>setcolor()</code> | Sets the drawing color (which is used for text, among other things) |
| <code>gettextsettings()</code> | Get current font, size, direction, etc. settings |
| <code>settextjustify()</code> | Set text justification parameters |
| <code>settextstyle()</code> | Set font style, direction, and size |
| <code>setusercharsize()</code> | Set user-defined character sizes |
| <code>outtext()</code> | Output text at current position |
| <code>outtextxy()</code> | Output text at pixel position (x,y) |
| <code>textheight()</code> | Get height of text string using current font size |
| <code>cextwidth()</code> | Get width of text string using current font size |

同一种 8×8 位映像的字体一道，Turbo C 提供了四种笔划字体。这些字体不是被写作位映像的图形，而是由一组线段构成。当字体较大时它们比位映像字体好看得多，而这就是它们的主要优势。

虽然上述例行程序相当有用，但它们仍有几个不足：

- 未提供格式输出函数。
- 不支持卷动。
- 覆盖已在屏幕上的文本时不擦除旧文本。
- 不支持突出显示文本

现在我们将集中于解决这些不足的办法。

11.14 格式化文本

用 `vsprintf()` 把文本格式化到缓冲区中而后调用 `outtext()` 和 `outtextxy()`。把该缓冲区写到屏幕，你能得到有限的文本格式化形式。请看下面的程序样例。

```
/*
   Formatting text in graphics mode (grphex6.c)
   Must link with: graphics.lib
*/
#include <stdio.h>
#include <conio.h>
#include <stdarg.h>
#include <graphics.h>
void outfmttext(int x, int y, char *fmt, ...);
void main() {
    int gd = DETECT, gm = 0, age = 30;
    initgraph(&gd, &gm, "");
    outfmttext(50, 50, "Sally is %d years old", age);
    getch();
    closegraph();
}
void outfmttext(int x, int y, char *fmt, ...)
/*
```

Outputs formatted text in graphics mode.
 NOTE: (1) newlines and tabs are not supported.
 (2) Formatting must not exceed 255 characters!.

```
*/
{
    va_list arg_ptr;
    char t[255];
    va_start(arg_ptr, fmt);    /* point to optional arguments */
    vsprintf(t, fmt, arg_ptr); /* format the string */
    va_end(arg_ptr);
    outtextxy(x, y, t);
}
```

注意：这种方法不支持新行或制表字符，因为 outtextxy() 把它们当作图形字符对待。如果你想支持这些控制字符，则你必须写一个每次输出一个字符，并对你看到的任何一个控制字符做检查，并采取适当措施。

一个较困难的问题是支持窗口中的滚动。这可通过由 getimage() 和 putimage() 在图形上移一行来完成。移动的图形的大小可通过知道窗口大小及使用 textheight() 求得一行文本的高度来计算。你可把这样的例行程序当做一个练习来尝试。

11.15 覆盖文本

另一个问题是试图覆盖已在屏幕上的文本。试运行下面的程序即可看到这个问题：

```
/*
   Graphics text overwriting example #1 (grphex7.c)
   Must link with: graphics.lib
*/
#include <conio.h>
#include <graphics.h>
char msg1[] = "Baby, come here";
char msg2[] = "I am the ape man";
void main() {
    int gd = DETECT, gm = 0;
    initgraph(&gd, &gm, "");
    settextstyle(DEFAULT_FONT, HORIZ_DIR, 1); /* use bit mapped font */
    outtextxy(0, 0, msg1);
    getch();
    outtextxy(0, 0, "I am the ape man");
    getch();
    closegraph();
}
```

你将注意到第二条信息中的文本只是与第一条信息做了 OR 运算。第一条信息未被清除。

有几种办法解决此问题。一种是定义一个恰为第一信息大小的视见区而后在写第二信息前调用 clearviewport()。另一办法是画适当大小的实心线条清除第一信息，并把颜色设置为背景色。还有另一办法是用 getimage() 取得原文本的位图，而后用一个异或

putimage() 有效地擦除原文本。

这些方法中，画实线条可能是最好的方法。因为你不会影响当前视见区坐标。还有，你不必为位图图形分配内存，而且使文本具有不同颜色背景也易于处理。所有这些方法都要求计算原文本的大小。你可这样做：事先知道其大小，或用 textheight() 和 textwidth() 函数，或干脆清除全屏幕或它的一个已知区域。

让我们修改一下我们的程序以使之能在写第二个信息之前在第一信息之上画实心背景彩色线：

```
/*
    Graphics text overwriting example #2  (grphex8.c)
    Must link with: graphics.lib
*/
#include <conio.h>
#include <graphics.h>
char msg1[] = "Baby, come here";
char msg2[] = "I am the ape man";
void main() {
    int gd = DETECT, gm = 0, th, tw;
    initgraph(&gd, &gm, "");
    settextstyle(DEFAULT_FONT, HORIZ_DIR, 1);
    outtextxy(0, 0, msg1);
    getch();
    th = textheight(msg1); tw = textwidth(msg1);
    setfillstyle(SOLID_FILL, 0); /* clear first message */
    bar(0, th/4, tw, th+th/4); /* with the background color */
    outtextxy(0, 0, "I am the ape man"); /* new message */
    getch();
    closegraph();
}
```

如果你使用笔划字体则有一个问题存在。虽然 textwidth() 和 textheight() 将正确计算出文本的大小，但文本的位置是错的。当写下档字母加 g 或 y 时，字母的尾部将在线的下面。为看到此效果，可用下面语句把字体改为 TRIPLEX-FONT：

```
settextstyle(TRIPLEX_FONT, HORIZ_DIR, 1);
```

你会注意到 y 的尾部和逗号都未被擦除。实际发生的是每个笔划字体的顶部都写得比用位映象字体时的位置低几个象素。这就允许“尾部”出现在线的下面。偏移结果大约是 textheight() 计算出的文本高度的四分之一，虽然你也许得对此数据做一些试验。所以为纠正之，可把线条语句改变为：

```
bar(0, th/4, tw, th+th/4);
```

你只想使用笔划字体，正确的方法是在调用该线条例行程序前检查字体类型。下面的程序是被纠正为可用于任一字体的例子：

```
/*
    Graphics text overwriting example #3  (grphex9.c)
    Must link with: graphics.lib
*/
#include <conio.h>
```

```

#include <graphics.h>
char msg1[] = "Baby, come here";
char msg2[] = "I am the ape man";
struct textsettingstype txtinfo;
void main() {
    int gd = DETECT, gm = 0, th, tw;
    initgraph(&gd, &gm, "");
    settextstyle(TRIPLEX_FONT, HORIZ_DIR, 0);
    outtextxy(0, 0, msg1);
    getch();
    th = textheight(msg1); tw = textwidth(msg1);
    gettextsettings(&txtinfo);
    /* clear the first message with the background color */
    setfillstyle(SOLID_FILL, 0);
    if (txtinfo.font) bar(0, th/4, tw, th+th/4); else bar(0, 0, tw, th);
    outtextxy(0, 0, "I am the ape man");
    getch();
    closegraph();
}

```

用不同类型和大小的字体来检验它是否能正确工作。

11.16 突出显示文本

上一节暗示了一种在图形状态下突出显示文本的方法。只需计算出该文本大小、画一个这么大的彩色实线框，然后输出文本即可。只要你用笔划字体，这就行得通。下面的程序显示了一个例子：

```

/*
    Text highlighting in graphics mode. (grphex10.c)
    Must link with: graphics.lib
*/
#include <graphics.h>
#include <conio.h>
#include <stdio.h>
void hilite(int x, int y, char *text, int back_color, int fore_color);
void main() {
    int gd = DETECT, gm = 0;
    initgraph(&gd, &gm, "");
    settextstyle(SANS_SERIF_FONT, HORIZ_DIR, 0);
    hilite(50, 50, "Hello there baby", WHITE, BLACK);
    getch();
    closegraph();
}
void hilite(int x, int y, char *text, int back_color, int fore_color)
/*
    Highlights the text at position (x,y) using the
    specified background and foreground colors.
    The shifted position of stroked fonts is accounted for.
*/

```

```

{
    struct textsettingstype txtinfo;
    int th, tw;
    gettextsettings(&txtinfo);
    th = textheight(text);
    tw = textwidth(text);
    if (txtinfo.font) th++; /* add one just to make sure */
    setfillstyle(SOLID_FILL, back_color);
    bar(x, y, x+tw-1, y+th-1);
    setcolor(fore_color);
    if (txtinfo.font) {
        outtextxy(x, y-th/4, text);
    }
    else {
        outtextxy(x, y, text);
    }
}
}

```

注意与我们的文本覆盖例子相反，没有把线条下移四分之一高度，而是把文本下移了。如此可使线条在同一位置出现，而不论你用的是位映像的还是笔划的字体。

有了这个突出显示例行程序，你就有了在图形状态下做菜单的基本成分。可以试着做一些。

11.17 EGA 的橡皮带式生成线

虽然 Turbo C 允许你用 putimage() 使用 OR, AND NOT 和 XOR 操作，但不幸的是当画像素、线、长方形等时却未提供同样的能力。一个特别有用的操作是 XOR，因为它可使你有效地做动画。画线时用 XOR，你可得到在大多数 CAD 软件包中可以看到橡皮带式生成线效果。用橡皮带式生成线，你可有一个中心点，围绕该点一条线可用某种定点办法被拉伸和旋转。这可以使你一条线被固定画出前看看它怎样。

为使任一般的图形状态得到橡皮带式生成线，你将得绕过 Turbo C 的线例行程序而写自己的。但是，在本节我们将给你显示一种用 EGA 得到橡皮带式生成线的简易方法。虽然详细讲述 EGA 超出了本书范围，但我们将告诉你足够的信息以得到橡皮带式生成线。

EGA（还有 VGA）有许多不同的寄存器可以被设置以给出不同的读和写模式。其中一个这样的寄存器，数据旋转/函数选择寄存器，可用于控制执行已在屏幕上的一个像素与即将用于覆盖之的像素的逻辑运算。表 11.2 显示了可能的操作：

Table 11.2 Function select register select update codes

| 7654 3210 | Operation |
|--------------------------------------|-----------|
| Bit XXX0 0XXX* | REPLACE |
| XXX0 1XXX | AND |
| XXX1 0XXX | OR |
| XXX1 1XXX | XOR |
| * X bits are used for other purposes | |

这里我们关心的是位 3 和 4。X 位是用于别的用途的且通常被设置为 0。为得到一个 XOR 运算，我们可以用 0×18 码。正常操作是 REPLACE，其码是 0×00。

