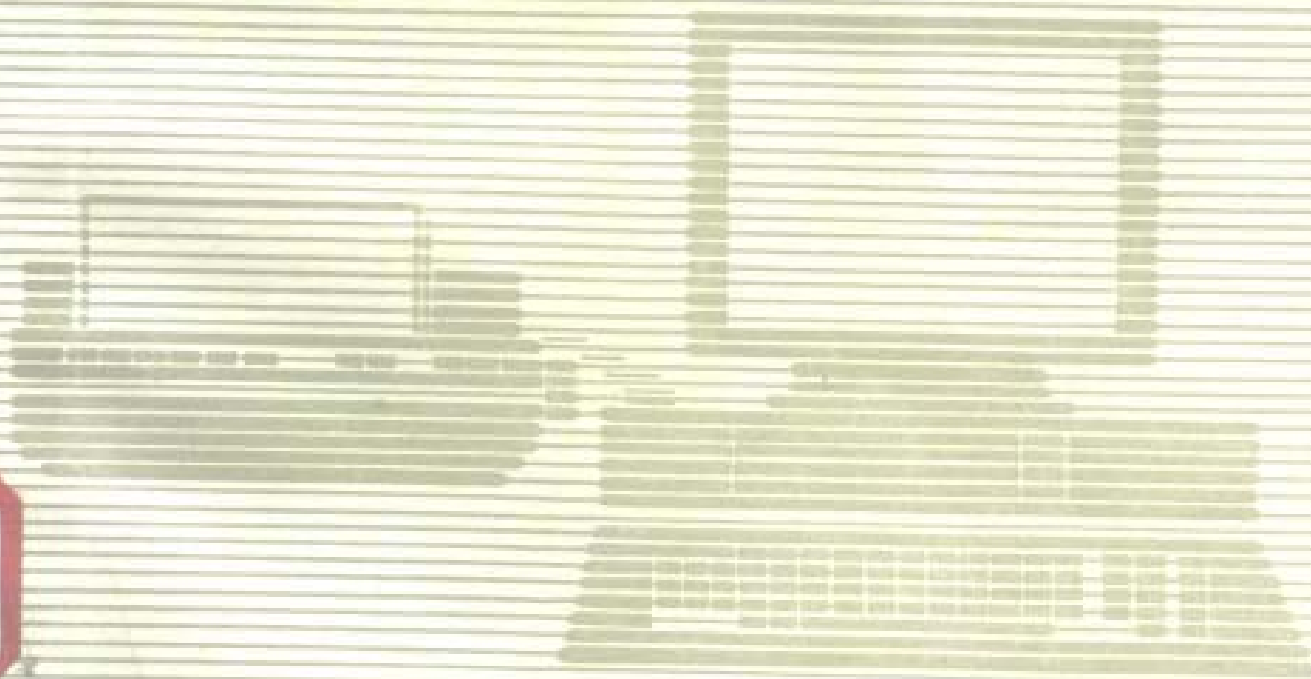


# Turbo C

## 实用高级编程技巧

北京科海培训中心组稿 王军政 编著



上海科学普及出版社

TP3-2  
W250

373634

# Turbo C 实用高级编程技巧

北京科海培训中心 组稿

王军政 编著



上海科学普及出版社

(沪)新登字第 305 号

责任编辑 胡名正 徐丽萍

封面设计 毛增南

**Turbo C 实用高级编程技巧**

北京科海培训中心 组稿

王军政 编著

上海科学普及出版社出版

(上海曹杨路 500 号 邮政编码 200063)

新华书店上海发行所发行 上海市委党校印刷厂印刷

开本 787×1092 1/16 印张 13.5 字数 322000

1993 年 11 月第 1 版 1993 年 11 月第 1 次印刷

ISBN 7-5427-0738-8 / TP·162 定价: 12.00 元

## 内 容 提 要

本书分为两大部分。第一部分介绍 Turbo C 2.0 语言，主要是为初学者编写的，但其中一些内容对熟悉 Turbo C 的读者也非常有用。第二部分介绍 Turbo C 语言的应用技巧，内容包括：西文操作系统下显示汉字技术；与 FoxBASE 接口技术；Turbo C 的高级打印技术；菜单设计技术；与汇编语言的接口技术。本书列举大量实例，全部例程均用中文注释，每个例程都是作者亲自编写并已调试通过，可直接被读者使用。

IS160/58



## 前 言

C语言以其强大的功能成为计算机语言中的佼佼者，而 Turbo C 则以高效的集成开发环境更赢得了广大用户的喜爱。

近几年国内出版了不少关于 Turbo C 语言及其应用的专著，但对于使用 Turbo C 编写显示、打印汉字程序的方法、中文下拉式菜单的设计以及 Turbo C 与 FOXBASE 和汇编语言的接口等技术作全面、系统、详细介绍的书籍几乎没有，可是这些技术非常实用而且应用普遍。为此，本书作者将其实际使用 Turbo C 编程所积累的一些经验和体会汇集成册，奉献给广大 Turbo C 爱好者。

本书分为两大部分共十三章。第一部分介绍 Turbo C 2.0 语言，主要是为初学者编写的，但其中一些内容，如连续使用字符输入函数时存在的问题及解决途径、键盘的动态扫描、BIOS 和 DOS 功能的调用方法、程序的调试等内容对熟悉 Turbo C 的读者也是非常有用的。第二部分介绍 Turbo C 语言的应用技巧，内容包括：西文操作系统下读取 16×16 点阵字模、24×24 点阵字模按水平或垂直方向屏幕显示或放大显示不同字体、不同颜色汉字的技术；读取 FOXBASE 数据库或从 FOXBASE 以传递参数的办法调用 Turbo C 程序的技术；高分辨率屏幕图形的打印机输出和打印机驱动程序的编写技术；中文下拉式菜单的设计技术；调用汇编子程序和行间嵌入汇编语句的 Turbo C 程序编写及编译、链接技术。

本书列举大量实例，全部例程均用中文注释，每个例程都是作者亲自编写并已调试通过，可直接被读者使用。

因作者水平所限，经验不足，加之时间仓促，书中难免有不少错误和疏漏，诚恳接受广大读者和有关专家的批评指正。

本书在出版过程中，得到了科海培训中心华根娣、夏非彼两位老师的大力协助，在此向她们表示衷心的感谢！

编 者  
1993 年 5 月

# 目 录

第一部分 Turbo C 2.0 语言 .....	( 1)
第一章 Turbo C 2.0 集成开发环境 .....	( 1)
1.1 C 语言概述 .....	( 1)
1.1.1 C 语言的产生与发展 .....	( 1)
1.1.2 C 语言的特点 .....	( 1)
1.2 Turbo C 2.0 概述 .....	( 2)
1.2.1 Turbo C 的产生与发展 .....	( 2)
1.2.2 Turbo C 2.0 基本配置要求 .....	( 2)
1.2.3 Turbo C 2.0 软盘内容简介 .....	( 2)
1.3 学习本书应具有的软件环境 .....	( 3)
1.4 Turbo C 2.0 的安装和启动 .....	( 4)
1.5 Turbo C 2.0 集成开发环境的使用 .....	( 4)
1.5.1 主菜单 .....	( 5)
1.5.2 Turbo C 的配置文件 .....	(11)
第二章 数据类型、变量和运算符 .....	(12)
2.1 Turbo C 程序的一般组成部分 .....	(12)
2.2 数据类型 .....	(14)
2.2.1 整型 (int) .....	(15)
2.2.2 浮点型 (float) .....	(15)
2.2.3 字符型 (char) .....	(16)
2.2.4 指针型 (*) .....	(17)
2.2.5 无值型 (void) .....	(17)
2.3 关键字和标识符 .....	(17)
2.3.1 关键字 .....	(17)
2.3.2 标识符 .....	(18)
2.4 变量 .....	(18)
2.4.1 变量说明 .....	(18)
2.4.2 变量种类 .....	(18)
2.4.3 变量存储类型 .....	(20)
2.4.4 数组变量 .....	(21)
2.4.5 变量的初始化和赋值 .....	(22)
2.5 运算符 .....	(28)
2.5.1 算术运算符 .....	(28)
2.5.2 逻辑运算符关系和运算符 .....	(30)

2.5.3 按位运算符 .....	(31)
2.5.4 Turbo C 的特殊运算符 .....	(32)
2.5.5 Turbo C 运算符的优先级 .....	(33)
<b>第三章 输入输出函数 .....</b>	<b>(35)</b>
3.1 标准输入输出函数 .....	(35)
3.1.1 格式化输入输出函数 .....	(35)
3.1.2 非格式化输入输出函数 .....	(39)
3.2 文件的输入输出函数 .....	(42)
3.2.1 标准文件函数 .....	(43)
3.2.2 非标准文件函数 .....	(49)
<b>第四章 流程控制语句 .....</b>	<b>(51)</b>
4.1 条件语句 .....	(51)
4.2 循环语句 .....	(52)
4.2.1 for 循环 .....	(52)
4.2.2 while 循环 .....	(53)
4.2.3 do-while 循环 .....	(55)
4.3 开关语句 .....	(55)
4.4 break、continue 和 goto 语句 .....	(57)
4.4.1 break 语句 .....	(57)
4.4.2 continue 语句 .....	(57)
4.4.3 goto 语句 .....	(58)
<b>第五章 结构、联合和枚举 .....</b>	<b>(60)</b>
5.1 结构 (struct) .....	(60)
5.1.1 结构说明和结构变量定义 .....	(60)
5.1.2 结构变量的使用 .....	(61)
5.1.3 结构数组和结构指针 .....	(63)
5.1.4 结构的复杂形式 .....	(65)
5.2 联合 (union) .....	(67)
5.2.1 联合说明和联合变量定义 .....	(67)
5.2.2 结构和联合的区别 .....	(68)
5.3 枚举 (enum) .....	(69)
5.4 类型说明 .....	(70)
5.5 预处理指令 .....	(71)
<b>第六章 函数 .....</b>	<b>(74)</b>
6.1 函数的说明和定义 .....	(74)
6.1.1 函数说明 .....	(74)
6.1.2 函数定义 .....	(75)
6.2 函数的调用 .....	(75)
6.2.1 函数的简单调用 .....	( 75)

6.2.2 函数的参数传递 .....	( 76)
6.2.3 函数的递归调用 .....	( 81)
6.3 函数作用范围 .....	( 82)
<b>第七章 字符屏幕和图形函数 .....</b>	<b>( 83)</b>
7.1 字符屏幕函数 .....	( 83)
7.1.1 文本窗口的定义 .....	( 83)
7.1.2 文本窗口颜色的设置 .....	( 83)
7.1.3 窗口内文本的输入输出函数 .....	( 85)
7.1.4 有关屏幕操作的函数 .....	( 86)
7.2 图形函数 .....	( 88)
7.2.1 图形模式的初始化 .....	( 88)
7.2.2 独立图形运行程序的建立 .....	( 90)
7.2.3 屏幕颜色的设置和清屏函数 .....	( 91)
7.2.4 基本图形函数 .....	( 93)
7.2.5 封闭图形的填充 .....	( 96)
7.2.6 图形窗口和图形屏幕操作函数 .....	(100)
7.2.7 图形模式下的文本输出 .....	(103)
<b>第八章 Turbo C 实用编程 .....</b>	<b>(107)</b>
8.1 含汉字输入输出的程序的编制 .....	(107)
8.2 Turbo C 提供的 BIOS 和 DOS 系统调用函数 .....	(111)
8.2.1 键盘操作函数 bioskey( ) .....	(111)
8.2.2 打印机操作函数 biosprint( ) .....	(114)
8.2.3 DOS 软中断功能调用函数 intdos( ) .....	(117)
8.2.4 BIOS 和 DOS 软中断调用的函数 int86( ) .....	(118)
8.2.5 其它的系统调用函数 .....	(118)
8.3 字符串函数和数字字符串与数值的转换函数 .....	(120)
8.3.1 字符串函数 .....	(120)
8.3.2 数字字符串与数值的转换函数 .....	(122)
8.4 动态内存分配、过程控制和数学运算函数 .....	(123)
8.4.1 动态内存分配函数 .....	(123)
8.4.2 过程控制函数 .....	(124)
8.4.3 数学运算函数 .....	(126)
8.5 Turbo C 集成开发环境下程序的调试 .....	(126)
8.5.1 编译时的常见错误 .....	(127)
8.5.2 链接时的常见错误 .....	(127)
8.5.3 运行时的常见错误 .....	(127)
8.6 Turbo C 的命令行编译 .....	(128)



第二部分 Turbo C 2.0 应用技术专题 .....	(130)
第九章 西文状态下的汉字显示技术 .....	(130)
9.1 西文状态下显示 16×16 点阵汉字 .....	(130)
9.1.1 16×16 点阵汉字字模的存储格式 .....	(130)
9.1.2 西文状态下显示 16×16 点阵汉字的实现 .....	(130)
9.2 西文状态下显示 24×24 点阵汉字 .....	(136)
9.2.1 24×24 点阵汉字字模的存储格式 .....	(136)
9.2.2 西文状态下显示 24×24 点阵汉字的实现 .....	(137)
9.3 西文状态下放大任意倍数显示不同字体的 24×24 点阵汉字 .....	(141)
第十章 与 FOXBASE(dBASE)接口技术 .....	(147)
10.1 Turbo C 直接读取 FOXBASE 的数据库 .....	(147)
10.1.1 FOXBASE 数据库的结构 .....	(147)
10.1.2 Turbo C 读取数据库的实现 .....	(149)
10.2 FOXBASE 给 Turbo C 传递参数 .....	(152)
第十一章 Turbo C 的高级打印技术 .....	(155)
11.1 利用打印机驱动程序放大打印汉字 .....	(155)
11.2 VGA 高分辨率(640×480)屏幕图形的打印机输出 .....	(157)
11.2.1 M1724 打印机的控制命令 .....	(157)
11.2.2 VGA 高分辨率屏幕图形的打印机输出程序 .....	(158)
11.3 自编打印机驱动程序 .....	(164)
第十二章 菜单设计技术 .....	(173)
12.1 西文下拉式菜单的设计 .....	(173)
12.2 中文窗口式菜单的设计 .....	(177)
12.3 FOXBASE 和 Turbo C 程序交替使用时的菜单设计 .....	(182)
12.4 中文下拉式菜单的设计 .....	(186)
第十三章 与汇编语言的接口技术 .....	(193)
13.1 Turbo C 调用汇编子程序 .....	(193)
13.1.1 Turbo C 与汇编语言的接口方法 .....	(193)
13.1.2 自动产生汇编语言的框架程序 .....	(196)
13.1.3 接口程序的编译、链接和运行 .....	(199)
13.2 Turbo C 行间嵌入汇编语句 .....	(200)
附录 ASCII 字符代码表 .....	(203)

# 第一部分 Turbo C 2.0 语言

这一部分详细地讲述 Turbo C 2.0 语言。主要包括：集成开发环境的使用、变量类型、操作运算、输入输出函数、控制流程语句以及字符和图形屏幕函数等内容。

## 第一章 Turbo C 2.0 集成开发环境

本章首先介绍 C 和 Turbo C 的特点以及 Turbo C 2.0 的软盘内容、安装方法，最后介绍 Turbo C 的集成开发环境。通过本章的学习，读者可以全面系统地学习 Turbo C 集成开发环境的使用方法，这也是学习 Turbo C 必不可少的。

### 1.1 C 语言概述

#### 1.1.1 C 语言的产生与发展

C 语言是 1972 年由美国的 Dennis Ritchie 设计发明的，并首次在 UNIX 操作系统的 DEC PDP-11 计算机上使用。它由早期的编程语言 BCPL ( Basic Combined Programming Language ) 发展演变而来。在 1970 年，AT&T 贝尔实验室的 Ken Thompson 根据 BCPL 语言设计出较先进的并取名为 B 的语言，最后导致了 C 语言的问世。

随着微型计算机的日益普及，出现了许多 C 语言版本。由于没有统一的标准，因此这些 C 语言版本出现了一些不一致的地方。为了改变这种情况，美国国家标准研究所 (ANSI) 为 C 语言制定了一套 ANSI 标准，成为现行的 C 语言标准。

#### 1.1.2 C 语言的特点

C 语言发展如此迅速，而且成为最受欢迎的语言之一，主要因为它具有强大的功能。许多著名的系统软件，如 dBASE III PLUS、dBASE IV 都是由 C 语言编写的。用 C 语言加上一些汇编语言子程序，就更能显示 C 语言的优势了，象 PC-DOS、WORDSTAR 等就是用这种方法编写的。归纳起来，C 语言具有下列特点：

##### 1. C 是中级语言

它把高级语言的基本结构和语句与低级语言的实用性结合起来。C 语言可以象汇编语言一样对位、字节和地址进行操作，而这三者正是计算机最基本的工作单元。

##### 2. C 是结构式语言

结构式语言的显著特点是代码及数据的分隔化，即程序的各个部分除了必要的信息交流外彼此独立。这种结构化方式可使程序层次清晰，便于使用、维护以及调试。C 语言是

以函数形式提供给用户的，这些函数可让用户方便地调用，并具有多种循环、条件语句控制程序流向，从而使程序完全结构化。

### 3. C 语言功能齐全

C 语言具有各种各样的数据类型，并引入了指针概念，可使程序效率更高。另外，C 语言也具有强大的图形功能，支持多种显示器和驱动器，而且计算功能、逻辑判断功能也比较强大，可以实现决策目的。

### 4. C 语言适用范围广

C 语言还有一个突出的优点就是适合于多种操作系统，如 DOS、UNIX，也适用于多种机型。

## 1.2 Turbo C 2.0 概述

### 1.2.1 Turbo C 的产生与发展

Turbo C 是美国 Borland 公司的产品，Borland 公司是一家专门从事软件开发、研制的大公司。该公司相继推出了一套 Turbo 系列软件，如 Turbo BASIC、Turbo Pascal、Turbo Prolog，这些软件很受用户欢迎。该公司在 1987 年首次推出了 Turbo C 1.0 产品，其中使用了全然一新的集成开发环境，即使用了一系列下拉式菜单，将文本编辑、程序编译、连接以及程序运行一体化，大大方便了程序的开发。1988 年，Borland 公司又推出 Turbo C 1.5 版本，增加了图形库和文本窗口函数库等，而 Turbo C 2.0 则是该公司 1989 年出版的。Turbo C 2.0 在原来集成开发环境的基础上增加了查错功能，并可以在 Tiny 模式下直接生成.COM(数据、代码、堆栈处在同一 64K 内存中)文件，还可对数学协处理器(支持 8087/80287/80387 等)进行仿真。

Borland 公司后来又推出了面向对象的程序软件包 Turbo C++，它继承发展了 Turbo C 2.0 的集成开发环境，并包含了面向对象的基本思想和设计方法。

1991 年为了适配 Microsoft 公司的 Windows 3.0 版本，Borland 公司又将 Turbo C++作了更新，即 Turbo C 的新一代产品 Borland C++ 2.0 也已经问世了。

### 1.2.2 Turbo C 2.0 基本配置要求

Turbo C 2.0 可运行于 IBM-PC 系列微机，包括 XT、AT 及 IBM 兼容机。此时要求 DOS 2.0 或更高版本支持，并至少需要 448K 的 RAM，可在任何彩色、单色 80 列监视器上运行。它能支持数学协处理器芯片，也可进行浮点仿真，这将加快程序的执行。

### 1.2.3 Turbo C 2.0 软盘内容简介

Turbo C 2.0 有六张低密软盘(或两张高密软盘)。下面对 Turbo C 2.0 的主要文件作一简单介绍：

INSTALL.EXE	安装程序文件
TC.EXE	集成编译器
TCINST.EXE	集成开发环境的配置设置程序
TCHELP.TCH	帮助文件

THELP.COM	读取TCHELP.TCH的驻留程序
README	关于Turbo C的信息文件
TCCONFIG.EXE	配置文件转换程序
MAKE.EXE	项目管理工具
TCC.EXE	命令行编译器
TLINK.EXE	Turbo系列连接器
TLIB.EXE	Turbo系列库管理工具
C07.OBJ	不同模式启动代码
C7.LIB	不同模式运行库
GRAPHICS.LIB	图形库
EMU.LIB	8087仿真库
FP87.LIB	8087库
*.H	Turbo C头文件
*.BGI	不同显示器图形驱动程序
*.C	Turbo C例行程序(源文件)

其中的?分别为:

T	Tiny(微型模式)
S	Small(小模式)
C	Compact(紧凑模式)
M	Medium(中型模式)
L	Large(大模式)
H	Huge(巨大模式)

### 1.3 学习本书应具有的软硬件环境

本书以 Turbo C 2.0 为蓝本, 因此必须有一套齐全的 Turbo C 2.0 版本。除此之外, 要很顺利的使用本书中的例程, 特别是第二部分的应用专题例行程序, 应具有以下软、硬件配置。

硬件:	80286(或兼容机)主机 VGA(或TVGA)分辨率(640×480)显示器 M1724打印机
软件:	DOS 3.0以上版本操作系统 UCDOS 2.0汉字操作系统 FOXBASE 2.0数据库管理语言 MASM 4.0以上宏汇编编译程序

另外, 本书介绍的函数和语句许多是 C 语言标准, 即其它 C 语言中也有, 但作者未对此进行区分而叙述成 Turbo C 的函数和语句, 特此说明。

### 1.4 Turbo C 2.0 的安装和启动

Turbo C 2.0 的安装非常简单, 只要将 1 盘插入 A 驱动器中, 在 DOS 的 "A>" 下键入:

A>INSTALL

即可，此时屏幕上显示三种选择：

1. 在硬盘上创建一个新目录来安装整个 Turbo C 2.0 系统。
  2. 对 Turbo C 1.5 更新版本。
- 这样的安装将保留原来对选择项、颜色和编辑功能键的设置
3. 为只有两个软盘而无硬盘的系统安装 Turbo C 2.0。

这里假定按第一种选择进行安装，只要在安装过程中按对盘号的提示，顺次插入各个软盘，就可以顺利地进行安装，安装完毕将在 C 盘根目录下建立一个 TC 子目录，TC 下还建立了两个子目录 LIB 和 INCLUDE，LIB 子目录中存放库文件，INCLUDE 子目录中存放所有头部文件。

运行 Turbo C 时, 只要在 TC 子目录下键入 TC 并回车即可进入 Turbo C 集成开发环境。

### 1.5 Turbo C 2.0 集成开发环境的使用

进入 Turbo C 集成开发环境后，屏幕上显示：

File Edit Run Compile Project Options Debug Break/watch

Edit

Line 1 Col 1 Insert Indent Tab File Unindent C:\NONAME.C

Message

F1 -Help F5 -Zoom F6 -Switch F7 -Trace F8 -Step F9 -Make F10 -Menu

其中顶上一行为 Turbo C 主菜单，中间窗口为编辑区，接下来是信息窗口，最底下一行为参考行。这四个窗口构成了 Turbo C 的主屏幕，以后的编程、编译、调试以及运

行都将在这个主屏幕中进行。下面详细介绍主菜单的内容。

### 1.5.1 主菜单

主菜单在 Turbo C 主屏幕的顶上一行, 显示下列内容:

File Edit Run Compile Project Options Debug Break / watch

除 Edit 外, 其它各项均有子菜单, 只要用 Alt 加上某项的第一个字母(即大写字母), 就可进入该项的子菜单中。

#### 一、File(文件)菜单

按 Alt+F 可进入 File 菜单, 该菜单包括以下内容:

##### • Load (加载)

装入一个文件, 可用类似 DOS 的通配符(如 \*.C)来进行列表选择。也可装入其它扩展名的文件, 只要给出文件名(或只给路径)即可。该项的热键为 F3, 即只要在主菜单中按 F3 就可进入该项, 而不需要先进入 File 菜单再选此项。

##### • Pick(选择)

将最近装入编辑窗口的 8 个文件列成一个表让用户选择, 选择后将该程序装入编辑区, 并将光标置在上次修改过的地方。其热键为 Alt+F3。

##### • New(新文件)

说明文件是新的, 缺省文件名为 NONAME.C, 存盘时可改名。

##### • Save(存盘)

将编辑区中的文件存盘, 若文件名是 NONAME.C 时, 将询问是否更改文件名, 其热键为 F2。

##### • Write to(存盘)

可由用户给出文件名将编辑区中的文件存盘, 若该文件已存在, 则询问是否覆盖。

##### • Directory(目录)

显示目录及目录中的文件, 并可由用户选择。

##### • Change dir (更换目录)

显示当前目录, 用户可以更换显示的目录。

##### • Os shell (暂时退出)

暂时退出 Turbo C 到 DOS 提示符下, 此时可以运行 DOS 命令。若又想回到 Turbo C 中, 只要在 DOS 状态下键入 EXIT 即可。

##### • Quit (退出)

退出 Turbo C, 返回到 DOS 操作系统中, 其热键为 Alt+X。

注:

● 以上各项可用光标键移动色条进行选择, 回车则执行。也可用每一项的第一个大写字母直接选择。若要退到主菜单或从它的下一级菜单列表框退回均可用 Esc 键, Turbo C 的所有菜单均采用这种方法进行操作, 以下不再说明。

#### 二、Edit(编辑)菜单

按 Alt+E 可进入编辑菜单, 若再回车, 则光标出现在编辑窗口, 此时用户可以进行文本编辑。

编辑方法基本上与 Wordstar 相同, 可用 F1 键获得有关编辑方法的帮助信息。  
与编辑有关的功能键如下:

F1	获得Turbo C编辑命令的帮助信息
F5	扩大编辑窗口到整个屏幕
F6	在编辑窗口与信息窗口之间进行切换
F10	从编辑窗口转到主菜单

编辑命令简介:

Pg Up	向前翻页
Pg Dn	向后翻页
Home	将光标移到所在行首
End	将光标移到所在行尾
Ctrl+Y	删除光标所在的一行
Ctrl+T	删除光标所在处的一个词
Ctrl+KB	设置块头
Ctrl+KK	设置块尾
Ctrl+KV	块移动
Ctrl+KC	块拷贝
Ctrl+KY	块删除
Ctrl+KR	读文件
Ctrl+KW	存文件
Ctrl+KP	块文件打印
Ctrl+F1	如果光标所在处为Turbo C库函数, 则获得有关该函数的帮助信息
Ctrl+Q[	查找Turbo C双界符的后匹配符
Ctrl+Q]	查找Turbo C双界符的前匹配符

注:

● Turbo C 的双界符包括以下几种符号:

花括号	{ 和 }
尖括号	< 和 >
圆括号	( 和 )
方括号	[ 和 ]
注释符	/ * 和 * /
双引号	"
单引号	'

● Turbo C 在编辑文件时还有一种功能, 就是能够自动缩格。所谓自动缩格是指在硬回车之后, 光标定位和上一个非空字符对齐。在编辑窗口中, Ctrl+OL 为自动缩格开关的控制键。

三、Run(运行)菜单

按 Alt+R 可进入 Run 菜单, 该菜单有以下各项:

- Run(运行程序)

运行由 Project / Project name 项指定的文件名或当前编辑区的文件。如果对上次编译后的源代码未做过修改, 则直接运行到下一个断点(没有断点则运行到结束), 否则先进行编译、连接后才运行, 其热键为 Ctrl+F9。

- Program reset(程序重启)

中止当前的调试, 释放分配给程序的空间, 其热键为 Ctrl+F2。

- Go to cursor(运行到光标处)

调试程序时使用, 选择该项可使程序运行到光标所在行。光标所在行必须是一条可执行语句, 否则提示出错。其热键为 F4。

- Trace into(跟踪进入)

在执行一条调用由其它用户定义的子函数时, 若用 Trace into 项, 则执行长条将跟踪到该子函数内部去执行, 其热键为 F7。

- Step over(单步执行)

执行当前函数的下一条语句, 即使是用户函数调用, 执行长条也不会跟踪进函数内部, 其热键为 F8。

- User screen(用户屏幕)