在调用 Turbo C 线例行程序之前设置这个寄存器可使我们把一行与已在屏幕上的东西进行 XOR 运算。这种方式调用该行例行程序两次可使线被画出而后又被擦掉，并把下面的象素恢复到它们原来的颜色。不幸的是，由于 Turbo C 画水平线的方式，它并不能总是正确地恢复这些象素。（似乎是水平线中的某些象素被画了两次，使 XOR 失去了同步）。所以你必须检查水平线并用别的方法画它们。方法之一是用 getimage() 和 putimage() 存储和恢复水平线底下的图形。下面的橡皮带式生成线显示了这种技术的使用：

```
/******  
 * Rubber banding lines example (grphex11.c)  
 * Must link with: mouse.obj, graphics.lib  
 * Works for EGA 640x350 and 640X200 color modes only  
 *****/  
#include <math.h>  
#include <dos.h>  
#include <conio.h>  
#include <graphics.h>  
#include <alloc.h>  
#include "mouse.h"  
void *save_image;  
int been_saved = 0;  
char msg[] = "This is a test of rubber banding lines";  
void rubber_line(int px, int py, int ex, int ey, int drawflag);  
void main() {  
    int gd = EGA, gm = EGAHI;  
    int px, py, ex, ey, mx, my;  
    unsigned int k;  
    initgraph(&gd, &gm, "");  
    init_mouse(MOUSE_NEEDED, gd, gm);  
    /* Allocate enough room for a two-pixel high line the width of  
       the screen.  
    */  
    save_image = calloc(imagesize(0,0,getmaxx(),1),1);  
    while(!mouse_trigger(1));  
    px = mouse_grph_x; py = mouse_grph_y;  
    ex = px; ey = py;  
    mouse_off(1);  
    settextstyle(TRIPLEX_FONT, HORIZ_DIR, 0);  
    outtextxy((getmaxx()-textwidth(msg))/2, getmaxy()/2, msg);  
    mouse_on(1);  
    setlinestyle(DOTTED_LINE, 0, 1);  
    rubber_line(px, py, ex, ey, 1);  
    do {
```

```

mouse_grph_posn(&mx,&my);
if (k = mouse_trigger(1)) { /* ie. mouse button release */
    mouse_off(1);
    rubber_line(px,py,ex,ey,0); /* erase, then make line permanent */
    setcolor(GREEN);
    line(px,py,ex,ey);
    setcolor(WHITE);
    mouse_on(1);
    px = ex; py = ey; /* change pivot point */
}
else {
    if (mx != ex || my != ey) {
        mouse_off(1);
        rubber_line(px,py,ex,ey,0); /* move line via XOR */
        ex = mx; ey = my;
        rubber_line(px,py,ex,ey,1);
        mouse_on(1);
    }
}
} while(k != RIGHT_MOUSE_PRESS);
mouse_reset();
closegraph();
}

void rubber_line(int px, int py, int ex, int ey, int drawflag)
/*
    XOR's a line in EGA HiRes mode, using Turbo C line routine.
    Must use getimage() and putimage() for horizontal lines
    due to way Turbo C draws horizontal lines.
    Works with any line style, but must use normal width.
*/
{
    int sx, fx;
    if (py == ey) { /* horizontal line */
        if (px > ex) { sx = ex; fx = px; } else { sx=ex; fx=ex; }
        if (drawflag) {
            /* must use a two pixel high rectangle */
            getimage(sx,py,fx,pv+1,save_image);
            bean_saved = 1;
            line(sx,py,fx,py);
        }
        else {
            if (bean_saved) putimage(sx,py,save_image,COPY_PUT);
            bean_saved = 0;
        }
    }
    else {
        outp(0x3ce,3); /* select function register */
        outp(0x3cf,0x18); /* set it to XOR */
    }
}

```



```

        line(px,py,ex,ey); /* then draw the line      */
        outp(0x3ce,3);    /* set it back to replace  */
        outp(0x3cf,0);
    }
}

```

这个程序先写一信息到屏幕上，再用鼠标器画橡皮带式生成线。按一下左鼠标器按钮会使当前线被固定画出，而此线的结束点又成了新的中心点。按一下右鼠标器按钮则退出此程序。虽然你在此处使用了系统设置的实线格式，但 XOR 技术也适用于别的线格式（如点线）。但你必须只使用 NORM-WIDTH（单象素）线，而不用 THICK-WIDTH 线。

虽然此程序相当短，但其中有几个精巧之处，当水平线被画出时，它底下的图形就被擦除该线时被恢复而被存储起来。但是，getimage() 和 putimage() 不适用于一个象素高的长方形（线的正常大小），所以用了两个象素高的长方形。还有，虽然你可以任意顺序把坐标传给线例程序，但对 getimage() 和 putimage()，有几条规则必须遵守。左 X 坐标必须小于右边的，同样，顶上的坐标必须小于底部的，这样在必要时可检查并交换这些坐标。

另一问题是如果在错误的时刻按了一下鼠标器按钮则可能使图形的存储和恢复混乱。因此，使用了标志“been-saved”来查看是否需要恢复水平线。

通过输出到用于函数和数据寄存器的两个端口即设定了 XOR 状态。下面的程序序列用于把状态更新到 XOR，画线，而后把状态设置为正常。与往常一样，鼠标器在此期间应被关闭。

```

        outp(0x3ce,3);    /* select function register */
        outp(0x3cf,0x18); /* set it to XOR           */
        line(px,py,ex,ey); /* then draw the line      */
        outp(0x3ce,3);    /* set it back to replace mode */
        outp(0x3cf,0);

```

11.18 总结

本章我们看到了许多 Turbo C 图形函数的使用。看了并了解了这里给出的程序后，你应了解到 Turbo C 中提供的图形函数中的某些特有的效力。我们已指出了使用这些例行程序中的一些问题和应注意的事项。我们也给出了如何改进这些例行程序和如何绕过它们的某些缺陷的建议。

本章以很快的进度给出了这些例行程序，尤其如果你是个初学者的话。即使这样，我们也只顾及到了几个详细的方面。本书没有足够的地方全部讨论它们。一定要透彻地看一看 Turbo C 使用手册的图形部分，因为还有别的函数和排版样式可以使用。你也许想考虑买一本关于图形，尤其是一本专门关于 Turbo C 图形的好书，虽然没有一本这样的书，对许多应用你也能运用得很好。

Listing 11.1 Source code for header file "gpopup.h"

```

/* Graphics popup window toolkit header file (gpopup.h) */
/* Center window code */

```

```

#define CTRWIN 999
/* These help port from text popup windows */
#define xul wc.viewinfo.left
#define yul wc.viewinfo.top
#define xlr wc.viewinfo.right
#define ylr wc.viewinfo.bottom
/* popupwindows save the image underneath, tiled windows don't */
enum windowtype {popup,tile};
/* A structure to hold the box type, and a set of window colors */
typedef struct wincolors_struct {
    struct linesettingtype lineinfo;
    struct fillsettingtype fillinfo;
    struct textsettingtype textinfo;
    struct linesettingtype brdinfo;
    struct viewporttype viewinfo;
    char border_type;
    unsigned char border_color, text_color, hilite_color, back_color;
} wincolors;
/* A structure to hold information for each window */
typedef struct winstruct {
    char *name; /* window title */
    void *image; /* ptr to image save area */
    struct winstruct *under,*over; /* ptrs to window below and above
                                   on popup stack */
    wincolors wc; /* current drawing settings */
    int wd, ht; /* computed width & height */
    int xsave,ysave; /* saved cursor posn */
    enum windowtype wtype; /* window type */
} windesc;
extern windesc *base_win; /* can be used to access whole screen */
extern windesc *curr_win; /* current window in use */
extern wincolors defcolors; /* some other global variables */
extern int graphdriver;
extern int graphmode;
extern char *pathdriver;
/* macros for easy use in removing and selecting window */
#define rmv_win(w) view_win(w,0)
#define slot_win(w) view_win(w,1)
/* Now the prototypes for the popup functions */
extern void init_win(void);
extern windesc *draw_win(int x, int y, int wd, int ht, char *title,
                        enum windowtype wt, wincolors *wc);
extern void view_win(windesc *this, int select);
extern void swap_image(windesc *w);
extern void clr_win(void);
extern void chgviewport(windesc *this);

```

Listing 11.2 Source code for file "gpopup.c"

```

/*****
 * Graphics popup window toolkit for Turbo C. (gpopup.c)
 *****/
#include <stddef.h>
#include <stdarg.h>
#include <stdio.h>
#include <stdlib.h>
#include <alloc.h>
#include <conio.h>

```

```

#include <string.h>
#include <process.h>
#include <graphics.h>
#include "gpopup.h"
#include "mouse.h"
#define MAX(a,b) ((a) > (b) ? (a) : (b))
#define MIN(a,b) ((a) < (b) ? (a) : (b))
/*
 * Global data definitions
 */
windesc *base_win = NULL; /* The "Base Window" pointer */
windesc *curr_win = NULL; /* The current window pointer */
int graphdriver = 0; /* Default to auto detect, */
int graphmode = 0; /* Highest resolution */
char *pathdriver = ""; /* Current directory */
/* Window settings */
wincolors defcolors = {
    (SOLID_LINE, 0, NORM_WIDTH), /* line style - solid line, 1 pixel wide */
    (EMPTY_FILL, 0), /* empty fill, background color */
    (DEFAULT_FONT, HORIZ_DIR, 1, LEFT_TEXT, TOP_TEXT), /* text style */
    (SOLID_LINE, 0, THICK_WIDTH), /* border - solid 3 pixel wide line */
    (0,0,0,0,1), /* viewport loaded by draw_win, but clip set here */
    1, /* will actually have a border */
    7,7,7,0 /* border and text will be gray, hilite will be white,
             and background black */
};
static windesc *top_win; /* window stack pointer */
static void dispose_window_node(windesc *w);
static windesc *push_window_node(void);
static windesc *make_window_node(void);
static void chg_win(windesc *this);
static void fill_win(windesc *w);
void init_win(void)
/*
    init_win initializes the internal variables for the popup windows
    package. This function creates a base window "base_win" which
    represents the entire screen.
    This function MUST be called before any popup routines are used.
*/
{
    initgraph(&graphdriver, &graphmode, pathdriver);
    base_win = make_window_node(); /* allocate a window node */
    getviewsettings(&base_win->wc.viewinfo);
    base_win->wc.text_color = getcolor();
    getfillsettings(&base_win->wc.fillinfo);
    getlinesettings(&base_win->wc.lineinfo);
    gettextsettings(&base_win->wc.textinfo);
    base_win->wc.brdinfo = base_win->wc.lineinfo;
    base_win->xsave = getx();
    base_win->ysave = gety();
    base_win->wtype = tile; /* no saved image underneath */
    base_win->wc.border_type = 0; /* has no border either */
    top_win = base_win; /* set up window stack */
    curr_win = base_win;
}
windesc *draw_win(int x, int y, int wd, int ht, char *title,
    enum windowtype wt, wincolors *wc)