显示程序运行时在屏幕上显示的结果。其热键为 Alt+F5。

#### 四、 Compile(编译) 菜单

按 Alt+C 可进入 Compile 菜单, 该菜单有以下几个内容:

- Compile to OBJ(编译生成目标码)

将一个 .C 源文件编译生成 .OBJ 目标文件, 同时显示生成的文件名。其热键为 Alt+F9。

- Make EXE file(生成执行文件)

此命令生成一个 .EXE 的文件, 并显示生成的 .EXE 文件名。其中 .EXE 文件名是下面几项之一。

1. 由 Project / Project name 说明的项目文件名。
2. 若没有项目文件名, 则由 Primary C file 说明的源文件。
3. 若以上两项都没有文件名, 则为当前窗口的文件名。

- Link EXE file (连接生成执行文件)

把当前的 .OBJ 文件及库文件连接在一起生成 .EXE 文件。

- Build all(建立所有文件)

重新编译项目里的所有文件并进行装配, 生成 .EXE 文件。该命令不作过时检查(上面的几条命令要作过时检查, 即如果当前项目里源文件的日期和时间与目标文件相同或更早, 则拒绝对源文件进行编译)。

- Primary C file (主 C 文件)

当在该项中指定了主文件名后, 在以后的编译中, 如没有项目文件名则编译此项中规定的主 C 文件, 如果编译中有错误, 则将此文件调入编辑窗口, 不管当前窗口中是不是主 C 文件。



- Get info (获得信息)

该命令可获得有关当前路径、源文件名、源文件字节数、编译中的错误数目、可用空间等信息。

#### 五、Project (项目) 菜单

按 Alt+P 可进入 Project 菜单, 该菜单包括以下内容:

- Project name (项目名)

项目名具有 .PRJ 的扩展名, 其中包含将要编译、连接的文件名。例如有一个程序由 file1.c、file2.c、file3.c 组成, 要将这 3 个文件编译装配成一个 file.exe 的执行文件, 可以先建立一个 file.prj 的项目文件, 其内容如下:

```
file1.c
file2.c
file3.c
```

此时将 file.prj 放入 Project name 项中, 以后进行编译时将自动对项目文件中规定的三个源文件分别进行编译, 然后连接成 file.exe 文件。

如果其中有些文件已经编译成 .OBJ 文件, 并且又没有修改过, 可直接写上 .OBJ 扩展名。此时将不再编译而只进行连接。

例如:

```
file1.obj
file2.c
file3.c
```

将不对 file1.c 进行编译, 而直接连接。

注:

- 当项目文件中的每个文件无扩展名时, 均按源文件对待。另外, 其中的文件也可以是库文件, 但必须写上扩展名 .lib。

- Break make on(中止编译)

由用户选择是否有 Warning(警告)、Errors(错误)、Fatal Errors(致命错误)时或 Link(连接)之前退出 Make 编译。

- Auto dependencies(自动依赖)

当开关置为 on, 编译时将检查源文件与对应的 .OBJ 文件的日期和时间, 否则不进行检查。

- Clear project(清除项目文件)

清除 Project / Project name 中的项目文件名。

- Remove messages(删除信息)

把错误信息从信息窗口中清除掉。

#### 六、Options(选择)菜单

按 Alt+O 可进入 Options 菜单, 该菜单对初学者来说要谨慎使用。

- Compiler(编译器)

本项选择又有许多子菜单, 可以让用户选择硬件配置、存储模型、调试技术、代码优

化、对话信息控制和宏定义。这些子菜单如下:

#### Model

共有 tiny,small,medium,compact,large,huge,model 七种不同内存模式可由用户选择。

#### Define

打开一个宏定义框,用户可输入宏定义。多重定义可用分号,赋值可用等号。

#### Code generation

它又有许多任选项,这些任选项告诉编译器产生什么样的目标代码。

Calling convention      可选择C或Pascal方式传递参数。

Instruction set      可选择8088/8086或80186/80286指令系列。

Floating point      可选择仿真浮点、数学协处理器浮点或无浮点运算。

Default char type      规定char的类型。

Alignment      规定地址对准原则。

Merge duplicate strings      作优化用,将重复的字符串合并在一起。

Standard stack frame      是否产生一个标准的栈结构。

Test stack overflow      是否产生一段程序运行时堆栈溢出的代码。

Line member      是否在.OBJ文件中放进行号以供调试时用。

OBJ debug information      是否在.OBJ文件中产生调试信息。

#### Optimization

Optimize for      选择对程序小型化还是对程序速度进行优化处理。

Use register variable      用来选择是否允许使用寄存器变量。

Register optimization      尽可能地使用寄存器变量以减少过多的取数操作。

Jump optimization      通过去除多余的跳转和调整循环与开关语句的办法来压缩代码。

#### Source

Identifier length      说明标识符有效字符的个数,默认为32个。

Nested comments      是否允许嵌套注释。

ANSI keywords only      是只允许ANSI关键字还是也允许Turbo C关键字。

#### Error

Error stop after      多少次出错时停止编译,默认为25个。

Warning stop after      多少个错误警告时停止编译,默认为100个。

Display warning      此开关为on时将显示下列错误警告:

Portability warning      移植性错误警告。

ANSI Violations      侵犯了ANSI关键字的错误警告。

Common error      普通的错误警告。

Less common error      较少见的错误警告。

#### Names

用于改变段(segment)、组(group)和类(class)的名字,默认值为 CODE、DATA、BSS。

#### • Linker(连接器)

本菜单设置有关连接的选择项，它有以下内容：

Map file menu	选择是否产生.MAP文件。
Initialize segments	是否在连接时初始化尚未初始化的段。
Default libraries	是否在连接其它编译程序产生的目标文件时去寻找其缺省库。
Graphics library	是否连接graphics库中的函数。
Warn duplicate symbols	当有重复符号时是否产生警告信息。
Stack warning	是否让连接程序产生No stack的警告信息。
Case-sensitive link	是否区分大、小写字母。

• Environment(环境)

本菜单规定是否对某些文件自动存盘及制表键和屏幕大小的设置。

Message tracking

Current file	跟踪在编辑窗口中的文件错误。
All files	跟踪所有的文件错误。
Off	不跟踪。

Keep message 编译前是否清除Message窗口中的信息。

Config auto save 选on时，在Run、Shell或退出集成开发环境之前，如果Turbo C 的配置被修改过，则所做的改动将存入配置文件中。选 off 时则不存。

Edit auto save 是否在Run或Shell之前，自动存储编辑的源文件。

Backup files 是否在源文件存盘时产生后备文件(.BAK文件)。

Tab size 设置制表键的宽度，默认为8。

Zoomed windows 将现行活动窗口放大到整个屏幕，其热键为F5。

Screen size 设置屏幕文本大小。

• Directories(路径)

规定编译、连接所需文件的路径。有下列各项：

Include directories	包含文件的路径，多个子目录用";"分开。
Library directories	库文件路径，多个子目录用";"分开。
Output directories	输出文件(.OBJ、.EXE、.MAP文件)的目录。
Turbo C directories	Turbo C所在的目录。
Pick file name	定义加载的pick文件名，如不定义则从current pick file 中取。

• Arguments(命令行参数)

允许用户使用命令行参数。

• Save options(存储配置)

保存所有选择的编译、连接、调试和项目任选到配置文件中，缺省的配置文件为TCCONFIG.TC。

• Rctrive options

装入一个配置文件到 TC 中，TC 将使用该文件的选择项。

## 七、 Debug(调试)菜单

按 Alt+D 可选择 Debug 菜单, 该菜单主要用于查错, 它包括以下内容:

### Evaluate

Expression	要计算结果的表达式。
Result	显示表达式的计算结果。
New value	赋给新值。

Call stack      该项不可接触。而在 Turbo C debugger 时用于检查堆栈的情况。

Find function    在运行 Turbo C debugger 时用于显示规定的函数。

Refresh display    如果编辑窗口偶然被用户窗口重写了, 可用它来恢复编辑窗口的内容。

## 八、 Break/watch(断点及监视表达式)

按 Alt+B 可进入 Break/watch 菜单, 该菜单有以下内容:

Add watch	向监视窗口插入一监视表达式。
Delete watch	在监视窗口中删除当前的监视表达式。
Edit watch	在监视窗口中编辑一个监视表达式。
Remove all watches	在监视窗口中删除所有的监视表达式。
Toggle breakpoint	对光标所在的行设置或清除断点。
Clear all breakpoints	清除所有的断点。
View next breakpoint	将光标移动到下一个断点处。

### 1.5.2 Turbo C 的配置文件

所谓配置文件是包含 Turbo C 有关信息的文件, 其中存有编译、连接的选择和路径等信息。

可以用下述方法建立 Turbo C 的配置:

#### 1. 建立用户自命名的配置文件

可以从 Options 菜单中选择 Options/Save options 命令, 将当前集成开发环境的所有配置存入一个由用户命名的配置文件中。下次启动 TC 时只要在 DOS 下键入:

tc/c<用户命名的配置文件名>

就会按这个配置文件中的内容作为 Turbo C 的选择。

2. 若设置 Options/Environment/Config auto save 为 on, 则在退出集成开发环境时, 当前的设置会自动存放到 Turbo C 配置文件 TCCONFIG.TC 中。Turbo C 在启动时会自动寻找这个配置文件。

3. 用 TCINST 设置 Turbo C 的有关配置, 并将结果存入 TC.EXE 中。Turbo C 在启动时, 若没有找到配置文件, 则取 TC.EXE 中的缺省值。

## 第二章 数据类型、变量和运算符

本章首先介绍 Turbo C 程序的基本组成部分，然后主要介绍 Turbo C 的数据类型、变量类型、变量的初始化和赋值，最后介绍 Turbo C 的有关操作。通过本章的学习，读者可以对 Turbo C 语言有一个初步的认识。

### 2.1 Turbo C 程序的一般组成部分

Turbo C 象其它语言一样按其规定的格式和提供的语句由用户编写应用程序。请看下面一段 Turbo C 源程序。

#### 例 2-1:

```
/* Example program of Turbo C */
#include <stdio.h>          /* 包含文件说明 */
void sub(void);             /* 用户函数说明 */
char test;                  /* 定义全程变量 */
main( )                     /* 主函数定义 */
{
    char c;                 /* 定义局部变量 */
    clrscr( );
    gotoxy(22,8);
    puts("Welcome to use this book!");
    gotoxy(25,13);
    printf("<CR> ---- Continue");
    gotoxy(25,15);
    printf("<Esc> --- Exit");
    while(1)
    {
        c = getch( );
        if(c == 27)
            break;
        if(c == 13)
        {
            sub( );
            if(test == 'y' || test == 'Y')
            {
                gotoxy(23,14);
                puts("Please contact with me");
                getch( );
            }
        }
    }
}
```

```

        break;
    }
}
exit(0);
}
void sub( )                /* 用户定义的子函数 */
{
    clrscr( );
    gotoxy(22,8);
    printf("The best choice for you.");
    gotoxy(21,12);
    printf("Do you have any question?(Y/N)");
    test = getch( );
}

```

由例子程序可以看出，Turbo C 源程序主要有以下几个特点：

1. 程序一般都用小写字母书写；
2. 大多数语句结尾必须用“;”作为终止符，否则Turbo C不认为该语句已结束；
3. 每个程序必须有一个而且只能有一个称作主函数的main( )函数；
4. 每个程序体(主函数和每个子函数，如上例中的main( )函数和sub( )函数)必须用一对花括号“{”和“}”括起来；
5. 一个较完整的程序大致包括：包含文件(一组#include <\*.h> 语句)、用户函数说明部分、全程变量定义、主函数和若干子函数组成。在主函数和子函数中又包括局部变量定义、若干个 Turbo C 库函数、控制流程语句、用户函数的调用语句等；
6. 注释部分包含在“/\*”和“\*/”之间，在编译时它被Turbo C编译器忽略。

注：

- 象其它一些语言一样，Turbo C 的变量在使用之前必须先定义其数据类型，未经定义的变量不能使用。定义变量类型应在可执行语句前面，如上例的 main( )函数中的第一条语句就是变量定义语句，它必须放在第一条执行语句 clrscr( )前面。
- 在Turbo C中，大、小写字母是有区别的，同一字母的大、小写分别表示不同的变量；
- Turbo C程序的书写格式非常灵活，没有严格限制。

例 2-1 中的主函数可写成：

```

main( ) { char c; clrscr( ); gotoxy(22,8);
        puts("Welcome to use this book!"); gotoxy(25,13);
        printf("<CR> —— Continue") ;gotoxy(25,15);...}

```

这样写在语法上并没有错误，但阅读起来很不方便，同时也使得程序的层次不明晰。

作者建议在用 Turbo C 编程时,一行写一条语句,遇到嵌套语句并向后缩格,必要时再给程序加上注释行,这样可以使程序结构清楚、易于阅读、维护和修改。

通过以上介绍,可以得出 Turbo C 源程序的一般形式为:

```
包含文件
用户函数类型说明
全程变量定义
main( )
{
    局部变量定义
    <程序体>
}
func1( )
{
    局部变量定义
    <程序体>
}
func2( )
{
    局部变量定义
    <程序体>
}
.
.
.
funcN( )
{
    局部变量定义
    <程序体>
}
```

其中 func1( ),...,funcN( )代表用户定义的函数,程序体指 Turbo C 2.0 提供的任何库函数调用语句、控制流程语句或其它用户函数调用语句等。

## 2.2 数据类型

在 Turbo C 语言中,每个变量在使用之前必须定义其数据类型。Turbo C 有以下几种数据类型:整型(int)、浮点型(float)、字符型(char)、指针型(\*),无值型(void)以及结构(struct)和联合(union)。其中前五种是 Turbo C 的基本数据类型,后两种数据类型(结构和联合)将在第五章介绍。

### 2.2.1 整型(int)

#### 一、整型数说明

加上不同的修饰符, 整型数有以下几种类型:

- signed short int 有符号短整型数说明。简写为short或int, 字长为2字节共16位二进制数, 数的范围是-32768~32767。
- signed long int 有符号长整型数说明。简写为long, 字长为4字节共32位二进制数, 数的范围是-2147483648~2147483647。
- unsigned short int 无符号短整型数说明。简写为unsigned int, 字长为2字节共16位二进制数, 数的范围是0~65535。
- unsigned long int 无符号长整型数说明。简写为unsigned long, 字长为4字节共32位二进制数, 数的范围是0~4294967295。

#### 二、整型变量定义

可以用下列语句定义整型变量:

```
int a, b;          /* a、b被定义为有符号短整型变量 */
unsigned long c;    /* c被定义为无符号长整型变量 */
```

#### 三、整型常数表示

按不同的进位制区分, 整型常数有三种表示方法:

十进制数: 以非0开始的数

如: 230, -100, 32960

八进制数: 以0开始的数

如: 06, 0106, 05307

十六进制数: 以0X或0x开始的数

如: 0X0D, 0XFF, 0x4c

另外, 可在整型常数后添加一个"L"或"l"字母表示该数为长整型数, 如10L, 0532L, 0X8EA4l。

### 2.2.2 浮点型(float)

#### 一、浮点数说明

Turbo C 中有以下两种类型的浮点数:

- float 单浮点数。字长为4个字节共32位二进制数, 数的范围是 $3.4 \times 10^{-38} \sim 3.4 \times 10^{+38}$ 。
- double 双浮点数。字长为8个字节共64位二进制数, 数的范围是 $1.7 \times 10^{-308} \sim 1.7 \times 10^{+308}$ 。

注:

- 浮点数均为有符号的, 没有无符号的浮点数。

#### 二、浮点型变量定义

可以用下列语句定义浮点型变量:

```
float a, f; /* a, f 被定义为单浮点型变量 */
```



double b; /\* b 被定义为双浮点型变量 \*/

### 三、浮点常数表示

例如: +87.18, -4.75, -3.5e-10, 3.718

注:

- 浮点常数只有一种进位制(十进制)。
- 所有的浮点常数都被默认为double。
- 绝对值小于1的浮点数, 其小数点前面的零可以省略。如0.95可写为.95, 又如-0.0086e-5可写为-.0086e-5。
- Turbo C以默认格式输出浮点数时, 最多只保留小数点后六位。

## 2.2.3 字符型(char)

### 一、字符型变量定义

加上不同的修饰符, 可以定义有符号和无符号两种类型的字符型变量, 例如:

char c; /\* c被定义为有符号字符变量 \*/

unsigned char h; /\* h被定义为无符号字符变量 \*/

字符在计算机中以其 ASCII 码方式表示, 其长度为 1 个字节, 有符号字符型数取值范围是-128~127, 无符号字符型数取值范围是 0~255。因此在 Turbo C 语言中, 字符型数据在操作时将按整型数处理, 如果某个变量定义成 char, 则表明该变量是有符号的, 即它将转换成有符号的整型数。

Turbo C 中规定 ASCII 码值大于 0x80 的字符将被认为是负数。例如 ASCII 值为 0x8c 的字符, 定义成 char 时, 被转换成十六进制的整数 0xff8c。这是因为当 ASCII 码值大于 0x80 时, 该字节的最高位为 1, 计算机会认为该数为负数, 对于 0x8c 表示的数实际上是-74(8c 的各位取反再加 1), 而-74 转换成两字节整型数并在计算机中表示时就是 0xff8c(对 0074 各位取反再加 1)。因此只有定义为 unsigned char, 0x8c 转换成整型数时才是 8c, 这一点在处理大于 0x80 的 ASCII 码字符时(例如汉字码)要特别注意。汉字一般均定义为 unsigned char(在以后的程序中会经常碰到)。

另外, 也可以定义一个字符型数组(关于数组后面再作详细介绍), 此时该数组表示一个字符串。

例如:

```
char str[10];
```

计算机在编译时, 将留出连续 10 个字符的空间, 即 str[0]到 str[9]共 10 个变量, 但只有前 9 个供用户使用。第 10 个即 str[9]用来存放字符串终止符 NULL 即"\0", 但终止符是编译程序自动加上的, 这一点应特别注意。

### 二、字符常数表示

能用符号表示的字符可直接用单引号括起来表示, 如'a','9','Z', 也可用该字符的 ASCII 码值表示, 例如十进制数 85 表示大写字母"U", 十六进制数 0x5d 表示"J", 八进制数 0102 表示大写字母"B"。

一些不能用符号表示的控制符, 只能用 ASCII 码值来表示, 如十进制数 10 表示换行, 十六进制数 0x0d 表示回车, 八进制数 033 表示 Esc。Turbo C 中也有另外一种表示

方法, 如'\033'表示 Esc, 这里"\0"符号后面的数字表示八进制的 ASCII 码值; '\X0D'或'\x0d'表示回车, 这里"\X"或"\x"后面的数字表示十六进制的 ASCII 值。当然这种表示方法也适用于可直接用符号表示的字符。

另外, Turbo C 中有些常用的字符用下列特殊规定来表示:

规定符	等价于	含 义
'\f'	'\X0C'	换页
'\r'	'\X0D'	回车
'\t'	'\X09'	制表键
'\n'	'\X0A'	换行
'\\'	'\X5C'	\ 符
'\"'	'\X27'	' 符
'\"'	'\X22'	" 符

对于字符串常量, 一般用双引号括起来表示, 如"Hello, Mr. Wang."。

#### 2.2.4 指针型(\*)

指针是一种特殊的数据类型, 在其它语言中一般都没有。指针是指向变量的地址, 实质上指针就是存储单元的地址。根据所指的变量类型不同, 可以是整型指针(int \*), 浮点型指针(float \*), 字符型指针(char \*), 结构指针(struct \*)和联合指针(union \*) (结构指针和联合指针将在第五章介绍)。

指针型变量的定义如下:

```
char *s;      /* s被定义为字符型指针 */
int *a;       /* a被定义为整数型指针 */
float *f;     /* f被定义为单浮点型指针 */
```

要注意的是指针没有常量, 即在 Turbo C 中没有"\* 常量"这种表示法。

#### 2.2.5 无值型(void)

无值型的字节长度为 0, 主要有两个用途: 一是明确地表示一个函数不返回任何值; 一是产生一个同一类型的指针(可根据需要动态地给其分配内存)。

例如:

```
void *buffer; /* buffer 被定义为无值型指针 */
```

### 2.3 关键字和标识符

#### 2.3.1 关键字

所谓关键字就是已被 Turbo C 本身使用, 因此不能作其它用途使用的字。例如关键字不能用作变量名、函数名等。

Turbo C 2.0 有以下关键字:

属 Turbo C 2.0 扩展的共 11 个:

asm	__cs	__ds	__es	__ss	cdecl
far	near	huge	interrupt	pascal	

由 ANSI 标准定义的共 32 个:

auto	double	int	struct	break	else
long	switch	case	enum	register	typedef
char	extern	return	union	const	float
short	unsigned	continue	for	signed	void
default	goto	sizeof	volatile	do	if
while	static				

### 2.3.2 标识符

所谓标识符是指常量、变量、语句标号以及用户自定义函数的名称。Turbo C 标识符的定义十分灵活。作为标识符必须满足以下规则:

1. 所有标识符必须由一个字母(a~z、A~Z)或下划线(\_)开头;
2. 标识符的其它部分可以用字母、下划线或数字(0~9)组成;
3. 大小写字母表示不同意义, 构成不同的标识符;
4. 标识符只有前 32 个字符有效;
5. 标识符不能使用 Turbo C 的关键字。

下面举出几个正确和不正确的标识符:

正 确	不正确
smart	5smart
__decision	bomb?
key__board	key.board
FLOAT	float

## 2.4 变量

### 2.4.1 变量说明

Turbo C 规定所有的变量在使用前都必须加以说明。一条变量说明语句由数据类型和其后的一个或多个变量名组成。变量说明的格式如下:

类型 <变量表>;

这里的类型是指 Turbo C 的有效数据类型, 变量表是一个或多个标识符名, 每个标识符之间用“,”分隔。

例如:

```
int i, j, k;    unsigned char c, str[5], *p;
```

### 2.4.2 变量种类

变量可以放在程序中的三个地方说明: 函数内部、函数的参数定义中或所有的函数外部, 根据所定义位置的不同, 变量可分为局部变量、形式参数和全程变量。

#### 一、局部变量

局部变量是指在函数内部说明的变量(有时也称为自动变量)。局部变量可以用关键字 auto 进行说明, 当 auto 省略时, 所有的非全程变量都被认为是局部变量, 因此 auto 实

上是从来不用。

局部变量在函数调用时自动产生，但并不会自动初始化，随着函数调用的结束，这个变量也就自动消失了，下次调用此函数时再自动产生，还要再赋值，退出时又自动消失。

## 二、形式参数

形式参数是指在函数名后面的圆括号里定义的变量，用于接受来自调用函数的参数。形式参数在函数内部可以象其它局部变量那样来使用。

例如：

```
puthz(int x, int y, int color, char *p)
{
    int i, j, k;    /* 定义局部变量 */
    <程序体>
}
```

其中 x,y,color,\*p 都是函数的形式参数，不需要再进行说明就可在该函数内直接使用。

## 三、全程变量

全程变量是指在所有函数之外说明的变量，它在整个程序内部都是“可见的”，可以被任何一个函数使用，并且在整个程序的运行中都保留其值。全程变量只要符合使用前和函数外这样两个条件，就可以在程序中的任何位置进行说明，习惯上通常放在程序的主函数前说明。

例如：

```
#include <stdio.h>
int test;    /* test 定义为全程变量 */
main( )
{
    test = 5;    /* 给全程变量赋值 */
    f1(20, 5.5); /* 调用有形式参数的用户函数 f1( ), 调用
                  之后 test 的值变成 115 */
    f2( );    /* 调用用户函数 f2( ), 在调用之后 test 的值变成 1150 */
}
f1(int x, float y) /* 用户定义函数 */
{
    float z;    /* z 定义为局部变量 */
    z = x * y;    /* 计算 */
    test = test + z;
}
f2( )
{
    int count = 10; /* 定义局部变量并初始化 */
    test = test * count;
```

}

由于全程变量可以被整个程序内的任何一个函数使用，所以可以用作函数之间传递参数的工具，但全程变量太多时，会增加内存开销。

### 2.4.3 变量存储类型

Turbo C 支持四种变量存储类型。说明符如下：

auto          static          extern          register

下面分别来介绍。

#### 一、auto

auto 称为自动变量，已在前面作了介绍，这里不再重复。

#### 二、static

static 称为静态变量。根据变量的类型可以分为静态局部变量和静态全程变量。

##### 1. 静态局部变量

它与局部变量的区别在于：在函数退出时，这个变量始终存在，但不能被其它函数使用，在再次进入该函数时，将保存上次的结果。其它与局部变量一样。

##### 2. 静态全程变量

Turbo C 允许将大型程序分成若干独立模块文件分别编译，然后将所有模块的目标文件连接在一起，从而提高编译速度，同时也便于软件的管理和维护。静态全程变量就是指只在定义它的源文件中可见而在其它源文件中不可见的变量。它与全程变量的区别是：全程变量可以再说明为外部变量(extern)，被其它源文件使用，而静态全程变量却不能再被说明为外部的，即只能被所在的源文件使用。

#### 三、extern

extern 称为外部变量。为了使变量除了在定义它的源文件中可以使用外，还能够供其它文件使用，因此就必须将全程变量通知每一个程序模块文件，此时可用 extern 来说明。

例如：

文件1为file1.c

```
int i, j; /* 定义全程变量 */
char c;
main( )
{
    func1(20); /* 调用函数 */
    func2( );
    .
    .
    .
}
func1(int k) /* 用户定义函数 */
{
```

文件2为file2.c

```
extern int i, j; /* 说明将i,j从
                  文件1中复制过来 */
extern char c; /* 将c也复制过来 */
func2( ) /* 用户定义函数 */
{
    static float k; /* 定义静态变量 */
    i=j*5/100;
    k=i/1.5;
    .
    .
    .
}
```

```
j=k*100;
}
```

对于以上两个文件 file1.c 和 file2.c, 在利用 Turbo C 的集成开发环境进行编译连接时, 首先应建立一个 .prj 的文件。例如 file.prj, 该文件的内容如下:

```
file1.c
file2.c
```

然后将 file.prj 的文件名写入主菜单 Project 中的 Project Name 项中, 再用 F9 编译连接, 就可产生一个文件名为 file.exe 的可执行文件。

外部变量和 FORTRAN 语言中的 COMMON 定义的公共变量一样。

#### 四、register

register 称为寄存器变量, 它只适用于整型和字符型变量。定义符 register 所说明的变量被 Turbo C 存储在 CPU 的寄存器中, 而不是象普通的变量那样存储在内存中, 这样可以提高运算速度。但是, Turbo C 只允许同时定义两个寄存器变量, 一旦超过两个, 编译程序会自动地将超过限制数目的寄存器变量当作非寄存器变量来处理。因此, 寄存器变量常用在同一变量名出现频繁的场所。

另外, 寄存器变量只适用于局部变量和函数的形式参数, 属于 auto 型变量, 因此不能用作全程变量。定义一个整型寄存器变量可写成:

```
register int a;
```

对于以上所介绍的变量类型和变量存储类型将会在以后的学习中, 通过例行程序中的定义、使用来逐渐加深理解。

#### 2.4.4 数组变量

所谓数组就是指一些具有相同数据类型的变量的集合, 拥有一个共同的名字。数组中的每个特定元素都使用下标来访问。数组由一段连续的存储地址构成, 最低的地址对应于第一个数组元素, 最高的地址对应最后一个数组元素。数组可以是一维的、也可以是多维的。Turbo C 象其它高级语言一样也使用数组变量。

##### 一、一维数组

一维数组的说明格式是:

```
类型 变量名[长度];
```

类型是指数据类型, 即每一个数组元素的数据类型, 包括整型、浮点型、字符型、指针型以及结构和联合。

例如:

```
int a[10];           /* 定义一个含有10个元素的整型数组 */
unsigned long l[20]; /* 定义一个含有20个元素的长整型数组 */
char * s[5];         /* 定义一个含有5个元素的字符型指针数组 */
char * f[];          /* 定义一个下标数不确定的字符型指针数组 */
```