```

```

/*
draw_win creates a new window at a designated screen location.
The viewport will actually reside inside the border.
Besides the usual coordinates for (x,y), you can use the following
codes:
    x = CTRWIN    (Center window in x direction)
    y = CTRWIN    (Center window in y direction)
*/
{
    windesc *w;
    int maxx, maxy;
    w = push_window_node(); /* create and link up a window node */
    maxx = getmaxx(); maxy = getmaxy();
    /* Check for valid window size */
    wd = MAX(wd, 3);
    wd = MIN(wd, maxx);
    ht = MAX(ht, 3);
    ht = MIN(ht, maxy);
    /* Check for centering coordinates and current coordinates.
    Reminder: these are absolute coordinates !!! */
    if (x == CTRWIN) x = (maxx-wd) / 2;
    if (y == CTRWIN) y = (maxy-ht) / 2;
    /*
    Check for valid coordinates.
    */
    x = MAX(x, 0);
    y = MAX(y, 0);
    if ((x+wd) > maxx) x = maxx-wd+1;
    if ((y+ht) > maxy) y = maxy-ht+1;
    /* Store the window parameters */
    w->wc = *wc; /* set up our window line, fill and color settings */
    w->wd = wd;      w->ht = ht;
    w->xul = x;      w->yul = y;
    w->xlr = x + w->wd - 1; w->yhr = y + w->ht - 1;
    w->xsave = 1;      w->ysave = 1;
    w->wtype = wt;      w->name = strdup(title);
    /* allocate and save image underneath if a popup window */
    if (wt == popup) {
        /* use calloc so that the size of the image gets initialized
        to zero!! It will pop up faster first time that way, and
        also keep swap_image from calling put_image with an
        uninitialized buffer */
        w->image =
            calloc(imagesize(w->xul, w->yul, w->xlr, w->yhr), 1);
        swap_image(w);
    }
    fill_win(w); /* draw border and fill window with color */
    chg_win(w); /* change to new window coords and styles */
    return w; /* return new window pointer */
}

static void fill_win(windesc *w)
/* Draws the border, and fills window with background color */
{
    mouse_off(1);
    /* Must go to absolute coordinates so we can get to
    the borders of our window */
    chgviewport(base_win);
}

```

```

setcolor(w->wc.border_color);
setlinestyle(w->wc.brdinfo.linestyle,
             w->wc.brdinfo.upattern,
             w->wc.brdinfo.thickness);
setfillstyle(SOLID_FILL,w->wc.back_color);
/* Tricky!! If border is more than one pixel thick, we must
   compensate so that rectangle pixels are all inside the
   image region we've saved.
*/
if (w->wc.border_type && w->wc.brdinfo.thickness == THICK_WIDTH)
    bar3d(w->xul+1,w->yul+1,w->xlr-2,w->yldr-2,0,0);
else
    bar3d(w->xul,w->yul,w->xlr,w->yldr,0,0);
chgviewport(curr_win); /* Back to previous viewport */
mouse_on(1);           /* restore mouse state */
}
void clr_win(void)
/* Clears the current window */
{
    fill_win(curr_win);
}
void view_win(windesc *this, int select)
/* If select = 1, then view_win moves "this" window to the
   top of the stack and makes it the active window.
   If select = 0, then the window is removed from the stack and erased.
   Note that no moves take place if already at the top.
   If this is merely a tiled window, then it is just selected.
*/
{
    windesc *p;
    if (select && this == top_win) return;
    mouse_off(1); /* make sure mouse hidden */
    /* if this is a popup window, move its image to the top */
    if (this->wtype == popup) {
        p = top_win;
        while(p != this) { /* hide all window above */
            swap_image(p);
            p = p->under;
        }
        swap_image(this); /* and then this window */
        p = this;         /* then put rest of windows back */
        while(p != top_win) {
            p = p->over;
            swap_image(p);
        }
    }
    /* link up window underneath this one, with the window above it */
    if (this == top_win) { /* if this == top_win here, then it is also */
        this->under->over = NULL; /* true that we're removing it for good */
        top_win = this->under;
    }
    else {
        this->under->over = this->over;
        this->over->under = this->under;
    }
    if (select) {

```

```

top_win->over = this; /* move window to the top */
this->under = top_win;
top_win = this;
swap_image(this); /* put back it's image. Does nothing if tiled */
chg_win(this); /* change window */
}
else {
    chg_win(top_win); /* might as well select top window */
    dispose_window_node(this); /* but do before free old window */
}
mouse on(1); /* restore mouse state */
}
static void chg_win(windesc *this)
/* Internal routine to select a window */
{
    curr_win->xsave = getx();
    curr_win->ysave = gety();
    getfillsettings(&curr_win->wc.fillinfo);
    getlinesettings(&curr_win->wc.lineinfo);
    gettextsettings(&curr_win->wc.textinfo);
    chgviewport(this);
    setcolor(this->wc.text_color);
    setfillstyle(this->wc.fillinfo.pattern, this->wc.fillinfo.color);
    setlinestyle(this->wc.lineinfo.linestyle, this->wc.lineinfo.upattern,
        this->wc.lineinfo.thickness);
    settextjustify(this->wc.textinfo.horiz, this->wc.textinfo.vert);
    settextstyle(this->wc.textinfo.font, this->wc.textinfo.direction,
        this->wc.textinfo.charsize);
    moveto(this->xsave, this->ysave);
    curr_win = this; /* selected window is active */
}
void chgviewport(windesc *this)
/* Changes to this windows coordinates. Takes care of border thickness
   in it coordinate calculations.
*/
{
    int delta;
    if (this->wc.border_type)
        delta = this->wc.brdinfo.thickness;
    else delta = 0;
    setviewport(this->xul+delta, this->yul+delta,
        this->xlr-delta, this->yld-delta,
        this->wc.viewinfo.clip);
}
void swap_image(windesc *w)
/*
   This routines swaps the image buffer of a window with what is
   on the screen. If the window is not a popup window, then
   nothing happens.
*/
{
    char *temp_image;
    int nbytes;
    if (w->wtype == popup) {
        nbytes = imagesize(w->xul, w->yul, w->xlr, w->yld);
        temp_image = malloc(nbytes);
    }
}

```

```

    /* Must go to absolute coordinates so we can get to
       the borders of our window */
    chgviewport (base_win);
    mouse_off(1);
    getimage(w->xul,w->yul,w->xlr,w->y1r,temp_image);
    putimage(w->xul,w->yul,w->image,COPY_PUT);
    mouse_on(1);
    /* now back to relative coords */
    chgviewport (w);
    memcpy(w->image,temp_image,nbytes); /* save screen image */
    free(temp_image);
}
}
static windesc *make_window_node(void)
{
    /*
       Make_window_node allocates room for a new window structure,
       and initializes it's links to NULL.
    */
    windesc *q;
    q = (windesc *)malloc(sizeof(windesc));
    q->image = NULL; q->under = NULL; q->over = NULL;
    return q;
}
static windesc *push_window_node(void)
{
    /*
       Push_window_node "pushes" the window w onto the window stack.
    */
    windesc *q;
    q = make_window_node(); /* allocate a window node */
    top_win->over = q; /* link top of stack to new node */
    q->under = top_win; /* link new node to top of stack */
    top_win = q; /* set top of stack to new node */
    return q;
}
static void dispose_window_node(windesc *w)
{
    /*
       Dispose_window_node frees up the image save area of the window,
       and the window structure itself.
    */
    if (w != NULL) { /* safety test for base window */
        if (w->wtype == popup) free(w->image);
        free(w->name); /* free up window title */
        free(w); /* and then the structure itself */
    }
}
}

```

第十二章 高级计划——多维文本系统

本章是你在本书已学到的许多东西的顶点。给出的是一个使用前面各章开发的各种工具和技术的多维文本系统。它使用弹出窗口、鼠标器和键盘 I/O、变化动态串、和带变长记录的文件 I/O。

首先，多维文本确切的是什​​么？本质上讲，它是一种可用以与你看书时所用的常规的、二维的方法不同的方法扫描的文本。正如名字所暗示的那样，多维文本允许你有“多维文本”。这是通过在文本中拥有一些可从其跳到文本别的部分去的特殊连接来实现的。总的来说，这些连接可以是双向的，且可在你浏览该文本的同时很快被创建。在一种方式中，文本被组织进一些卡 (cards) 而且连接不指向文本中任何具体位置，而是指向别的卡。

这里开发的多维文本系统有许多刚讨论过的功能特性。但是，为使之简单，没有支持动态连接。这就意味着运行时不能创建多维文本。相反，你应创建一个特殊文本文件，它含有把文本组织成卡并在它们之间提供连接的命令。该文本文件随后被“编译”入多维文本并存储于另一文件中。它可由“browser”程序读出，此程序允许你浏览多维文本。

我们的多维文本系统由两个程序实现：一个多维文本编译程序和一个多维文本浏览程序。

12.1 多维文本编译程序

多维文本编译程序是一个读 ASCII 文本文件并把它转化为含有特殊多维文本码的 VLR 文件的程序。ASCII 文件含有一些文本行，其间散布着一些命令行。文本被以 `#card` 开头的命令行分为文本行组，称为卡。图 12.1 显示了这种文件的一个例子。指令 `#cavdxxxxx` 告诉编译程序一个新卡开始了，且该卡的名字是 `xxxxxx`。这个名字用做关键字，该卡通过这个名字被引用。

清单 12.1 显示了多维文本编译程序的源程序。即将编译的文件的文件名在命令行给出，并带有假定的扩展名 `.txt`。多维文本文件用同一名字创建且被给定相同的名字，但带有扩展名 `.htx`。

编译程序以两次通过的形式工作。第一次通过扫描文件查找 `#card` 行并收集所有的卡名；第二次通过随后取这个卡名表并扫描文本以寻找匹配。无论在何处找到一个匹配，它在文件中的位置都被记录到一个关键字位置键表中，每个卡都有一个这样的链表。这些链表由浏览程序用于判别何时一个连接已被选择。用这种方式，编译程序为你自动定出卡连接。你所做的全部只是用它们在文本中的名字引用这些卡。

`preprocess()` 函数做第一个通过并扫描该文件以找 `#card` 指令。卡名被认为是在同一行指令的后面。它由该行第一个非空格字符开始而以空格或新行结束。它不应超过 20 个字符长，虽然这可通过宏 `MAXKEYSIZE` 改变。如果未找到名字，则产生系统设置的

名字-cardNN, 其中 NN 是该文件中的当前行号。

```
A sample card text file.

#card Cow
I've seen a lot of cows in my day,
but never like the one i saw riding
a horse out back behind the pig pen.
#card
This is the stuff of an unnamed card.
#card Horse
Talk about a pig of a diff...
oops!!, I meant a horse of a
different color. Did you know
that a cow doesn't think a horse
is better than a zebra?
#card Zebra
Zebras are a lot of fun, except
when you get to know them. Then
they can be real pen pals, that
is, as in pigs.
#card Pig
Piggy does as piggies do. But
they don't do as horses do.
```

Figure 12.1 Sample hypertext source text file, "cards.txt"

process-card() 函数随后重绕此文件并对文本做另一通过, 把文本收集到卡中。一旦一个卡的文本收集好, 该卡的物理大小 (即字符宽度和行数) 就被记录下, 然后文本被传给一个多维文本扫描函数。

函数 hypertext-scan() 用于做扫描。依次试每个卡名以看它是否出现在文本中。当找到一个匹配后, 该匹配的位置就被记录在一个关键字位置链表中。匹配过程忽视上下档。当一个卡被完全扫描过之后, 卡文本和关键字位置链表每个都被存入 VLR 文的记录之中。这两个记录的地址被存储在一个索引表中。

一旦所有的卡都被处理过后, 该索引表本身就被加到该 VLR 文件。索引含有卡号、名字、大小、和它的文本与关键字位置数据地址。现在该文件就可以由多维文本浏览程序读取了。

图 12.2 显示了图 12.1 给出的程序被转换为一个 VLR 文件。这个文件是以十六进制倾卸的形式显示的并说明了 VLR 文件所含的内容。作为一种练习, 看看你是否能用 VLR 文件那一章给出的 VLR 结构译出该文件中的数据。

12.2 多维文本浏览程序

多维文本浏览程序在清单 12.2 中给出。它读一个多维文本文件并允许你用键盘和/或鼠标器浏览它。每个卡都在它自己的窗口中被弹出, 所有的连接都被突出显示。在任一连接上按一下鼠标器按钮或按下回车键都将使你跳到指定的卡去。