注:

- 数组都是以0作为第一个元素的下标。因此, 在说明一个 int A[6] 的整型数组时,

表明该数组具有 6 个元素 A[0]~A[5]，每个元素都是一个整型变量。

- 大多数字符串用一维数组表示。数组元素的多少表示字符串长度，数组名表示字符串中第一个字符的地址，例如在语句 `char str[6]` 说明的数组中存入 "Hello" 字符串后，`str` 表示第一个字母 "H" 所在的内存单元地址，`str[0]` 存放的是字母 "H" 的 ASCII 值，依次类推，`str[4]` 存放的是字母 "o" 的 ASCII 值，而 `str[5]` 则应存放字符串终止符 '\0'。

- Turbo C 对数组不作边界检查。例如用下面语句说明两个数组

```
char str1[4], str2[3];
```

当赋给 `str1` 一个字符串 "ABCDE" 时，只有 "ABCD" 被赋给，"E" 将会自动地赋给 `str2`，这点应特别注意。

## 二、多维数组

多维数组的一般说明格式是：

类型 数组名[第 n 维长度][第 n-1 维长度].....[第 1 维长度];

这种说明方式与 BASIC、FORTRAN 等语言中多维数组的说明不一样。

例如：

```
int m[3][2];      /* 定义一个整数型的二维数组 */
```

```
char c[2][2][3];  /* 定义一个字符型的三维数组 */
```

数组 `m[3][2]` 共有  $3 * 2 = 6$  个元素，顺序为：

`m[0][0]`, `m[0][1]`, `m[1][0]`, `m[1][1]`, `m[2][0]`, `m[2][1]`;

数组 `c[2][2][3]` 共有  $2 * 2 * 3 = 12$  个元素，顺序为：

`c[0][0][0]`, `c[0][0][1]`, `c[0][0][2]`, `c[0][1][0]`,

`c[0][1][1]`, `c[0][1][2]`, `c[1][0][0]`, `c[1][0][1]`,

`c[1][0][2]`, `c[1][1][0]`, `c[1][1][1]`, `c[1][1][2]`

数组占用的内存空间(即字节数)的计算公式为：

字节数 = 第 1 维长度 \* 第 2 维长度 \* ... \* 第 n 维长度 \* 该数组数据类型占用的字节数。

## 2.4.5 变量的初始化和赋值

### 一、变量的初始化

变量的初始化是指变量在被说明的同时赋给一个初值。Turbo C 的外部变量和静态全程变量在程序开始处被初始化，局部变量包括静态局部变量是在进入定义它们的函数或复合语句时才作初始化。所有的全程变量在没有明确的初始化时将被自动清零，而局部变量和寄存器变量在未赋值前其值是不确定的。

对于外部变量和静态变量，初值必须是常数表达式，而对于自动变量和寄存器变量则可以是任意的表达式，这个表达式可以包含常数和前面说明过的变量和函数。

#### 1. 单个变量的初始化

例如：

```
float f0, f1 = 0.2; /* 定义全程变量，在初始化时 f0 被清零，f1 被赋 0.2 */
```

```

main( )
{
    static int i=10, j; /* 定义静态局部变量, 初始化时 i 被赋 10, j 不确定 */
    int k=i*5;          /* 定义局部变量, 初始化时 k 被赋 10*5=50 */
    char c='y';         /* 定义局部变量, 初始化时 c 被赋字母 y */
    .
    .
    .
}

```

## 2. 指针型变量的初始化

例如:

```

main( )
{
    int *i=1234;        /* 定义整型数指针变量并初始化 */
    float *f=3.14159;   /* 定义浮点数指针变量并初始化 */
    char *s="Hello";    /* 定义字符型指针变量并初始化 */
    .
    .
    .
}

```

## 3. 数组变量的初始化

例如:

```

main( )
{
    int p[2][3]={{2,-9,0}, {8,2,-5}}; /* 定义数组 p 并初始化 */
    int m[2][4]={{27,-5,19,3}, {1,8,-14,-2}}; /* 定义数组 m 并初始化 */
    char *f={'A','B','C'}; /* 定义数组 f 并初始化 */
    .
    .
    .
}

```

从上例可以看出, Turbo C 中数组进行初始化有下述规则:

- (1) 数组的每一行初始化赋值用“{”并用“,”分开, 总的再加一对“{”括起来, 最后以“;”表示结束。
- (2) 多维数组是按最右维的下标最先变化的原则分配存储空间的。
- (3) 多维数组的存储是连续的, 因此可以用一维数组初始化的办法来初始化多维数组。

例如:

```

int x[2][3]={1,2,3,4,5,6}; /* 用一维数组来初始化二维数组 */

```



(4) 对数组初始化时, 如果初值表中的数据个数比数组元素少, 则不足部分的数组元素用 0 来填补。

(5) 对指针型变量数组可以不规定维数, 在初始化赋值时, 数组按下标从 0 开始被连续赋值。

例如:

```
char * f[] = {'A', 'B', 'C'};
```

初始化时将会给 3 个字符指针赋值, 即: \*f[0]='A', \*f[1]='B', \*f[2]='C'。

## 二、变量的赋值

变量赋值是给已说明的变量赋给一个特定值。

### 1. 单个变量的赋值

#### (1) 整型变量和浮点变量

这两种变量采用下列格式赋值:

变量名 = 表达式;

例如:

```
main( )
{
    int i, j;    /* 定义局部整型变量 i, j */
    float f;     /* 定义局部浮点型变量 f */
    i = 100; j = 20; /* 给变量赋值 */
    f = i * j * 0.1;
    .
    .
}
```

注:

● Turbo C 在给多个变量赋同一值时允许使用连等的形式。

例如:

```
main( )
{
    int i, j, k;
    i = j = k = 0; /* 同时给 i, j, k 赋 0 */
    .
    .
    .
}
```

#### (2) 字符型变量

字符型变量可以用三种方法赋值。

例如:

```
main( )
```

```

{
    char c0, c1, c2; /* 定义局部字符型变量 c0、c1、c2 */
    c0='B';          /* 将字母 B 赋给 c0 */
    c1=50;            /* 将数字 2(ASCII 值为十进制 50)赋给 c1 */
    c2='\X0D';        /* 将回车符赋给 c2 */
    .
    .
    .
}

```

### (3) 指针型变量

例如:

```

main( )
{
    *int i;
    char *str;
    *i=100;
    str="Hello, Mr. Wang.";
    .
    .
    .
}

```

\*i 表示 i 是一个指向整型数的指针, 即 \*i 是一个整型变量, i 是一个指向该整型变量的地址。

\*str 表示 str 是一个字符型指针, 即保留某个字符的地址。在初始化时, str 没有什么特殊的值, 而在执行 str="Hello, Mr. Wang."时, 编译器先在目标文件的某处保留一个空间存放"Hello, Mr. Wang.\0"的字符串, 然后把这个字符串的第一个字母"H"的地址赋给 str, 其中字符串结尾符"\0"是编译程序自动加上的。

对于指针变量的使用要特别注意。上例中两个指针在说明时都没有初始化, 因此这两个指针为随机地址, 在小存储模式下使用将会有破坏机器的危险。正确的使用办法如下:

例如:

```

main( )
{
    int *i;
    char *str;
    i=(int *)malloc(sizeof(int));
    str=(char *)malloc(20);    *i=420;
    str="Hello, Mr. Wang.";
    .
    .
    .
}

```

```
}
```

上例中, 函数(int \*)malloc(sizeof(int))表示分配连续的 sizeof(int)=2 个字节的整型数存储空间并返回其首地址。同样, (char \*)malloc(20)表示分配连续 20 个字节的字符存储空间并返回首地址(有关该函数以后再详述)。由动态内存分配函数 malloc( )分配了内存空间后, 这部分内存将专供指针变量使用。

如果要使 i 指向三个整型数, 则用下述方法。

例如:

```
#include <alloc.h>
main( )
{
    int *i;
    i = (int *)malloc(3 * sizeof(int));
    *i = 2048;
    *(i+1) = 0;
    *(i+2) = 4096;
    .
    .
    .
}
```

\*i = 2048 表示把 2048 存放到 i 指向的地址中去。但对于 \*(i+1)=0, 如果认为将 0 存放到 i 指向的下一个字节中就错了。Turbo C 只要说明 i 为整型指针, 则

(i+1) 等价于 i+1 \* sizeof(int)

同样,

(i+2) 等价于 i+2 \* sizeof(int)

## 2. 数组变量的赋值

### (1) 整型数组和浮点数组的赋值

例如:

```
main( )
{
    int m[2][2];
    float n[3];
    m[0][0]=0, m[0][1]=17, m[1][0]=21; /* 数组元素赋值 */
    n[0] = 109.5, n[1]=-8.29, n[2]=0.7;
    .
    .
    .
}
```

### (2) 字符串数组的赋值

例如:

```

main( )
{
    char s[30];
    strcpy(s,"Good Bye! Mr. Wang.");    /* 给数组赋字符串 */
    .
    .
    .
}

```

上面的程序在编译中，遇到 `char s[30]` 这条语句时，编译程序会在内存的某处留出连续 30 个字节的区域，并将第一个字节的地址赋给 `s`。当遇到 `strcpy` (`strcpy` 为 Turbo C 的函数) 时，首先在目标文件的某处建立一个 "Good Bye! Mr. Wang.\0" 的字符串。其中 \0 表示字符串终止，终止符是编译时自动加上的，然后一个字符一个字符地复制到 `s` 所指向的内存区域。因此定义字符串数组时，其元素个数至少应该比字符串的长度多 1。

注：

- 字符串数组不能用 `"="` 直接赋值，即 `s="Good Bye! Mr. Wang."` 是不合法的。所以应分清字符串数组和字符串指针的不同赋值方法。
- 对于长字符串，Turbo C 允许使用下述方法：

例如：

```

main( )
{
    char s[100];
    strcpy(s,"The writer would like to thank you for"
           " your interest in his book. He hopes you"
           " can get some helps from the book.");
    .
    .
    .
}

```

### (3) 指针数组赋值

例如：

```

main( )
{
    char * f[2];
    int * a[2];
    f[0]="Thank you!";    /* 给字符型数组指针变量赋值 */
    f[1]="Mr. Wang.";
    *a[0]=1, *a[1]=-100;  /* 给整型数数组指针变量赋值 */
    .
    .
    .
}

```

}

### 三、数组与指针

数组与指针有密切的联系。数组名本身就是该数组的指针，反过来，也可以把指针看成一个数组，数组名和指针实质上都是地址。但是指针是变量，可以作运算；而数组名是常量，不能进行运算。

例如：

```
main( )
{
    char s[30, * p;    ] /* 定义字符型数组和指针变量 */
    p = s;              /* 指针p指向数组s的第一个元素s[0]的地址 */
    :
    * (p+8);            /* 指针p指向数组s的第9个元素s[8]的地址 */
    :
}
```

由上例可以看出数组和指针有如下关系：

$(p+i) = \&(s[i])$

$* (p+i) = s[i]$

因此，利用上述表达式可以对数组和指针进行互换。两者的区别仅在于：数组  $s$  是程序自动为它分配了所需的存储空间；而指针  $p$  则是利用动态分配函数为它分配存储空间或赋给它一个已分配的空间地址。

## 2.5 运算符

Turbo C 的运算符非常丰富，主要分为三大类：算术运算符、关系运算符与逻辑运算符、按位运算符。除此之外，还有一些用于完成特殊任务的运算符。下面分别进行介绍。

### 2.5.1 算术运算符

Turbo C 的算术运算符如下：

操作符	作用
+	加，一目取正
-	减，一目取负
*	乘
/	除
%	取模
--	减 1
++	加 1

### 一、一目和二目操作

一目操作是指对一个操作数进行的操作。例如： $-b$  是对  $b$  进行一目的取负操作。

二目操作(或多目操作)是指对两个操作数(或多个操作数)进行的操作。

在 Turbo C 中加、减、乘、除、取模的运算与其它高级语言相同。需要注意的是除法和取模运算。

例如:

$20 / 6$       是 20 除以 6 商的整数部分 3

$20 \% 6$       是 20 除以 6 的余数部分 2

取模运算符“ $\%$ ”不能用于浮点数。

另外, 由于 Turbo C 中字符型数会自动地转换成整型数, 因此字符型数也可以参加二目运算。

例如:

```
main( )
{
    char c, s;      /* 定义字符型变量 */
    c = 'b';      /* 给 c 赋值小写字母'b' */
    s = c + 'A' - 'a';      /* 将 c 中的小写字母变成大写字母'B'后赋给 s */
    .
    .
    .
}
```

本例中  $c = 'b'$  即  $c = 98$ , 由于字母 A 和 a 的 ASCII 码值分别为 65 和 97。这样可以将小写字母变成大写字母。反之, 如果要将大写字母变成小写字母, 则用  $c + 'a' - 'A'$  进行计算。

### 二、增减运算

在 Turbo C 中有两个很有用的运算符, 而在其它高级语言中通常是没有的。这两个运算符就是增 1 和减 1 的运算符“ $++$ ”和“ $--$ ”, 运算符“ $++$ ”是操作数加 1, 而“ $--$ ”则是操作数减 1。

例如:

$x = x + 1$       可写成  $x++$ , 或  $++x$

$x = x - 1$       可写成  $x--$ , 或  $--x$

$x++(x--)$  与  $++x(--x)$  在本例中没有什么区别, 但  $x = m++$  和  $x = ++m$  却有很大差别。

$x = m++$       表示将  $m$  的值赋给  $x$  后,  $m$  加 1。

$x = ++m$       表示  $m$  先加 1 后, 再将新值赋给  $x$ 。

### 三、赋值语句中的数据类型转换

类型转换是指不同类型的变量混用时的类型改变。

在赋值语句中, 类型转换规则是:

等号右边的值转换为等号左边变量所属的类型。

例如:

```
main( )
{
    int i, j;      /* 定义整型变量 */
    float f, g;    /* 定义浮点型变量 */
    f=i*j;         /* i与j的乘积是整型数, 被转换为浮点数赋给f */
    i=g;           /* g中的浮点型数转换为整型数赋给i */
    .
    .
}
```

由于 Turbo C 遵循上述数据类型转换规则, 因此在作除法运算时应特别注意。

例如:

```
main( )
{
    float f;
    int i;
    i=52;
    f=i/5;
}
```

上面程序经运行后得到  $f=10$ , 并不等于精确值 10.4。正确的程序应该是:

```
main( )
{
    float f;
    int i;
    i=52;
    f=i/5.0;
}
```

或直接将  $i$  定义为浮点数也可以。

## 2.5.2 逻辑运算符和关系运算符

### 一、逻辑运算符

逻辑运算符是用形式逻辑原则来建立数值间关系的符号。

Turbo C 的逻辑运算符如下:

操作符	作用
&&	逻辑与
	逻辑或
!	逻辑非

## 二、关系运算符

关系运算符是比较两个操作数大小的符号。

Turbo C 的关系运算符如下:

操作符	作用
>	大于
>=	大于等于
<	小于
<=	小于等于
==	等于
!=	不等于

关系运算符和逻辑运算符的关键是真(true)和假(false)的概念。Turbo C 的 true 可以是不为 0 的任何值,而 false 则为 0。使用关系运算符和逻辑运算符表达式时,若表达式为真(即 true)则返回 1;反之,表达式为假(即 false)则返回 0。

例如:

100 > 99	返回 1
10 > (1+10)	返回 0
!!&&0	返回 0

对本例中表达式!!&&0,先求!!或先求 1&&0 将会得出不同的结果,那么何者优先呢?这在 Turbo C 中是有规定的。有关运算符的优先级本章后面将会讲到。

### 2.5.3 按位运算符

Turbo C 和其它高级语言的不同是它能完全支持按位运算符。这与汇编语言的位操作有些相似。

Turbo C 的按位运算符有:

操作符	作用
&	位逻辑与
	位逻辑或
^	位逻辑异或
~	位逻辑反
>>	右移
<<	左移

按位运算是对于字节或字中的实际位进行检测、设置或移位,它只适用于字符型和整数型变量以及它们的变体,对其它数据类型不适用。

关系运算和逻辑运算表达式的结果只能是 1 或 0。而按位运算的结果可以取 0 或 1 以外的值。

要注意区别按位运算符和逻辑运算符的不同,例如,若  $x=7$ ,则  $x\&\&8$  的值为真(两个非零值相与仍为非零),而  $x\&8$  的值却为 0。



移位运算符">>"和"<<"是指将变量中的每一位向右或向左移动,其通常格式为:

右移: 变量名>>移位的位数

左移: 变量名<<移位的位数

经过移位后,一端的位被"挤掉",而另一端空出的位以0填补。所以,Turbo C 的移位不是循环移动的。

#### 2.5.4 Turbo C 的特殊运算符

##### 一、"?"运算符

"?"运算符是一个三目运算符,其一般格式是:

<表达式1> ? <表达式2> : <表达式3>;

"?"运算符的含义是:先求表达式1的值,如果为真,则求表达式2的值并把它作为整个表达式的值;如果表达式1的值为假,则求表达式3的值并把它作为整个表达式的值。

例如:

```
main( )
{
    int x,y;
    x=50;
    y=x>70?100:0;
}
```

本例中,y将被赋值0。如果x=80,y将被赋值100。

因此,"?"运算符可以代替某些if-then-else形式的语句。

##### 二、"&"和"\*"运算符

"&"运算符是一个返回操作数地址的单目操作符。

"\*"运算符是对"&"运算符的一个补充,它返回位于这个地址内的变量值,也是单目操作符。

例如:

```
main( )
{
    int i, j, *m;
    i=10;
    m=&i;          /* 将变量i的地址赋给m */
    j=*m;          /* 地址m所指的单元的值赋给j */
}
```

上面程序运行后,i=10,m为其对应的内存地址值,j的值也为10。

##### 三、","运算符

","运算符用于将多个表达式串在一起","运算符的左边总不返回,右边表达式的值才是整个表达式的值。

例如:

```
main( )
{
    int x, y;
    x = 50;
    y = (x = x - 5, x / 5);
}
```

上面的程序执行后 y 的值为 9，因为 x 的初始值为 50，减 5 后变为 45，45 除以 5 得 9 赋值给 y。

#### 四、sizeof 运算符

sizeof 运算符是一个单目运算符，它返回变量或类型的字节长度。

例如：

```
sizeof(double)为 8
sizeof(int)为 2
```

也可以求已定义的变量，例如：

```
float f;
int i;
i = sizeof(f);
```

则 i 的值将为 4。

#### 五、联合操作

Turbo C 中有一种特殊的简写方式，它用来简化一种赋值语句，适用于所有的双目运算符。对一般格式：

<变量> = <变量> <操作符> <表达式>

简写方式为

<变量> <操作符> = <表达式>

例如：

```
a = a + b    可写成    a += b
a = a & b    可写成    a &= b
a = a / (b - c) 可写成    a /= b - c
```

#### 2.5.5 Turbo C 运算符的优先级

Turbo C 规定了运算符的优先次序即优先级，当一个表达式中有多个运算符参加运算时，将按下页的表所规定的优先级进行运算。表中优先级从上往下逐渐降低，同一行里优先级相同。

例如：

```
表达式 10 > 4 && !(100 < 99) || 3 <= 5 的值为 1
表达式 10 > 4 && !(100 > 99) && 3 <= 5 的值为 0
```

## Turbo C 运算符的优先次序

表达式	优先级
( )(小括号) [ ](数组下标) .(结构成员) ->(指针型结构成员)	最高
!(逻辑非) ~(位取反) -(负号) ++(加 1) --(减 1) &(变量地址)	↑
*(指针所指内容) type(函数说明) sizeof(长度计算)	
*(乘) /(除) %(取模)	
+(加) -(减)	
<<(位左移) >>(位右移)	
<(小于) <=(小于等于) >(大于) >=(大于等于)	
==(等于) !=(不等于)	
&(位与)	
^(位异或)	
(位或)	
&&(逻辑与)	
(逻辑或)	
?:(?表达式)	
= += -=(联合操作)	
,(逗号运算符)	最低

## 第三章 输入输出函数

本章主要介绍 Turbo C 的标准输入输出函数和文件的输入输出函数。通过本章的学习可以使读者掌握 Turbo C 的屏幕输出、键盘输入以及磁盘文件的读写函数，并能开始进行一些简单程序的编写。

### 3.1 标准输入输出函数

#### 3.1.1 格式化输入输出函数

Turbo C 标准库提供了两种控制台格式化输入、输出函数 `scanf()` 和 `printf()`，这两个函数可以在标准输入输出设备上以各种不同的格式读写数据。`printf()` 函数用来向标准输出设备(屏幕)写数据；`scanf()` 函数用来从标准输入设备(键盘)上读数据。下面详细介绍这两个函数的用法。

##### 一、`printf()` 函数

`printf()` 函数是格式化输出函数，一般用于向标准输出设备按规定格式输出信息。在编写程序时经常会用到此函数。`printf()` 函数的调用格式为：

`printf("<格式化字符串>", <参量表>);`

其中格式化字符串包括两部分内容：一部分是常规字符，这些字符将按其原样输出；另一部分是格式化规定字符，以“%”开始，后跟一个或几个规定字符，用来确定输出内容的格式。

参量表是需要输出的一系列参数名，其个数必须与格式化字符串所说明的输出参数个数一样多，各参数名之间用“,”分开，且顺序一一对应，否则将会出现意想不到的错误。

##### 1. 格式化规定符

Turbo C 提供的格式化规定符如下：

符号	作用
%d	十进制有符号整数
%u	十进制无符号整数
%f	浮点数
%s	字符串
%c	单个字符
%p	指针的值
%e	指数形式的浮点数
%x 或 %X	无符号以十六进制表示的整数
%o	无符号以八进制表示的整数

注：

- 可以在“%”和字母之间插进数字表示最大场宽。

例如：

%3d    表示输出3位整型数，不够3位则右对齐。

%9.2f    表示输出场宽为9的浮点数，其中小数位数为2，整数位数为6，小数点占1

位, 不够 9 位则右对齐。

**%8s** 表示输出 8 个字符的字符串, 不够 8 个字符则右对齐。

如果字符串的长度, 或整型数的位数超过说明的场宽, 将按其实际长度输出。但对于浮点数, 若整数部分位数超过了说明的整数位宽度, 将按实际整数位输出; 若小数部分位数超过了说明的小数位宽度, 则按说明的宽度以四舍五入输出。

另外, 若想在输出值前加一些 0, 就应在场宽项前加个 0。

例如:

**%04d** 表示在输出一个不足 4 位的数值时, 将在前面填补 0 使其总宽度为 4 位。

如果用浮点数来表示字符或整型量的输出格式, 小数点后的数字代表最大宽度, 小数点前的数字代表最小宽度。

例如:

**%6.9s** 表示显示一个长度不小于 6 且不大于 9 的字符串。若大于 9, 则第 9 个字符以后的内容将被删除。

注:

- 可以在“%”和字母之间加小写字母 l, 表示输出的是长型数。

例如:

**%ld** 表示输出 long 型整数

**%lf** 表示输出 double 型浮点数

- 可以控制输出左对齐或右对齐, 即在“%”和字母之间加入一个“-”号表明要求输出左对齐, 否则为右对齐。

例如:

**%-7d** 表示输出 7 位整数左对齐

**%-10s** 表示输出 10 个字符左对齐

## 2. 一些特殊的规定符

字 符	作 用
\n	换行
\f	清屏并换页
\r	回车
\t	Tab (制表) 符
\xhh	表示一个 ASCII 码用 16 进制表示, 其中 hh 是 1 到 2 个 16 进制数。

由本节所学的 printf() 函数, 并结合前一章学习的数据类型, 编制下面的程序, 以加深对 Turbo C 数据类型的了解。

### 例 3-1:

```
#include <stdio.h>
#include <string.h>
main( )
{
    char c,s[20], * p;    /* 定义各种数据类型, 并对其中一些初始化 */
}
```

```

int a=1234,*i;
float f=3.141592653589;
double x=0.12345678987654321
p="Where are you going?";    /* 给一些变量赋值 */
strcpy(s,"Hello, Mr. wang.");
*i=12;
c='\x41';
printf("a =%d\n",a);          /* 输出十进制整数 */
printf("a =%6d\n",a);          /* 输出6位十进制整数 */
printf("a =%06d\n",a);          /* 输出6位十进制整数, 不够6位前补0 */
printf("a =%2d\n",a);          /* 超过2位按实际值输出 */
printf(" *i=%4d\n",*i);          /* 输出4位十进制整数 */
printf(" *i=%-4d\n",*i);          /* 输出左对齐4位十进制整数 */
printf("i =%p\n",i);           /* 输出地址 */
printf("f =%f\n",f);           /* 输出浮点数 */
printf("f =%6.4f\n",f);          /* 输出6位其中小数点后4位的浮点数 */
printf("x =%lf\n",x);           /* 输出长浮点数 */
printf("x =%18.16lf\n",x);          /* 输出18位其中小数点后16位的长浮点数 */
printf("c =%c\n",c);            /* 输出字符 */
printf("c =%x\n",c);            /* 输出字符的ASCII码值 */
printf("s[ ]=%s\n",s);          /* 输出数组字符串 */
printf("s[ ]=%6.9s\n",s);          /* 输出最多9个字符的字符串 */
printf("s =%p\n",s);            /* 输出数组字符串首字符地址 */
printf(" *p=%s\n",p);           /* 输出指针字符串 */
printf("p =%p\n",p);           /* 输出指针的值 */
}

```

## 输出结果

```

a =1234
a = 1234
a =001234
a =1234
*i= 12
*i=12
i =06E4
f =3.141593
f =3.1416
x =0.123457
x =0.1234567898765430
c =A
c =41
s[ ]=Hello, Mr. wang.