有两个特殊的预定义的卡: Home Card 和 Stack Card。Home Card 含有该文件中所含全部关键字的索引。它本身也是个多维文本卡, 所以人们可以用同对别的任何卡同样的

方式从该索引中选取一个新关键字。Stack Card 含有你浏览过的最后 15 个卡的历史记录，按你访问的顺序排列。它也是个多维文本卡，允许你打开该堆栈中任一卡并跳到那里。

图 12.3 给出了一个样例卡屏幕。每个卡的顶部是它的名字和一个命令菜单。这些命令是：

| Command | Use |
|---------|--|
| Home | Takes you to the Home Card |
| Stack | Takes you to the Stack Card |
| Prev | Takes you to the previous card, as defined by the order of cards in the original text file |
| Next | Takes you to the next card, as defined by the order of cards in the original text file |
| Delete | Removes the current card from the stack (You cannot remove a card if it is the only one) |
| Quit | Erases all of the cards and exits |

```

0200 00 00 00 75 00 5a 65 62-72 61 73 20 61 72 65 20 ...u.Zebras are
0210 61 20 6c 6f 74 20 6f 66-20 66 75 6e 2c 20 65 78 a lot of fun, ex
0220 63 65 70 74 0d 0a 77 68-65 6e 20 79 6f 75 20 67 cept..when you g
0230 65 74 20 74 6f 20 6b 6e-6f 77 20 74 68 65 6d 2e et to know them.
0240 20 54 68 65 6e 0d 0a 74-68 65 79 20 63 61 6e 20 Then..they can
0250 62 65 20 72 65 61 6c 20-70 65 6e 20 70 61 6c 73 be real pen pals
0260 2c 20 74 68 61 74 0d 0a-69 73 2c 20 61 73 20 69 , that..is, as i
0270 6e 20 70 69 67 73 2e 0d-0a 00 20 61 20 68 6f 72 n pigs.... a hor
0280 73 65 0d 0a 21 fd 00 00-00 00 10 00 01 00 01 00 se...!.....
0290 05 00 05 00 0b 00 04 00-03 00 06 00 1c 00 04 00 .....
02a0 05 00 04 00 12 00 4e fd-00 00 00 00 3d 00 50 69 .....N.....=Pi
02b0 67 67 79 20 64 6f 65 73-20 61 73 20 70 69 67 67 ggy does as piggy
02c0 69 65 73 20 64 6f 2e 20-42 75 74 0d 0a 74 68 65 ies do. But..the
02d0 79 20 64 6f 6e 27 74 20-64 6f 20 61 73 20 68 6f y don't do as ho
02e0 72 73 65 73 20 64 6f 2e-0d 0a 00 68 65 6e 0d 0a rses do....hen..
02f0 74 68 65 79 20 29 fd 00-00 00 00 18 00 12 00 02 they ).....

0300 00 05 00 04 00 01 00 01-00 03 00 06 00 0f 00 01 .....
0310 00 03 00 06 00 12 00 05-00 05 00 05 00 0e 00 b6 .....
0320 fd 00 00 00 00 a5 00 43-6f 77 00 00 00 be ff 04 .....Cow.....
0330 00 01 00 4c 01 00 00 00-00 ce ff b7 20 00 00 00 ...L.....
0340 a2 00 00 00 26 00 03 00-5f 63 61 72 64 37 00 ff ....&..._card?..
0350 04 00 01 00 4c 01 00 00-00 00 ce ff b7 cc 00 00 ....L.....
0360 00 06 01 00 00 26 00 01-00 48 6f 72 73 65 00 00 .....&...Horse..
0370 ff 04 00 01 00 4c 01 00-00 00 00 ce ff b7 18 01 ....L.....
0380 00 00 c3 01 00 00 22 00-05 00 5a 65 62 72 61 00 .....".Zebra.
0390 00 ff 04 00 01 00 4c 01-00 00 00 00 ce ff b7 fd ....L.....
03a0 01 00 00 84 02 00 00 21-00 04 00 50 69 67 00 61 .....!...Pig.a
03b0 00 00 ff 04 00 01 00 4c-01 00 00 00 00 ce ff b7 ....L.....
03c0 a6 02 00 00 f5 02 00 00-1e 00 02 00 00 00 00 .....
03d0 00 00 00 00 00 00 ff

```

Figure 12.2 Hexadecimal dump image of the hypertext file, "cards.htx" (continued)

```

0000 d6 03 00 00 d6 03 00 00-1f 03 00 00 0b 00 00 00 .....
0010 00 00 00 00 00 00 e5 99-cc 10 db 2f 98 0a 00 00 ...../....
0020 81 fd 00 00 00 00 70 00-49 27 76 65 20 73 65 65 .....p.I've see
0030 6e 20 61 20 6c 6f 74 20-6f 66 20 63 6f 77 73 20 n a lot of cows
0040 69 6e 20 6d 79 20 64 61-79 2c 0d 0a 62 75 74 20 in my day,...but
0050 6e 65 76 65 72 20 6c 69-6b 65 20 74 68 65 20 6f never like the o
0060 6e 65 20 69 20 73 61 77-20 72 69 64 69 6e 67 0d ne i saw riding.
0070 0a 61 20 68 6f 72 73 65-20 6f 75 74 20 62 61 63 .a horse out bac
0080 6b 20 62 65 68 69 6e 64-20 74 68 65 20 70 69 67 k behind the pig
0090 20 70 65 6e 2e 0d 0a 00-00 00 00 00 00 00 00 00 pen.....
00a0 00 00-29 fd 00 00 00 00-18 00 14 00 01 00 03 00 ..).....
00b0 02 00 03 00 03 00 05 00-04 00 1d 00 03 00 03 00 .....
00c0 06 00 00 00 00 00 00 00-00 00 00 00 39 fd 00 00 .....9...
00d0 00 00 28 00 54 68 69 73-20 69 73 20 74 68 65 20 ..(.This is the
00e0 73 74 75 66 66 20 6f 66-20 61 6e 20 75 6e 6e 61 stuff of an unna
00f0 6d 65 64 20 63 61 72 64-2e 0d 0a 00 6e 65 76 65 med card....neve

0100 72 20 6c 69 6b 65 11 fd-00 00 00 00 00 00 14 00 r like.....
0110 01 00 03 00 02 00 03 00-aa fd 00 00 00 00 99 00 .....
0120 5 61 6c 6b 20 61 62 6f-75 74 20 61 20 70 69 67 Talk about a pig
0130 20 6f 66 20 61 20 64 69-66 66 2e 2e 2e 0d 0a 6f of a diff.....o
0140 6f 70 73 21 21 2c 20 49-20 6d 65 61 6e 74 20 61 ops!! , I meant a
0150 20 68 6f 72 73 65 20 6f-66 20 61 0d 0a 64 69 66 horse of a..dif
0160 66 65 72 65 6e 74 20 63-6f 6c 6f 72 2e 20 20 44 ferent color. D
0170 69 64 20 79 6f 75 20 6b-6e 6f 77 0d 0a 74 68 61 id you know..tha
0180 74 20 61 20 63 6f 77 20-64 6f 65 73 6e 27 74 20 t a cow doesn't
0190 74 68 69 6e 6b 20 61 20-68 6f 72 73 65 0d 0a 69 think a horse...i
01a0 73 20 62 65 74 74 65 72-20 74 68 61 6e 20 61 20 s better than a
01b0 7a 65 62 72 61 3f 0d 0a-00 00 00 00 00 00 00 00 zebra?.....
01c0 00 00 00 39 fd 00 00 00-00 28 00 08 00 04 00 03 ...9.....(.....
01d0 00 02 00 13 00 02 00 05-00 04 00 1c 00 04 00 05 .....
01e0 00 04 00 12 00 05 00 05-00 05 00 0e 00 01 00 03 .....
01f0 00 06 00 00 00 00 00 00-00 00 00 00 86 fd 00 .....

```

Figure 12.3 Hexadecimal dump image of the hypertext file, "cards.htx"

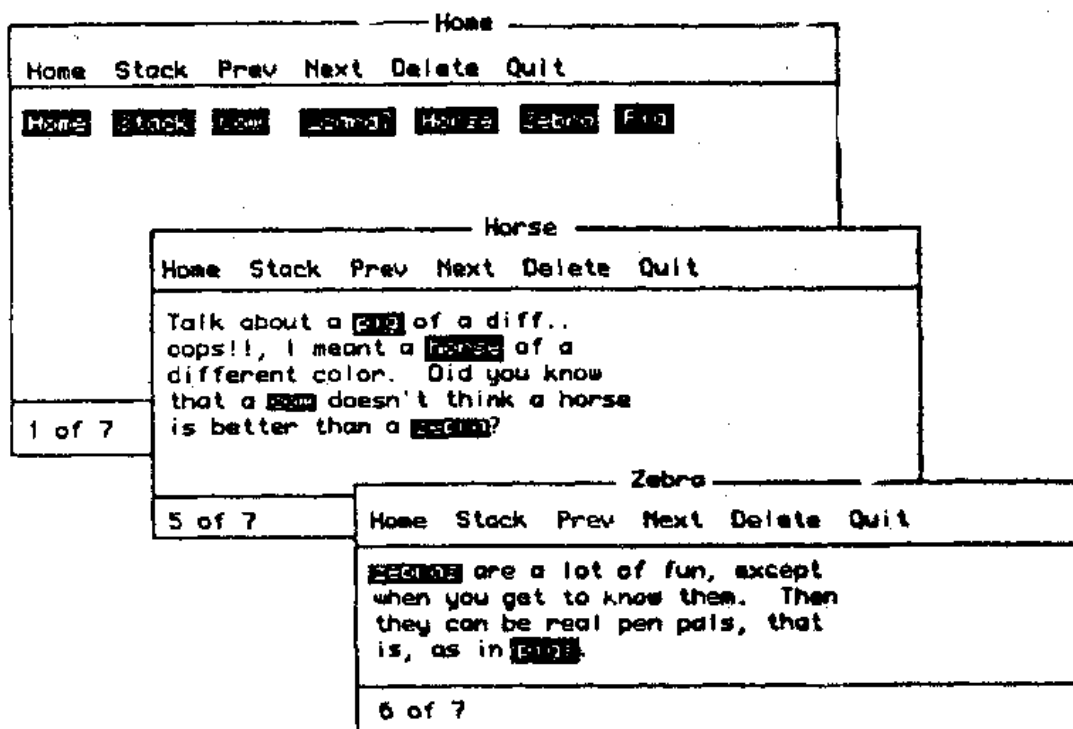


Figure 12.3 Screen image for a sample session with the display of hypertext cards

这些命令可以用鼠标器打开它们来访问，或把光标移到它们之上并按回车，或用下面的键：

| Key | Use |
|--------------------------------------|--|
| Arrows | Moves the cursor around in the card window |
| Shift Arrows | Allows you to move the card around on the screen. Can also move card by dragging left mouse button on its border. Only works for active cards. |
| H, E, "h", "H" | Takes you to the Home Card |
| PGUPKEY, "p", "P" | Takes you to the previous card |
| PGDNKEY, "n", "N" | Takes you to the next card |
| BSKEY, DELKEY, "d", "D" | Deletes the top card from the stack |
| Right Mouse Button, ESCKEY, "q", "Q" | Quits the program |