```

```

s[ ]=Hello, Mr
s =FFBE
* p=Where are you going?
p =0194

```

上面结果中的地址值在不同计算机上可能不同。

## 二、scanf( )函数

scanf( )函数是格式化输入函数，它从标准输入设备(键盘)读取输入的信息。其调用格式为：

```
scanf("<格式化字符串>", <地址表>);
```

格式化字符串包括以下三类不同的字符：

1. 格式化说明符：格式化说明符与 printf( )函数中的格式化说明符基本相同。
2. 空白字符：空白字符会使 scanf( )函数在读操作中略去输入中的一个或多个空白字符。
3. 非空白字符：一个非空白字符会使 scanf( )函数在读入时剔除掉与这个非空白字符相同的字符。

地址表是需要读入的所有变量的地址，而不是变量本身。这与 printf( )函数完全不同，要特别注意，各个变量的地址之间用“,”分开。

### 例 3-2:

```

main( )
{
    int i, j;
    printf("i,j=?\n");
    scanf("%d,%d", &i,&j);
}

```

本例中的 scanf( )函数先读一个整型数，然后把接着输入的逗号剔除掉，最后读入另一个整型数。如果“,”这一特定字符没有找到，scanf( )函数就终止。若参数之间的分隔符为空格，则参数之间必须输入一个或多个空格。

注：

● 对于字符串数组或字符串指针变量，由于数组名和指针变量名本身就是地址，因此使用 scanf( )函数时，不需要在它们前面加上“&”操作符。

### 例 3-3:

```

main( )
{
    char * p,str[20];
    scanf("%s",p);      /* 从键盘输入字符串 */
    scanf("%s",str);
    printf("%s\n",p);    /* 向屏幕输出字符串 */
    printf("%s\n",str);
}

```

● 可以在格式化字符串中的“%”和格式化规定符之间加入一个整数，表示在任何读操作中的最大读入位数。

如例 3-3 中若规定只能输入 10 字符给字符串指针 p，则第一条 scanf( )函数语句变为

```
scanf("%10s",p);
```

程序运行时一旦输入字符个数大于 10，p 就不再继续读入了，而后面的一个读入函数即 scanf("%s",str)就会从第 11 个字符开始读入。

实际使用 scanf( )函数时存在一个问题，下面举例进行说明：

当使用多个 scanf( )函数连续给多个字符变量输入时，例如：

```
main( )
{
    char c1,c2;
    scanf("%c",&c1);
    scanf("%c",&c2);
    printf("c1 is %c, c2 is %c",c1,c2);
}
```

运行该程序，输入一个字符 A 后回车(要完成输入必须回车),在执行 scanf("%c",&c1)时，给变量 c1 赋值“A”，但回车符仍然留在缓冲区内，执行输入语句 scanf("%c",&c2)时，变量 c2 将接收一个回车符而使输入结束。输出结果中变量 c2 没有赋值，仔细观察可以发现 c2 输出的是一空行。如果输入 AB 后回车，那么输出结果为：c1 is A, c2 is B。

要解决以上问题，可以在输入函数前加入清除函数 fflush( )(这个函数的使用方法将在本章最后讲述)。修改以上程序变成：

```
#include <stdio.h>
main( )
{
    char c1,c2;
    scanf("%c",&c1);
    fflush(stdin);
    scanf("%c",&c2);
    printf("c1 is %c, c2 is %c",c1,c2);
}
```

### 3.1.2 非格式化输入输出函数

非格式化输入输出函数可以由上面讲述的标准格式化输入输出函数代替，但这些函数编译后代码少，相对占用内存也小，从而提高了速度，同时使用也比较方便。下面分别进行介绍。

#### 一、puts( )和 gets( )函数

##### 1. puts( )函数



puts( )函数用来向标准输出设备(屏幕)写字符串并换行,其调用格式为:

```
puts(s);
```

其中 s 为字符串变量(字符串数组名或字符串指针)。

puts( )函数的作用与语句 printf("%s\n",s)相同。

例 3-4:

```
main( )
{
    char s[20], *f;           /* 定义字符串数组和字符串指针变量 */
    strcpy(s, "Hello! Mr. Wang."); /* 数组字符串变量赋值 */
    f = "Fine, Thank you! and you?"; /* 指针字符串变量赋值 */
    puts(s);
    puts(f);
}
```

注:

- puts( )函数只能输出字符串,不能输出数值或进行格式变换。
- 可以将字符串直接写入 puts( )函数中。如:

```
puts("Hello, Mr. wang.");
```

## 2. gets( )函数

gets( )函数用来从标准输入设备(键盘)读取字符串直到回车结束,但回车符不属于这个字符串。其调用格式为:

```
gets(s);
```

其中 s 为字符串变量(字符串数组名或字符串指针)。

gets(s)函数与 scanf("%s",s)相似,但不完全相同,使用 scanf("%s",s)函数输入字符串时存在一个问题,就是如果输入了空格会认为输入字符串结束,空格后的字符将作为下一个输入项处理,但 gets( )函数将接收输入的整个字符串直到回车为止。

例 3-5:

```
main( )
{
    char s[20], *f;
    printf("What's your name?\n");
    gets(s);          /* 等待输入字符串直到回车结束 */
    puts(s);          /* 将输入的字符串输出 */
    puts("How old are you?");
    gets(f);
    puts(f);
}
```

注:

● gets(s)函数中的变量 s 为一字符串。如果为单个字符，编译连接不会有错误，但运行后会出现“Null pointer assignment”的错误。

二、 putchar( )、getch( )、getche( )和 getchar( )函数

1. putchar( )函数

putchar( )函数是向标准输出设备输出一个字符，其调用格式为：

```
putchar(ch);
```

其中 ch 为一个字符变量或常量。

putchar( )函数的作用等同于 printf("%c",ch)。

例 3-6:

```
#include <stdio.h>
main( )
{
    char c;           /* 定义字符变量 */
    c='A';           /* 给字符变量赋值 */
    putchar(c);       /* 输出该字符 */
    putchar('A');     /* 直接输出字母 A */
    putchar('\X4');   /* 输出字母 A */
    putchar(0x41);    /* 直接用 ASCII 码值输出字母 A */
}
```

例子中第一条语句#include <stdio.h> 的含义是调用另一个文件 stdio.h，这是一个头文件，其中包括全部标准输入输出库函数的数据类型定义和函数说明，Turbo C 对每个库函数使用的变量及函数类型都已作了定义与说明，放在相应头文件“\*.h”中，用户用到这些函数时必须要用#include <\*.h>或#include “\*.h”语句调用相应的头文件，以供连接。若没有用此语句说明，则连接时将会出现错误。

printf( )、scanf( )、puts( )、gets( )四个函数的头文件包含说明可以省略。本章所讲的其它输入输出函数，其头文件都是 stdio.h，在下面介绍中将不再重复。

从例 3-6 中连续的四个字符输出函数语句，可以分清字符变量的不同赋值方法。

2. getch( )、getche( )和 getchar( )函数

(1) getch( )和 getche( )函数

这两个函数都是从键盘上读入一个字符。其调用格式为：

```
getch( );
```

```
getche( );
```

两者的区别是：getch( )函数不把读入的字符回显在显示屏幕上，而 getche( )函数却将读入的字符回显到显示屏幕上。

例 3-7:

```
#include <stdio.h>
main( )
{
    char c, ch;
```

```

c=getch( );    /* 从键盘上不回显读入一个字符送给字符变量 c */
putchar(c);    /* 输出该字符 */
ch=getche( );  /* 从键盘上带回显读入一个字符送给字符变量 ch */
putchar(ch);
}

```

利用回显和不回显的特点，这两个函数经常用于交互输入的过程中完成暂停等功能。

### 例 3-8:

```

#include <stdio.h>
main( )
{
    char c, s[20];
    printf("Name :");
    gets(s);
    printf("Press any key to continue...");
    getch( );    /* 等待输入任一键 */
}

```

### (2) getchar( )函数

getchar( )函数也是从键盘上读入一个字符，并带回显。它与前面两个函数的区别在于：getchar( )函数等待输入直到按回车才结束，回车前的所有输入字符都会逐个显示在屏幕上，但只有第一个字符作为函数的返回值。

getchar( )函数的调用格式为：

```
getchar( );
```

### 例 3-9:

```

#include <stdio.h>
main( )
{
    char c;
    c=getchar( );    /* 从键盘读入字符直到回车结束 */
    putchar(c);    /* 显示输入的第一个字符 */
    getch( );    /* 等待按任一键 */
}

```

## 3.2 文件的输入输出函数

键盘、显示器、打印机、磁盘驱动器等逻辑设备，其输入输出都可以通过文件管理的方法来完成。而在编程时使用最多的要算是磁盘文件，因此本节主要以磁盘文件为主，详细介绍 Turbo C 提供的文件操作函数，当然这些对文件的操作函数也适合于非磁盘文件的情况。

另外，Turbo C 提供了两类关于文件的函数。一类称做标准文件函数也称缓冲型文

件函数，这是 ANSI 标准定义的函数；另一类叫非标准文件函数，也称非缓冲型文件函数，这类函数最早仅用于 UNIX 操作系统，但现在 MS-DOS3.0 以上版本的操作系统也可以使用。下面分别进行介绍。

### 3.2.1 标准文件函数

标准文件函数主要包括文件的打开、关闭、读和写等函数。诸如 BASIC 和 FORTRAN 等语言有顺序文件和随机文件之分，在打开时就应按不同的方式确定，而 Turbo C 并不区分这两种文件，但提供了两组函数，即顺序读写函数和随机读写函数。

#### 一、文件的打开和关闭

任何一个文件在使用之前和使用之后，必须要进行打开和关闭，这是因为操作系统对于同时打开的文件数目是有限制的。DOS 操作系统中，可以在 DEVICE.SYS 中定义允许同时打开的文件数  $n$  (用 `files=n` 定义)。其中  $n$  为可同时打开的文件数，一般  $n \leq 20$ 。因此在使用文件前应打开文件，才可对其中的信息进行存取，用完之后需要关闭，否则将会出现一些意想不到的错误。Turbo C 提供了打开和关闭文件的函数。

#### 1. fopen( )函数

fopen( ) 函数用于打开文件，其调用格式为：

```
FILE *fopen(char *filename, *type);
```

在介绍这个函数之前，先了解一下下面的知识。

#### (1) 流(stream)和文件(file)

流和文件在 Turbo C 中是有区别的，Turbo C 为编程者和被访问的设备之间提供了一层抽象的东西，称之为“流”，而将具体的实际设备叫做文件。流是一个逻辑设备，具有相同的行为。因此，用来进行磁盘文件写的函数也同样可以用来进行打印机的写入。在 Turbo C 中有两种性质的流：文字流(text stream)和二进制流(binary stream)。对磁盘来说，就是文本文件和二进制文件。本书为了便于读者理解 Turbo C 语言而没有对流和文件作特别的区分。

#### (2) 文件指针 FILE

实际上 FILE 是一个新的数据类型。它是 Turbo C 的基本数据类型的集合，称之为结构指针。有关结构的概念将在第五章中详细介绍，这里只要将 FILE 理解为一个包括了文件管理有关信息的数据结构，即在打开文件时必须先定义一个文件指针。

(3) 以后介绍的函数调用格式时将直接写出形式参数的数据类型和函数返回值的数据类型。例如：上面打开文件的函数，返回一个文件指针，其中形式参数有两个，均为字符型变量(字符串数组或字符串指针)。本书不再对函数的调用格式作详细说明。

现在再来看打开文件函数的用法。

fopen( ) 函数中第一个形式参数表示文件名，可以包含路径和文件名两部分。如：

```
"B:TEST.DAT"
```

```
"c:\tc\test.dat"
```

如果将路径写成 "c:\tc\test.dat" 是不正确的，这一点要特别注意。

第二个形式参数表示打开文件的类型。关于文件类型的规定参见表 3-1。

表 3-1 文件操作类型

字符	含 义
" r "	打开文字文件只读
" w "	创建文字文件只写
" a "	增补, 如果文件不存在则创建一个
" r+ "	打开一个文字文件读 / 写
" w+ "	创建一个文字文件读 / 写
" a+ "	打开或创建一个文件增补
" b "	二进制文件(可以和上面每一项合用)
" t "	文字文件(默认项)

如果要打开一个 ccdos 子目录中, 文件名为 CLIB 的二进制文件, 可写成:

```
fopen("c:\\ccdos\\clib","rb");
```

如果成功地打开一个文件, fopen( )函数就返回文件指针; 否则返回空指针(NULL)。由此可判断文件打开是否成功。

## 2. fclose( )函数

fclose( )函数用来关闭一个由 fopen( )函数打开的文件, 其调用格式为:

```
int fclose (FILE * stream);
```

该函数返回一个整型数。当文件关闭成功时, 返回 0; 否则返回一个非零值。可以根据函数的返回值判断文件是否关闭成功。

### 例 3-10:

```
#include <stdio.h>
{
    FILE * fp;           /* 定义一个文件指针 */
    int i;
    fp = fopen("CLIB","rb"); /* 打开当前目录中名为 CLIB 的文件只读 */
    if(fp == NULL)         /* 判断文件打开是否成功 */
        puts("File open error"); /* 提示打开不成功 */
    i = fclose(fp);        /* 关闭打开的文件 */
    if(i == 0)             /* 判断文件关闭是否成功 */
        printf("O.K.");    /* 提示关闭成功 */
    else
        puts("File close error"); /* 提示关闭不成功 */
}
```

## 二、有关文件操作的函数

本节所讲的文件读写函数均是指顺序读写, 即读写了一条信息后, 指针自动加 1。下面分别介绍写操作函数和读操作函数。

### 1. 文件的顺序写函数

fprintf( )、fputs( )和 fputc( )函数

函数 `fprintf()`、`fputs()` 和 `fputc()` 均为文件的顺序写操作函数，其调用格式如下：

```
int fprintf(FILE * stream, char * format, <variable-list>);
int fputs(char * string, FILE * stream);
int fputc(int ch, FILE * stream);
```

上述三个函数的返回值均为整型量。`fprintf()` 函数的返回值为实际写入文件中的字符个数(字节数)。如果写错误，则返回一个负数。`fputs()` 函数返回 0 时表明将 `string` 指针所指的字符串写入文件中的操作成功，返回非 0 时，表明写操作失败。`fputc()` 函数若返回一个向文件所写字符的值，此时写操作成功，否则返回 EOF(文件结束符，其值为 -1，在 `stdio.h` 中定义)表示写操作错误。

`fprintf()` 函数中格式化的规定与 `printf()` 函数相同，所不同的只是 `fprintf()` 函数是向文件中写入，而 `printf()` 是向屏幕输出。

下面介绍一个例子，运行后产生一个 `test.dat` 的文件。

### 例 3-11:

```
#include <stdio.h>
main( )
{
    char * s = "That's a good news.";    /* 定义字符串指针并初始化 */
    int i = 617;                          /* 定义整型变量并初始化 */
    FILE * fp;                            /* 定义文件指针 */
    fp = fopen("test.dat", "w");          /* 建立一个文字文件只写 */
    fputs("Your score of TOEFL is", fp); /* 向所建文件写入一串字符 */
    fputc(':', fp);                       /* 向所建文件写冒号 */
    fprintf(fp, "%d \n", i);              /* 向所建文件写入一整型数 */
    fprintf(fp, "%s", s);                 /* 向所建文件写一字符串 */
    fclose(fp);                          /* 关闭文件 */
}
```

用 DOS 的 TYPE 命令显示 `test.dat` 的内容如下所示：

屏幕显示

Your score of TOEFL is:617

That's a good news.