把卡保存在一个窗口堆栈上用的方式与我们在第四章的移动窗口例子中见到的完全相同。实际上，有些程序是从那个例子中延伸来的，虽然做了细微的改动。无论何时选择一个尚不在屏幕上的卡，它都被弹出。如果它已画出，则它仅仅是移到窗口堆栈的顶部。无论何时选择了一个新卡，旧卡都改为暗色以减少屏幕杂乱。

该程序保持两个并行的栈，一个卡号栈和一个窗口栈。不要把这个窗口栈与弹出窗口软件包中用的那个搞混淆了。多维文本浏览程序中的窗口栈只含有一个由动态串实现的窗口指针链表。但是两个窗口堆栈中的窗口顺序一样都是真的。实际上，如果你选择了 stack Card，则那里显示的卡名将给你一个窗口弹出顺序的清楚图象。

12.3 动态串使用

多维文本浏览程序广泛地使用了动态串并为学习它们提供了一个好的工具。该程序保持了 6 个串。index 串是存储在 VLR 文件中的索引表的拷贝。card 串含有一个卡的实际文本。keypos 串含有当前卡上所示的每个关键字的位置信息（即起始（x 和 y）坐标，宽

度及关键字号)。keystack 串含有记录卡访问历史的卡号栈。这个栈被限制在 15 项，以防止弹出过多的窗口。winstack 串是与 keystack 平行工作的窗口指针栈。indexstk 串只是从 1 到 N 的卡号链表（其中 N 是卡的数目），它用于产生 Home Card。

注意我们怎样通过宏 NULLVSTR 有条不紊地在编译时间初始化每个串的。任何时候你想重新使用一个串，就要用 clrvstr() 函数。正如第七章讨论过的那样（动态串），这种方法比用 dimustr() 好。它允许我们在循环中（本程序中也这样做）重新使用一些串而不必担心过多次地分配内存、或更坏，压根就不用担心。

12.4 多维文本浏览程序中的函数

以下几段简短讨论了多维文本浏览程序中最复杂的一些函数。

12.4.1 main() 函数

main() 函数初始化窗口和文件系统，随后打开在命令行中给出的 VLR 文件。文件名应以不带扩展名的形式给出，假定有一个 .htx 扩展名。文件是为只读存取打开的，所以它不会被无意修改。然后索引表被读入。接着卡栈被初始化，Home Card 被弹出。

WHILE 循环是程序中的主循环。它弹出（如果已画出，则是移到栈顶）当前卡并等待新的选择。这个选择可能是一个新卡标记，或一个菜单命令。在两种情况下，都计算新卡号，画出新卡，此循环重复到选择了“quit”命令为止。那时屏幕上所有的卡都被删除并且程序终止。

12.4.2 paint-htx() 函数

这个函数取一个卡的文本并在指定的窗口打印出来。然后它取一个关键字位置链表并在窗口中并突出显示这些位置。注意：突出显示的方法与我们在第四章的突出显示例子中用的完全相似，但这不是最有效的方法。直接写屏幕仅改变属性将快些。做为一种练习，看看你是否能改变这个程序用这种直接方法工作。

12.4.3 get-next-card() 函数

这个函数用做主事件触发器，它等待一个按键、鼠标器按钮按下、或按钮放开。然后通过一个转换语句选择正确的行动。函数循环到通过按鼠标器左按钮或在一项上按回车键而选择了一个连接或命令为止。如果在当前卡的边界上按下左鼠标器按钮并保持，则你可以把此卡按到屏幕上一个新位置。这也可通过使用移动箭头键来完成。函数返回所选择的连接的关键字或卡号。如果一个连接碰巧实际上是个命令，则该命令用超出系统中卡数范围的数编码。main() 函数循环随后把这些数解码为卡选择或命令。

12.4.4 make-index-card() 函数

这个函数用于为 Home Card 和 Stack Card 产生多维文本。传递一个卡号链表，它与索引表一起使用以创建含有该链表中卡的名字的文本行。每个名字由两个空格分隔，并且这些行被做成能装进一个给定的宽度，。还产生该卡的关键字位置链表。

注意未提供任何方法把过多的卡名放入一个卡，因为浏览程序不支持滚动。

12.4.5 其它函数

on-key()、recolor()、mouse-on-border() 和 move-curr-win() 函数都是直观且从清单即能自我解释的。这些函数中有些是从早先的例子中借用的。

12.5 多维文本系统限制

虽然这里实现的多维文本系统以它自己的权利相当有效，但它确实有一些限制。这些限制是：它不支持文本卷动，不允许连接别名和不允许连接的动态创建和删除。

由于文本卷动不受支持，卡（虽然是自动定大小的）必须装在屏幕里。Home 和 Stack Cards 有固定的大小。来做任何检查看文本是否能实际装入卡中。如果不行，则文本将卷动（因为我们在用 cprintf() 打印文本），但连接位置将不能被正确更新。因为 Home 和 Stack Cards 是固定大小的，所以对你在任一多维文本文件中能有的卡数将有一内在的限制。这个数目决定于卡名的长度。注意只要涉及在文件中存储多维文本，则此限制就不适用。

虽然系统在文本中查寻连接时允许忽略上下档形式，但你必须用与卡名中相同的方式拼写其标记，即不允许有别名。还有，不允许空格出现在标记中。这些限制可通过改进编译程序来处理别名而克服。办法之一如下。

假设你有一名为 ghost 的卡，并且你有一段文本如下：

```
... I just saw Casper, the friendly ghost ...
```

而你想把 Casper 变成 ghost 的一个标记。这可用如下特殊的转换字符来完成：

```
... I just saw \ghost\Casper\, the friendly ghost ...
```

字符\指示编译程序有一别名随后。实际标记是给定的，后随另一\定界符，最后是你希望出现在卡上的文本。当为该卡创建关键字位置链表时，编译程序可以用适当的卡号，而后从文本中删除定界符和实际标记名。

支持动态连接将困难得多。这将不仅要求你有一个卡浏览程序，还要有一个卡编辑程序。该编辑程序允许你从一个卡中插入和删除文本和建立到别的卡的连接（甚至是删除）。一旦你已掌握了使用 VLR 文件和动态串，你就可以试着做了。

Listing 12.1 Source code for the hypertext compiler, "hc.c"

```
/******  
 * The Hypertext Compiler (hc.c)  
 *  
 * Must link with: popup.obj, mouse.obj, saverr.obj,  
 * fileio.obj, vstr.obj, vlr.obj  
 *****/  
#include <stdio.h>  
#include <dir.h>  
#include <string.h>  
#include <process.h>  
#include <ctype.h>  
#include <errno.h>  
#include "popup.h"
```

```

#include "fileio.h"
#include "vstr.h"
#include "vlr.h"
#define MAXLINESIZE 128
#define MAXKEYSIZE 21
#define INDEX ((index_elem *)index->data)
typedef struct {
    char key[MAXKEYSIZE]; /* card name */
    long txt_addr; /* text record address */
    long key_addr; /* keys record address */
    int wd,ht; /* size of card */
} index_elem;
typedef struct {
    int x, y, wd, keyno; /* key position info */
} key_pos_elem;
char infile[MAXPATH];
char outfile[MAXPATH];
char line_buff[MAXLINESIZE];
int fhin; /* input file is a text file */
int fhout; /* output file is a vlr file */
vstr index = NULLVSTR; /* key table */
vstr card = NULLVSTR; /* card data */
vstr posn = NULLVSTR; /* key position data */
static void preprocess(int fhin, vstr *index);
static void process_cards(int fhin, int fhout,
    vstr *index, vstr *card, vstr *posn);
static void hypertext_scan(vstr *card, vstr *index, vstr *posn);
void main(int argc, char *argv[]) {
    init_win(); /* initialize window package */
    init_files(); /* initialize our high level file table */
    if (argc < 2) {
        printf("Usage: >hc cardfile\nExtension of 'txt' assumed\n");
        exit(1);
    }
    else {
        strcpy(infile,argv[1]);
        strcat(infile,".txt");
        /* Open input file for character access, text mode, read only */
        if ((fhin = openfile(infile,"rt",1)) == -1) exit(1);
        strcpy(outfile,argv[1]);
        strcat(outfile,".htx");
        /* Rewrite vlr file, or create it doesn't exist. Start recs at 32 */
        if ((fhout = openvlr(outfile,"w+b",32L)) == NULL) exit(1);
        preprocess(fhin,&index); /* collect keys and posn's */
        process_cards(fhin,fhout,&index,&card,&posn);
        closefile(fhin);
        closefile(fhout);
    }
}
static void preprocess(int fhin, vstr *index)
/* Scan the input file, collect all card names as keys. */
{
    int i, line_no;
    char newkey[MAXKEYSIZE];
    char *p;
    line_no = 0;

```

```

clearstr(index, 30, sizeof(index_elem), 10); /* start with 30 keys */
while(!feof(ft[fhin].fp)) { /* accumulate keys till eof or error */
    if (rdstr(line_buff, MAXLINESIZE, fhin) == -1) break;
    p = line_buff;
    line_no++;
    if (!strncmp(p, "#card", 5)) {
        p += 5;
        while (isspace(*p)) p++; /* look for start of card name */
        if (!*p) {
            sprintf(newkey, "_card%-d", line_no);
            printf("Error: card on line %d has no name, given default\n");
        }
        else {
            i = 0;
            while(!isspace(*p) && i < MAXKEYSIZE-1) /* extract key */
                newkey[i++] = *p++;
            newkey[i] = 0; /* add null to key name */
        }
        printf("found key '%s'\n", newkey);
        strcat(index, newkey);
    }
}
for (i=0; i<index->currlen; i++) {
    printf("key %p\n", (char far *)&(INDEX[i].key));
}

static void process_cards(int fhin, int fhout,
                          vstr *index, vstr *card, vstr *posn)
/*
    For each card, read in the card's text, convert to hypertext, store
    in vlr file.
*/
{
    int i, wd, ht, len;
    long locn, fs, fe, ita, nl;
    rewind(ft[fhin].fp); /* remember to start file over !!! */
    rdstr(line_buff, MAXLINESIZE, fhin); /* get things started */
    for (i=0; i<index->currlen; i++) { /* for each card */
        clearstr(card, 500, sizeof(char), 200); /* init card data */
        while(strncmp(line_buff, "#card", 5) && !feof(ft[fhin].fp)) {
            printf("searching ... %s", line_buff);
            rdstr(line_buff, MAXLINESIZE, fhin);
        }
        if (!feof(ft[fhin].fp)) {
            /* collect up the lines of the card */
            do {
                if (rdstr(line_buff, MAXLINESIZE, fhin) == -1) break;
                printf("%s", line_buff);
                if (strncmp(line_buff, "#card", 5)) {
                    /* Ready to store line, but must force in
                       a carriage return ahead of the line feed
                       so that cprintf will work properly */
                    len = strlen(line_buff);
                    line_buff[len-1] = '\r'; /* put in cr */
                    line_buff[len] = '\n'; /* ahead of lf */
                    vstrins(card, card->currlen, line_buff, ++len);
                }
            } while (1);
        }
    }
}

```



```

    }
    else break;
} while(1);
/* add null so numnewlines will work properly */
vstrcat(card,"");
numnewlines((char *) (card->data), &ht, &wd); /* get size */
/* Scan the card for hypertext, add to file */
hypertext_scan(card, index, posn);
addv1r(fhout, card, &locn);
INDEX[i].txt_addr = locn;
addv1r(fhout, posn, &locn);
INDEX[i].key_addr = locn;
INDEX[i].wd = wd;
INDEX[i].ht = ht;
}
else {
    printf("Unexpected end of file\n");
    exit(1);
}
}
/* okay, now store the key table and store its pointer in
pre-designated location in the file header */
for (i=0; i<index->currlen; i++) {
    printf("key %s %ld %d %d \n",
        INDEX[i].key,
        INDEX[i].txt_addr,
        INDEX[i].key_addr,
        INDEX[i].wd,
        INDEX[i].ht
    );
}
addv1r(fhout, index, &locn);
printf("index starts at %ld\n", locn);
readhdr(fhout, &fs, &fe, &ita, &nl);
ita = locn;
writehdr(fhout, fs, fe, ita, nl);
}
static void hypertext_scan(vstr *card, vstr *index, vstr *posn)
/*
    Searches for keys in the text, and records their position.
    */
{
    char *key, *txt;
    unsigned int i, txtpos;
    int keylen, col, row;
    key_pos_elem kp;
    clrysttr(posn, 10, sizeof(key_pos_elem), 5); /* clear key posn data */
    /* for each key in the table, see if you can find it in the text */
    for (i=0; i<index->currlen; i++) {
        key = INDEX[i].key;
        keylen = strlen(key);
        txt = (char *) card->data; /* point to text */
        txtpos = 0; col = 1; row = 1;
        while(txtpos < card->currlen) {
            if (*txt == '\n') {
                col = 1; row++;
                txt++, txtpos++;
            }

```

```

    }
    else {
        if (!strnicmp(txt, key, keylen)) {
            /* Match, ignoring case, so record posn */
            kp.x = col;    kp.y = row;    kp.wd = keylen;
            kp.keyno = i+2; /* First two key numbers reserved */
            vstrcat(posn, &kp);
            txt += keylen;
            txtpos += keylen;
            col += keylen;
        }
        else {
            txt++; txtpos++; col++;
        }
    }
}
}
}

```

Listing 12.2 Source code for the hypertext browser, "hb.c"

```

/*****
 * The Hypertext Browser (hb.c)
 *
 * Must link with: popup.obj, sayerr.obj, mouse.obj,
 *                fileio.obj, vstr.obj, vlr.obj
 *****/
#include <stdio.h>
#include <dir.h>
#include <string.h>
#include <process.h>
#include <ctype.h>
#include <conio.h>
#include <alloc.h>
#include "popup.h"
#include "mouse.h"
#include "fileio.h"
#include "vstr.h"
#include "vlr.h"
#define MAX(a,b) ((a) > (b) ? (a) : (b))
#define MIN(a,b) ((a) < (b) ? (a) : (b))
#define MAXLINESIZE 128
#define MAXKEYSIZE 21
#define INDEX ((index_elem *) (index.data))
typedef struct {
    char key[MAXKEYSIZE]; /* card name */
    long txt_addr; /* text record address */
    long key_addr; /* keys record address */
    int wd, ht; /* size of card */
} index_elem;
typedef struct {
    int x, y, wd, keyno; /* key position info */
} key_pos_elem;
index_elem index_key = { "Home", 0L, 0L, 52, 10 };
index_elem stack_key = { "Stack", 0L, 0L, 52, 10 };
static void paint_htx(windesc *mw, vstr *keypos, vstr *text);

```

```

static int get_next_card(windesc *w, vstr *keypos);
static int on_key(vstr *keypos, int x, int y);
static void make_index_card(vstr *index, vstr *index_card,
                           vstr *stack, vstr *keypos, int wd);
static void move_curr_win(int x, int y);
static int mouse_on_border(windesc *, int *xcfs, int *yofs);
static void recolor(windesc *w, int attr);
char *cmds[] = {
    "Home", "Stack", "Prev", "Next", "Delete", "Quit"
};
key_pos_elem cmd_menu[6];
vstr index      = NULLVSTR; /* key table */
vstr card       = NULLVSTR; /* card data */
vstr keypos     = NULLVSTR; /* position of keys on screen */
vstr keystack   = NULLVSTR; /* our keystack through the cards */
vstr winstack   = NULLVSTR; /* our stack of card windows */
vstr indxstk    = NULLVSTR; /* an index stack */
wincolors deadcolors = {1, 7, 7, 7, 7};
char infile[MAXPATH]; /* hypertext file name */
int fhtx; /* and handle */
void main(int argc, char *argv[]) {
    int i, cardno, newcardno, command, cx, wd, ht;
    long fs, fe, ita, nl;
    init_win(); /* initialize window package */
    init_files(); /* initialize our high level file table */
    init_mouse(MOUSE_OPTIONAL, MOUSE_TEXT_MODE, MOUSE_TEXT_MODE);
    if (argc < 2) {
        printf("Usage: >hd hypertextfile\nExtension of 'htx' assumed\n");
        exit(1);
    }
    strcpy(infile, argv[1]);
    strcat(infile, ".htx");
    /* open the vlr file */
    if ((fhtx = openvlr(infile, "rb", 32L)) == -1) exit(1);
    clrvstr(&index, 30, sizeof(index_elem), 10); /* start with 30 keys */
    readhdr(fhtx, &fs, &fe, &ita, &nl);
    nl = (nl-1) / 2; /* two recs for every card, minus index rec */
    printf("Should be %d keys\n", nl);
    if (getvlr(fhtx, &index, ita) == -1) exit(1); /* get the keys */
    printf("index says there are %d\n", index.currlen);
    for (i=0; i<index.currlen; i++) {
        printf("%d %s %d\n", i, INDEX[i].key, INDEX[i].txt_addr);
    }
    printf("Press a key to get started ...\n");
    getch();
    clrscr();
    vstrins(&index, 0, &index_key, 1);
    vstrins(&index, 1, &stack_key, 1);
    /* Initialize the card key and window stacks */
    clrvstr(&keystack, 15, sizeof(int), 0);
    clrvstr(&winstack, 15, sizeof(windesc *), 0);
    /* set up our index list */
    clrvstr(&indxstk, 15, sizeof(int), 10);
    for (i=0; i<index.currlen; i++) {
        vstrcat(&indxstk, &i);
    }
}

```

```

cardno = 0; command = -1;
while(cardno != -1) {
    /* Popup the card, and add to the card key and window stacks */
    /* If window already up, don't popup a new one, just clear it */
    for(i=0; i<keystack.currlen; i++) {
        if (((int *) (keystack.data))[i] == cardno) break;
    }
    /* Deaden old window */
    if (keystack.currlen) recolor(curr_win,7);
    if (i < keystack.currlen) {
        slct_win( ((windesc **) (winstack.data))[i] );
        vstrdel(&keystack, i, 1);
        vstrdel(&winstack, i, 1);
    }
    else ( /* Add card, but keep stack from growing too big */
    if (keystack.currlen == 15) {
        rmv_win( ((windesc **) (winstack.data))[winstack.currlen-1] );
        winstack.currlen--;
        keystack.currlen--;
    }
    wd = MAX(INDEX[cardno].wd+2,52) + 1;
    ht = MAX(INDEX[cardno].ht+6,7) + 1;
    draw_win(curr_win->xul+2,curr_win->yul+1,
             wd,ht,INDEX[cardno].key,popup,&monocolors);
    }
    vstrcat(&keystack, &cardno);
    vstrcat(&winstack, &curr_win);
    /* Load hypertext into our scratch card */
    clrsvstr(&card,500,sizeof(char),100);
    clrsvstr(&keypos, 10, sizeof(key_pos_elem), 5);
    if (cardno == 0) {
        make_index_card(&index, &indxstk, &card, &keypos, index_key.wd);
    }
    else {
        if (cardno == 1) {
            make_index_card(&index, &keystack, &card, &keypos, stack_key.wd);
        }
        else {
            getv1r(fhtx, &card, INDEX[cardno].txt_addr);
            getv1r(fhtx, &keypos, INDEX[cardno].key_addr);
            /* relocate the y coordinate */
            for (i=0; i<keypos.currlen; i++) {
                ((key_pos_elem *) (keypos.data))[i].y += 2;
            }
        }
    }
    vstrcat(&card,""); /* null terminate so paint pgm works right */
    /* Print out menu and status information */
    prtfsstr(1, 1, " ", curr_win->wc.hilite_color, -80);
    for(i=0, dx=1; i<6; i++) {
        cmd_menu[i].keyno = index.currlen + i;
        cmd_menu[i].x      = dx;
        cmd_menu[i].y      = 1;
        prtfsstr(dx, 1, cmds[i], curr_win->wc.hilite_color, 80);
        dx                += strlen(cmds[i]);
        cmd_menu[i].wd      = strlen(cmds[i]);
        dx = dx + 2;
    }
}

```

```

}
prtfstr(1, 2,          "\xc4", curr_win->wc.border_color, -80);
prtfstr(1, curr_win->ht-3, "\xc4", curr_win->wc.border_color, -80);
prtfstr(1, curr_win->ht-2, " ", curr_win->wc.hilite_color, -80);
prtfstr(1, curr_win->ht-2, "%3d of %3d",
        curr_win->wc.hilite_color, 80, cardno+1, index.currlen);

/* paint the hypertext, highlighting all keys */
paint_htx(curr_win, &keypos, &card);
/* add the menu commands as keys */
vstrins(&keypos, keypos.currlen, cmd_menu, 6);
newcardno = get_next_card(curr_win, &keypos);
if (newcardno < index.currlen) {
    cardno = newcardno;
    command = -1; /* must set this to something */
}
else { /* must have been a command */
    command = newcardno - index.currlen;
    switch(command) {
        case 0: /* Home card */
            cardno = 0;
            break;
        case 1: /* Stack card */
            cardno = 1;
            break;
        case 2: /* Pgup, up one card, wrap around */
            if (-cardno < 2) cardno = index.currlen-1;
            break;
        case 3: /* Pgdn, down a card, so wrap around */
            if (++cardno == index.currlen) cardno = 2;
            break;
        case 4: /* Delete top card from stack */
            if (keystack.currlen > 1) {
                mv_win( curr_win );
                keystack.currlen--;
                winstack.currlen--;
            }
            cardno = ((int *) (keystack.data))[keystack.currlen-1];
            break;
        case 5: /* Esc, so force it to quit */
            cardno = -1;
            break;
        default: ;
    }
    if (command == 5) break;
}
closefile(fhtx);
mouse_off(0);
for (i = winstack.currlen; i>0; i--) mv_win(curr_win);
}
#define htx ((unsigned char *) (text->data))
static void paint_htx(windesc *w, vstr *keypos, vstr *text)
/*
    Paints the hypertext contained in *text, highlighting all
    the keys.

```

```

*/
{
    int i, j;
    static texel line[80];
    key_pos_elem kp;
    mouse_off(1);
    gotoxy(1,3);
    cprintf("%s",text->data);
    /* hilight the keys */
    for (i=0; i<keypos->currlen; i++) {
        kp = ((key_pos_elem *) (keypos->data))[i];
        kp.x += w->xul; kp.y += w->yul;
        gettext(kp.x, kp.y, kp.x + kp.wd + 1, kp.y, line);
        for (j=0; j<kp.wd; j++) line[j].attr = w->wc.hilite_color;
        puttext(kp.x, kp.y, kp.x + kp.wd + 1, kp.y, line);
    }
    mouse_on(1);
}

static int get_next_card(windesc *w, vstr *keypos)
/*
    Waits for keypress and mouse button release events. When an event occurs
    on a hypertext key, then the new card number is returned, else we
    keep trying.
*/
{
    int x, y, xofs, yofs, done;
    unsigned int key;
    int keyno;
    x = 1; y = 3; keyno = -1;
    do {
        gotoxy(x,y);
        if ((key = mouse_trigger(0)) || (key = mouse_trigger(1))) {
            done = 1;
            switch(key) {
                case LEFT_MOUSE_PRESS:
                    if (mouse_on_border(w, &xofs, &yofs)) {
                        while(button_state()) { /* if yes move till release */
                            mouse_txt_posn(&x,&y);
                            move_curr_win(x-xofs,y-yofs);
                        }
                    }
                    done = 0;
                    break;
                case LEFT_MOUSE_REL:
                    x = mouse_text_x - curr_win->xul;
                    y = mouse_text_y - curr_win->yul;
                    gotoxy(x,y);
                    if ((keyno = on_key(keypos, x, y)) == -1) done = 0;
                    break;
                case CRKEY:
                    if ((keyno = on_key(keypos, x, y)) == -1) done = 0;
                    break;
                case UPKEY:
                    if (!(-y)) y = 1;
                    done = 0;
                    break;
            }
        }
    } while (!done);
    return keyno;
}

```

```

case DOWNKEY :
    if (++y > w->ht-2) y = w->ht-2;
    done = 0;
break;
case LEFTKEY :
    if (!(-x)) x = 1;
    done = 0;
break;
case RIGHTKEY :
    if (++x > w->wd-2) x = w->wd-2;
    done = 0;
break;
case HOMEKEY :
case 0x2368 : /* "h" */
case 0x2348 : /* "H" */
    keyno = index.currilen; /* Home card */
break;
case 0x1f73 : /* "s" */
case 0x1f53 : /* "S" */
    keyno = index.currilen + 1; /* Stack Card */
break;
case PGUPKEY :
case 0x1970 : /* "p" */
case 0x1950 : /* "P" */
    keyno = index.currilen + 2; /* Previous Card */
break;
case PGDNKEY :
case 0x316e : /* "n" */
case 0x314e : /* "N" */
    keyno = index.currilen + 3; /* Next Card */
break;
case BSKEY :
case DELKEY :
case 0x2064 : /* "d" */
case 0x2044 : /* "D" */
    keyno = index.currilen + 4; /* Back One Card */
break;
case RIGHT_MOUSE_REL:
case ESCKEY :
case 0x1071 : /* "q" */
case 0x1051 : /* "Q" */
    keyno = index.currilen + 5; /* quit */
break;
case SHFTUPKEY:
    move_curr_win(curr_win->xul, curr_win->yul-1);
    done = 0;
break;
case SHFTDNKEY:
    move_curr_win(curr_win->xul, curr_win->yul+1);
    done = 0;
break;
case SHFTLEFT:
    move_curr_win(curr_win->xul-1, curr_win->yul);
    done = 0;
break;
case SHFTRIGHT:

```

```

        move_curr_win(curr_win->xul+1, curr_win->yul);
        done = 0;
        break;
    default : ;
    }
} /* end of looking for trigger */
else {
    done = 0;
}
} while(!done);
return keyno;
}
static int on_key(vstr *keypos, int x, int y)
/*
    Given a set of key positions, this function checks to see whether
    the point (x,y) is on any of the keys. If it is, it returns the
    key number, else, it returns -1.
*/
{
    int i;
    key_pos_elem kp;
    for (i = 0; i < keypos->currlen; i++) {
        kp = ((key_pos_elem *) (keypos->data))[i];
        if ((y == kp.y) && (x >= kp.x) && (x <= kp.x + kp.wd - 1))
            return kp.keyno;
    }
    return -1;
}
static void make_index_card(vstr *index, vstr *keystack,
                           vstr *index_card, vstr *keypos, int wd)
/* Given the index table index, a list of keys stack, and a vstr
   to hold the hypertext index_card, this routine creates some
   hypertext for the list of keys, and makes it fit in a window
   that is wd characters wide.
*/
{
    int i, k, keylen, col, row;
    key_pos_elem kp;
    char *keyst;
    for (i=0, col = 1, row = 3; i < keystack->currlen; i++) {
        k = ((int *) (keystack->data))[i];
        keyst = ((index_elem *) (index->data))[k].key;
        keylen = strlen(keyst);
        if ((col + keylen + 2) > wd) {
            col = 1; row++;
            vstrcat(index_card, "\n");
        }
        kp.x = col; kp.y = row;
        kp.keyno = k; kp.wd = keylen;
        vstrcat(keypos, &kp);
        col += keylen + 2;
        vstrins(index_card, index_card->currlen, keyst, keylen);
        vstrins(index_card, index_card->currlen, " ", 2);
    }
}

```



```

    }
}
static void recolor(windesc *w, int attr)
/* Paints the entire window (except borders) with the attr color */
{
    texel *ip;
    int numtexels, i;
    numtexels = (w->xlr - w->xul + 1) * (w->yul - w->yul + 1);
    ip = (texel *)malloc(numtexels*2);
    mouse_off(1);
    gettext(w->xul, w->yul, w->xlr, w->yul, ip);
    for (i=0; i<numtexels; i++) {
        ip[i].attr = attr;
    }
    puttext(w->xul, w->yul, w->xlr, w->yul, ip);
    mouse_on(1);
    free(ip);
}
static int mouse_on_border(windesc *w, int *xofs, int *yofs)
{
    int x, y;
    mouse_txt_posn(&x, &y);
    if ((x >= w->xul && x <= w->xlr &&
        y >= w->yul && y <= w->yul) &&
        (x == w->xul || x == w->xlr ||
        y == w->yul || y == w->yul)) {
        *xofs = x - w->xul; *yofs = y - w->yul;
        return 1;
    }
    return 0;
}
static void move_curr_win(int x, int y)
/*
    Moves the current window. If coordinates haven't changed,
    then nothing happens. Does bounds checking on the new position.
*/
{
    int xsave, ysave;
    x = MAX(x, 1);
    x = MIN(x, 81-curr_win->wd);
    y = MAX(y, 1);
    y = MIN(y, 26-curr_win->ht);
    if (x != curr_win->xul || y != curr_win->yul) {
        xsave = wherex();
        ysave = wherey();
        swap_image(curr_win); /* hide window */
        curr_win->xul = x;
        curr_win->yul = y;
        curr_win->xlr = curr_win->xul + curr_win->wd - 1;
        curr_win->yul = curr_win->yul + curr_win->ht - 1;
        swap_image(curr_win); /* show window */
        /* change window coordinates */
        window(curr_win->xul+1, curr_win->yul+1,
            curr_win->xlr-1, curr_win->yul-1);
        gotoxy(xsave, ysave);
    }
}

```

第十三章 调试器

调试是编程的主要部分，它寻找以判别并删除软件故障。错误的种类与程序一样多。判别软件错误使用了从原始的 printf 协助的到复杂的硬件支持的调试器的各种技术。调试方法依赖于调试器的类型和可用性，还有所检查的应用。Turbo C 2.0 带了一个并用在环境中的多用途的调试器。这缩短了修改和调试程序的周期。

软件错误产生三大类错误：

1. 运行时系统崩溃

根据应用，根据错误产生程序的位置，某些错误导致迫在眉睫的或有条件的系统崩溃。

2. 运行时错误

与第一类不同，这类错误通常使程序停止并把控制还给基础操作系统。与前一类型相似，这些错误或迫在眉睫或是带有条件的。

3. 程序逻辑错误

这种错误允许程序正常终止，但理想的结果和 / 或动作却未正确地执行。这可被认为是由下列两因素之一或两者造成的：

a. 所用算法未正确地被实现。

b. 编程错误

C 中的错误处理是借助于防范性编程技术，而非象 Ada 中的意外处理程序。

13.1 Turbo C 调试器

Turbo C 调试器提供有价值的调试工具。在 Turbo C 环境中，在与调试相关的主菜单中有三个下拉选择项。

1. Run 选择

此菜单选择下提供的这个选择使你能执行下面的几件事：

a. 以通常方式运行一个程序，或运行一个程序直到遇到程序中预设的下一断点为止。

b. 恢复程序以结束一个调试对话。

这使你能够编辑而后恢复调试，重做上次调试对话，或干脆运行该程序。

c. 跟踪选择以两种风格提供：一直跟踪和单步跟踪。一直跟踪一个程序意味着你能跟踪每个函数中语句的执行。这就把整个调试对话置入一完全调试状态，所以能对话。单步跟踪可使你很快执行完一个函数调用。你可用 F7 和 F8 键盘功能键任意选择两者之一。

2 Debugger 选择项

相应的下拉菜单提供的一些选择包括：

a. Evaluate 选择，它可使你敲入一个表达式并观察其结果。你也可给此表达式赋新值。当你想在恢复执行前改正一个变量值或数组位置时是有用的。输入的表达式，它的结

果、和可选择的新值以一单独的单行窗口出现，你只允许在表达式和新值窗口敲入。

b. Call Stack 选择，它显示在到达当前程序位置以前调用的函数的清单，对递归函数，这个选择给你以该函数被调用了多少次的清楚图象。

C. Find Function 选择，它使你能在一个长清单中很快找到一个函数的位置。

d. Screen Swap 选择，这个选择设置用于在环境与输出屏幕之间交换的三个条件之一。选择项是 Always, Smart 和 None。Smart 选择在输出屏幕被控制台输出函数写了之后显示输出屏幕。

3. Break / Watch 选择

这些选择使你能设法观察变量和断点。观察变量被放在一个特殊的显示于 Watch 窗口中的堆栈里。一个观察变量可以是一个简单变量或一个表达式。

下面是一样例说明：

```
int i;
double x;
double dptr = &x;
long numbers[10];
long *lptr = numbers;
char str[] = "Hello world!";
char ptr = str;
struct complex a; /* has double real, imag; fields */
```

下面的清单说明了上面的变量和指针怎样被观察：

| Watch | Contents viewed |
|------------|------------------------------------|
| i | value stored in i |
| x | value stored in x |
| dptr | address of pointer |
| *dptr | value stored in x |
| numbers | address of array numbers |
| numbers[0] | value stored in numbers[0] |
| numbers[2] | value stored in numbers[2] |
| numbers[i] | value stored in numbers[i] |
| lptr | address of pointer |
| *lptr | value stored in numbers[0] |
| *(lptr+2) | value stored in numbers[2] |
| *(lptr+i) | value stored in numbers[i] |
| str | the string "Hello world!" |
| ptr | the string "Hello world!" |
| *ptr | the character "H" stored in str[0] |
| a | address of structure a |
| a.real | the value store in a.real |

你可以添加和编辑一个观察变量，删除最后添加的观察变量，和删除全部观察变量。类似地，你可以触发断点，删除断点，和移到下一断点。Editor 屏幕的光标用于指向被触发的断点。断点被视为是环形链：当你在最后一个断点上请求访问下一个时，调试器就把你带到程序中的第一个去。

13.2 精选的错误

讨论所有类型的错误要求有多卷的教科书。程序错误仅仅受到想象力的限制。它们会在程序的每个方面跳出来！我们将用简单的例子来讨论下列选择的错误：

1. 数值溢出

在此类错误中，数值表达式增值到超出了 Turbo C 支持的限制。下例是一个 double 类型的阶乘函数。在 1 到 170 之间的参数是有效的（用 80×87 协处理器）。这个短程序显示如下：

```
/*
   C program that demonstrates the overflow error for
   factorial that exceed 170, using an 80x87 coprocessor
*/
#include <stdio.h>
#include "conio.h"
double factorial(int);
main()
{
    int n;
    clrscr();
    printf("Enter an integer : ");
    scanf("%d", &n);
    printf("\n\n%d! = %lg\n\n", n, factorial(n));
}
double factorial(int n)
{
    double result = 1.0L;
    while (n > 1)
        result *= (double) n--;
    return result;
}
```

当跟踪上面程序的执行时，激活 Watch / Break 选择并对变量 n 和 result 观察。你将注意到 n 值减少，而 result 值增大。如果提供的参数在有效范围内，则函数正常终止，否则将出现数值溢出错误。当出现溢出错误时，Watch 窗口赋给变量 result 一个 NAN。

2. 越界错误

这是另一类常见错误。在一维数组的情况下，问题表现在试图访问一个超出说明的数组大小范围的元素。在多维数组中，又多出另一类型的错误：在数组各维之间错误地交换了下标。这由下例说明。此例含有一个错误编写的 FOR 循环。该程序不崩溃，但着实也不返回正确结果：

```
/*
   C program that demonstrates the out-of-boundary errors
   that are associated with arrays.
*/
#include <stdio.h>
#include "conio.h"
#define MAX_ROW 10
#define MAX_COL 2
main()
{
```

```

int num[2][10] = { 1,2,3,4,5,6,7,8,9,10,
                  4,7,8,6,3,2,1,6,5,4 };
int i, j = 0, big = num[1][0];
clrscr();
/* -- should be MAX_ROW --
   vvvvvvv                */
for (i = 1; i < MAX_COL; i++)
    if (big < num[1][i]) {
        big = num[1][i];
        j = i;
    }
printf("Largest element is number %d and is equal to %d\n\n",
       j, big);
}

```

用 Turbo C 调试器，在观察循环索引 *i* 和变量 *big* 的同时跟踪上面的程序。FOR 循环只执行两次就退出。这就清楚地指出了问题的本质。用 MAX-ROW 替换 MAX-COL 则可纠正此错误。

3. 未初始化的变量或指针

这代表了另一类型的流行的错误。Turbo C 编译程序能警告你使用了未初始化的变量或指针，但缺乏重初始化的问题却未被判别出。下例中，程序扫描一个数组以确定最大和最小的元素。一指针用于顺序扫描该数组。对第一次循环指针被正确地初始化了，但对第二次循环却未被正确地重初始化。实际上这个程序中有两类错误：未初始化的数组和超界数组访问。后者是由于在第二个 FOR 循环中该指针被增值，它访问了数组 *num* 之外的元素这一事实：

```

/*
C program that demonstrates the out-of-boundary errors
that are associated with arrays. This version shows
the error due omitting re-initialization of a pointer.
The program finds the smallest and largest elements in
an array using a pointer for scanning the array elements
in sequence.
*/
#include <stdio.h>
#include "conio.h"
#define MAX_ROW 20
main()
{
    int num[20] = { 12,23,73,84,15,56,67,98,39,10,
                  54,67,98,16,23,32,41,56,65,74 };
    int i, j = 0, *ptr = num, big = *ptr, small = *ptr;
    clrscr();
    ptr++; /* point to second element */
    for (i = 1; i < MAX_ROW; i++, ptr++)
        if (big < *ptr) {
            big = *ptr;
            j = i;
        }
    /* missing statement is show below */
    /* ptr = num; */
}

```

```

printf("Largest element is num[%d] = %d\n\n", j, big);
ptr++; /* point to second element */
for (i = 1; i < MAX_ROW; i++, ptr++)
    if (small > *ptr) {
small = *ptr;
j = i;
    }
printf("Smallest element is num[%d] = %d\n\n", j, small);
}

```

用 Turbo C 调试器, 观察变量 big, small, 最重要的是观察 *ptr。后者显示指针访问的数组元素。你可在第一个 printf 后设一断点, 因为最大元素被正确报告了。在你跟踪第二个 FOR 循环时, 与 *ptr 相应的值就不是数组 num 的了, 而是在它之外的一个内存区的。插入丢掉的语句 *ptr=num, 则能纠正这个错误。此类错误的另一例子处理未初始化的变量。上一程序已被修改以纠正了与指针相关的错误并删去了赋给变量 big 的值。另外, 数组的第一个元素被改为最大值了。当程序如此运行时, 数 98 而非 120 被报告为最大数。如果你把 120 改得比 98 小, 则问题不会表现出来。这样, 该程序也说明了一种与一算法的错误实现相联系的带有条件的错误情形。程序清单如下:

```

/*
C program that demonstrates the out-of-boundary errors
that are associated with arrays. This version shows
the error due omitting variable initialization.
The program finds the smallest and largest elements in
an array using a pointer for scanning the array elements
in sequence.
The big variable is not initialized. Consequently, the
the first element which is the largest is not detected.
*/
#include <stdio.h>
#include "conio.h"
#define MAX_ROW 20
main()
{
    int num[20] = { 120, 23, 73, 84, 15, 56, 67, 98, 39, 10,
                    54, 67, 98, 16, 23, 32, 41, 56, 65, 74 };
    int i, j = 0, *ptr = num, big, small = *ptr;
    clrscr();
    ptr++; /* point to second element */
    for (i = 1; i < MAX_ROW; i++, ptr++)
        if (big < *ptr) {
            big = *ptr;
            j = i;
        }
    ptr = num; /* re-initialize pointer */
    printf("Largest element is num[%d] = %d\n\n", j, big);
    ptr++; /* point to second element */
    for (i = 1; i < MAX_ROW; i++, ptr++)
        if (small > *ptr) {
            small = *ptr;
            j = i;
        }
}

```

```
printf("Smallest element is num[%d] = %d\n\n", j, small);
scanf("%d", &j);
```

跟踪此程序，并观察变量 big、i 之及指针访问 *ptr。很快就明了了：变量 big 未被初始化就进入了第一个 FOR 循环。

4. 错误的操作等序列

这类错误很是 C 独具的，因为它与涉及++和-操作符的错误有关。把这两个操作符放在一个变量的“逻辑”错误的一边会使变量给一表达式产生一个错误值。下例是上面的阶乘函数的修改形式。操作符-被错误地放在表达式中函数参数 n 前面了：

```
result *= (double) -n;
```

这个错误值得阶乘函数返回 (n-1) 的阶乘而非寻找的参数 n 的阶乘。

```
/*
C program that demonstrates using the wrong operator sequence.
The error is located in the factorial function in the line:
    result *= (double) n;
which should be:
    result *= (double) n--;

#include <stdio.h>
#include "conio.h"
double factorial(int);
main()
{
    int n;
    clrscr();
    printf("Enter an integer : ");
    scanf("%d", &n);
    printf("\n\n%d! = %lg\n\n", n, factorial(n));
}
double factorial(int n)
{
    double result = 1.0L;
    while (n > 1 && n < 170)
        result *= (double) -n;
    return result;
}
```

跟踪上面的程序观察变量 n 和 result。专用 result 相应的值表明函数实际上在计算 n-1 的阶乘因为第一个 -n 在用它的值之前减了 1。

5. 死循环

这类错误使程序执行进入一个恶性循环。问题常表现为循环计数器在每次循环中都被改变的 WHILE 或 Do 循环。遗漏了循环计数器增/减会导致循环停滞不动。下一程序说明了此问题。程序用函数 pos-str() 扫描一个串中的子串以处理串区配码。第一个 WHILE 循环后随一个计十；这是增加值循环控制变量 i 的语句。该语句被括在一个注释中以说明问题中的错误：

/*
 C program that demonstrates the error of creating an infinite loop (a while, in this case) by forgetting to increment the index that makes the loop body progress. The program scans the location of a substring in a string. The error is located in the string-scanning function str_pos().

```
*/
#include <stdio.h>
#include "conio.h"
#include <string.h>
main()
{
    char str[] = "The rain in Spain";
    char *substr;
    int pos;
    clrscr();
    printf("String is '%s'\n\n", str);
    printf("Enter substring -> ");
    gets(substr);
    pos = pos_str(str, substr, 1);
    if (pos > 0)
        printf("Substring matches at character %d\n", pos);
    else
        printf("No match found\n");
}

int pos_str(char* str, char* substr, unsigned int start_index)
{
    int i, j, k, last;
    unsigned int sstrlen = strlen(str);
    unsigned int substrlen = strlen(substr);
    unsigned char nomatch;
    if ((substrlen == 0) || (start_index >= sstrlen))
        return -1;
    k = -1;
    if (sstrlen > substrlen) {
        i = start_index - 1;
        last = sstrlen - substrlen;
        nomatch = 1;
        while ((i <= last) && (nomatch == 1)) {
            /* i++; makes while loop infinite */
            if (substr[0] == str[i]) {
                k = i;
                j = 1;
                i++;
                nomatch = 0;
                while ((j < substrlen) && (nomatch == 0)) {
                    if (substr[j] == str[i]) {
                        i++;
                        j++;
                    }
                    else
                        nomatch = 1;
                }
            }
        }
    }
}
```




```

/* restore index before complete matching was attempted */
if (nomatch == 1) {
    i = k + 1;
    k = -1;
}
) /* if (substr[0] == str[i], */
) /* while ((i <= last) && (nomatch)) */
) /* if (strlen > substrlen) */
return k;

```

跟踪这个程序，观察 *i*、*str*、*substr*、*str[i]* 和 *substr[0]*。输入一个首字母不是 *L* 的这样的串。当执行到达函数的 *WHILE* 循环时，你会看到没有语句这计十；*i* 和 *str[i]* 的值保持不变，因为该循环在无限重复！

6. 无限递归

这类错误与只是不断调用它们自己的递归函数有关。根据函数和错误类型，此错可能受函数参数的某些值影响。下例显示了一个无限循环的递归阶乘函数。这个错误是由于错误的说明条件 `if (n > 1 || n < 170)`

其中 `||` 应为 `&&`：

```

/*
C program that demonstrates the error of infinite recursion
in a wrongly coded factorial function.
*/
#include <stdio.h>
#include "conio.h"
double factorial(int);
main()
{
    int n;
    clrscr();
    printf("Enter an integer : ");
    scanf("%d", &n);
    printf("\n\n%d! = %lg\n\n", n, factorial(n));
}
double factorial(int n)
{
    /* correct if test should be
    (n > 1 && n < 170) */
    if (n > 1 || n < 170)
        return (double) (factorial(n-1) * n);
    else
        return 1.0L;
}

```

跟踪此程序观察变量 *n*。观察 *n* 值怎样持续到 0 以下而进入负数区域。被观察的值使得很容易推断出 *IF* 语句起的作用不对并且未象它应该的那样防止无限递归。