## 2. 文件的顺序读操作函数

`fscanf()`、`fgets()` 和 `fgetc()` 函数

函数 `fscanf()`、`fgets()` 和 `fgetc()` 均为文件的顺序读操作函数，其调用格式如下：

```
int fscanf(FILE * stream, char * format, <address-list>);
char fgets(char * string, int n, FILE * stream);
int fgetc(FILE * stream);
```

`fscanf()` 函数的用法与 `scanf()` 函数相似，只是它是从文件中读取信息。`fscanf()` 函数的返回值若为 EOF(即 -1)，表明读错误，否则读数据成功。`fgets()` 函数从文件中读取

至多  $n-1$  个字符( $n$  用来指定字符数), 并把它们放入 `string` 指向的字符串中, 在读入之后自动向字符串末尾加一个空字符, 读成功返回 `string` 指针, 失败返回一个空指针。`fgetc()` 函数返回文件当前位置的一个字符, 读错误时返回 `EOF`。

下面的程序读取例 3-11 产生的 `test.dat` 文件, 并将读出的结果显示在屏幕上。

#### 例 3-12:

```
#include <stdio.h>
main( )
{
    char * s,m[20];
    int i;
    FILE * fp;
    fp=fopen("test.dat","r");      /* 打开文字文件只读 */
    fgets(s,24,fp);                 /* 从文件中读取 23 个字符 */
    printf("%s",s);                 /* 输出所读的字符串 */
    fscanf(fp,"%d",&i);             /* 读取整型数 */
    printf("%d",i);                 /* 输出所读整型数 */
    putchar(fgetc(fp));             /* 读取一个字符同时输出 */
    fgets(m,17,fp);                 /* 读取 16 个字符 */
    puts(m);                        /* 输出所读字符串 */
    fclose(fp);                     /* 关闭文件 */
    getch( );                       /* 等待按任一键 */
}
```

运行后屏幕显示:

```
Your score of TOEFL is:617
That's a good news.
```

如果将例中 `fscanf(fp,"%d",&i)` 改为 `fscanf(fp,"%s",m)`, 再将其后的输出语句改为 `printf("%s",m)`, 则可得出同样的结果。由此可见, Turbo C 中只要是读文字文件, 则不论是字符还是数字都将按其 ASCII 值处理。另外, 还要说明的一点就是 `fscanf()` 函数读到空白符时, 便自动结束, 在使用时要特别注意。

### 3. 文件的随机读写

有时用户想直接读取文件中某处的信息, 若用文件的顺序读写必须从文件头开始直到要求的文件位置再读, 这显然不方便。Turbo C 提供了一组文件的随机读写函数, 即将文件位置指针定位在所要求读写的地方直接读写。

文件的随机读写函数如下:

```
int fseek(FILE * stream,long offset,int fromwhere);
int fread(void * buf,int size,int count,FILE * stream);
int fwrite(void * buf,int size,int count,FILE * stream);
long ftell(FILE * stream);
```

`fseek()` 函数的作用是将文件的位置指针设置到从 `fromwhere` 开始的第 `offset` 个字节

的位置上, 其中 `fromwhere` 是下列几个宏定义之一:

文件位置指针起始计算位置 `fromwhere`

符号常数	数值	含 义
<code>SEEK_SET</code>	0	从文件开头
<code>SEEK_CUR</code>	1	从文件指针的现行位置
<code>SEEK_END</code>	2	从文件末尾

`offset` 是指文件位置指针从指定开始位置(`fromwhere` 指出的位置)跳过的字节数, 它是一个长整型量, 以支持大于 64K 字节的文件。`fseek()` 函数一般用于对二进制文件进行操作。

当 `fseek()` 函数返回 0 时表明操作成功, 返回非 0 表示失败。

下面的程序从二进制文件 `test_b.dat` 中读取第 8 个字节。

### 例 3-13:

```
#include <stdio.h>
main( )
{
    FILE *fp;
    if((fp=fopen("test_b.dat","rb"))==NULL) /* 打开二进制文件只读,
                                              并判断打开是否成功 */
    {
        printf("Can't open file"); /* 不成功时提示出错信息 */
        exit(1); /* 终止程序 */
    }
    fseek(fp,8,SEEK_SET); /* 文件指针定位 */
    fgetc(fp); /* 读取一个字符 */
    fclose(fp); /* 关闭文件 */
}
```

`fread()` 函数是从文件中读取 `count` 个字段, 每个字段长度为 `size` 个字节, 并把它们存放到 `buf` 指针所指向的缓冲器中。

`fwrite()` 函数是把 `buf` 指针所指的缓冲器中, 长度为 `size` 个字节的 `count` 个字段写到 `stream` 指向的文件中去。

随着读和写字节数的增大, 文件位置指示器也增大, 读多少个字节, 文件位置指示器相应也跳过多少个字节。读写完毕函数返回所读和所写的字段个数。

`ftell()` 函数返回文件位置指示器的当前值, 这个值是指示器从文件头开始算起的字节数, 返回的数为长整型数; 当返回 -1 时, 表明出现错误。

下面的程序把一个浮点数组以二进制方式写入文件 `test_b.dat` 中。

### 例 3-14:

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



```

float f[6] = {3.2, -4.34, 25.04, 0.1, 50.56, 80.5}; /* 定义浮点数组并初始化 */
int i;
FILE *fp;
fp = fopen("test__b.dat", "wb"); /* 创建一个二进制文件只写 */
fwrite(f, sizeof(float), 6, fp); /* 将 6 个浮点数写入文件中 */
fclose(fp); /* 关闭文件 */
}

```

下面的例子从一个叫做 test\_\_b.dat 的文件中读取 100 个整型数，并把它们放到数组 dat 中。

### 例 3-15:

```

#include <stdio.h>
main( )
{
    FILE *fp;
    int dat[100];
    fp = fopen("test__b.dat", "rb"); /* 打开一个二进制文件只读 */
    if(fread(dat, sizeof(int), 100, fp) != 100) /* 判断是否读了 100 个数 */
    {
        if(feof(fp))
            printf("End of file") /* 不到 100 个数文件结束 */
        else
            printf("Read error"); /* 提示读数错误 */
    }
    fclose(fp) /* 关闭文件 */
}

```

注:

- 当用标准文件函数对文件进行读写操作时，首先将所读写的内容放进缓冲区，即写函数只对输出缓冲区进行操作，读函数只对输入缓冲区进行操作。例如向一个文件写入内容，所写的内容将首先放在输出缓冲区中，直到输出缓冲区存满或使用 fclose( )函数关闭文件时，缓冲区的内容才会写入文件中。若无 fclose( )函数，则不会向文件中存入所写的内容或写入的文件内容不全。有一个对缓冲区进行刷新的函数，即 fflush( )，其调用格式为:

```
int fflush(FILE *stream);
```

该函数将输出缓冲区的内容实际写入文件中，而将输入缓冲区的内容清除掉。

#### 4. fcof( )和 rewind( )函数

这两个函数的调用格式为:

```

int fcof(FILE *stream);
int rewind(FILE *stream);

```

feof( )函数检测文件位置指示器是否到达了文件结尾，若是则返回一个非 0 值，否则返回 0。这个函数对二进制文件操作特别有用，因为在二进制文件中，文件结尾标志 EOF 也是一个合法的二进制数，只简单地检查读入字符的值来判断文件是否结束是不行的。如果那样的话，可能会造成文件未结束而被误认为已结束，所以就必须使用 feof( ) 函数。

下面的这条语句是常用的判断文件是否结束的方法。

```
while(!feof(fp)) fgetc(fp);
```

while 为循环语句，将在下一章介绍。

rewind( )函数用于把文件位置指示器移到文件的起点处，成功时返回 0；反之，则返回非 0 值。

### 3.2.2 非标准文件函数

这类函数最早用于 UNIX 操作系统，ANSI 标准未作定义，但有时也经常用到，DOS3.0 以上版本都支持这些函数。它们的头文件为 io.h。

一、文件的打开和关闭

#### 1. open( )函数

open( )函数的作用是打开文件，其调用格式为：

```
int open(char * filename, int access);
```

该函数表示按 access 的要求打开名为 filename 的文件，返回值为文件描述字，其中 access 有两部分内容：基本模式和修饰符，两者用“ ”(“或”)方式连接。修饰符可以有多个，但基本模式只能有一个。access 的规定如表 3-2。

表 3-2 access 的规定

基本模式	含 义	修饰符	含 义
O_RDONLY	只读	O_APPEND	文件指针指向末尾
O_WRONLY	只写	O_CREAT	文件不存在时创建文件，属性按基本模式属性
O_RDWR	读写	O_TRUNC	若文件存在，将其长度缩为 0，属性不变
		O_BINARY	打开一个二进制文件
		O_TEXT	打开一个文字文件

若 open( )函数打开成功，返回值就是文件描述字的值(非负值)，否则返回-1。

#### 2. close( )函数

close( )函数的作用是关闭由 open( )函数打开的文件，其调用格式为：

```
int close(int handle);
```

该函数关闭与文件描述字 handle 相连的文件。

二、读写函数

#### 1. read( )函数

read( )函数的调用格式为:

```
int read(int handle,void *buf,int count);
```

read( )函数从 handle(文件描述字)相连的文件中, 读取 count 个字节放到 buf 所指向的缓冲区中, 返回值为实际所读字节数, 返回-1 时表示出错, 返回 0 则表示文件结束。

## 2. write( )函数

write( )函数的调用格式为:

```
int write(int handle,void *buf,int count);
```

write( )函数把 count 个字节从 buf 指向的缓冲区写入与 handle 相连的文件中, 返回值为实际写入的字节数。

## 三、随机定位函数

### 1. lseek( )函数

lseek( )函数的调用格式为:

```
int lseek(int handle,long offset,int fromwhere);
```

该函数对与 handle 相连的文件位置指针进行定位, 功能和用法与 fseek( )函数相同。

### 2. tell( )函数

tell( )函数的调用格式为:

```
long tell(int handle);
```

该函数返回与 handle 相连的文件现行位置指针, 功能和用法与 ftell( )函数相同。

## 第四章 流程控制语句

Turbo C 提供了丰富、灵活的流程控制语句，主要有：条件语句、循环语句和开关语句。本章将对这些语句作详细介绍。

### 4.1 条件语句

象其它语言一样，Turbo C 也提供条件语句。在 Turbo C 中条件语句的一般格式为：

```
if(表达式)
```

```
    语句 1;
```

```
else
```

```
    语句 2;
```

上述结构表示：如果表达式的值为非 0(true)即真，则执行语句 1，执行完语句 1 转到语句 2 的后面继续向下执行；如果表达式的值为 0(false)即假，则跳过语句 1 而执行语句 2。所谓表达式是指关系表达式和逻辑表达式的结合式，关于表达式第二章中已作过介绍，这里不再重复。

注：

- 条件执行语句中的“else 语句 2;”部分是可选项，因此可以缺省，此时条件语句变成：

```
if(表达式) 语句 1;
```

表示若表达式的值为非 0 则执行语句 1，否则跳过语句 1 继续执行。

- 如果语句 1 或语句 2 部分有多于一条语句要执行时，必须使用“{”和“}”把这些语句包括在其中，此时条件语句形式为：

```
if(表达式)
```

```
{
```

```
    语句体 1;
```

```
}
```

```
else
```

```
{
```

```
    语句体 2;
```

```
}
```

- 条件语句可以嵌套，这种情况经常碰到，但条件嵌套语句容易出错，其原因主要是不知道哪个 if 对应哪个 else。

例如：

```

if(x > 10 || x < -10)
if(y <= 100 && y > x)
    printf("Yes");
else
    printf("No");

```

对于上述情况, Turbo C 规定: else 语句与最近的一个 if 语句匹配, 上例中的 else 与 if(y <= 100 && y > x) 相匹配。如要使 else 与 if(x > 10 || x < -10) 相匹配, 必须用花括号, 如下所示:

```

if(x > 10 || x < -10)
{
    if(y <= 100 && y > x)
        printf("Yes");
}
else
    printf("No");

```

● 可采用阶梯式 if-else-if 结构。

阶梯式结构的一般形式为:

```

if(表达式 1)
    语句 1;
else if(表达式 2)
    语句 2;
else if(表达式 3)
    语句 3;
    :
else
    语句 n;

```

这种结构是从上到下逐个对条件进行判断, 一旦发现有某个条件满足就执行与它有关的语句, 并跳过其它剩余阶梯; 若没有一个条件满足, 则执行最后一个 else 后的语句 n。最后这个 else 常起着“缺省条件”的作用。

同样, 如果对于某个条件有多于一条语句需要执行时, 必须使用“{”和“}”把这些语句包括在其中。

## 4.2 循环语句

Turbo C 提供三种基本的循环语句: for 语句、while 语句和 do-while 语句。

### 4.2.1 for 循环

for 循环是开界的。它的一般格式为:

```

for(<初始化>; <条件表达式>; <增量>)
    语句;

```

初始化总是一个赋值语句，它用来给循环控制变量赋初值；条件表达式是一个关系表达式，它决定什么时候退出循环；增量部分规定循环控制变量在每循环一次后按什么方式变化，这三个部分之间用“;”分开。

例如：

```
for(i=1; i<=10; i++)
```

语句;

本例中首先给 i 赋初值 1，判断 i 是否小于等于 10，若是则执行语句，之后值增加 1，再重新判断，直到条件为假，即 i>10 时，结束循环。

注：

- for 循环中的语句可以为语句体，但也要用“{”和“}”将参与循环的语句括起来。
- for 循环中的“初始化”、“条件表达式”和“增量”都是可选项；即可以缺省，但“;”不能缺省。省略了初始化，表示不对循环控制变量赋初值。省略了条件表达式时，如不做其它处理便成为死循环。省略了增量，则不对循环控制变量进行操作，这时可在语句体中加入修改循环控制变量的语句。
- for 循环可以有多层嵌套。

例 4-1:

```
main( )
{
    int i,j,k;
    printf("i j k\n");
    for(i=0;i<2;i++)
        for(j=0;j<2;j++)
            for(k=0;k<2;k++)
                printf("%d %d %d\n",i,j,k);
}
```

输出结果为：

```
i j k
0 0 0
0 0 1
0 1 0
0 1 1
1 0 0
1 0 1
1 1 0
1 1 1
1 1 1
```

#### 4.2.2 while 循环

while 循环的一般格式为：

```
while(条件)
    语句;
```

while 循环表示当条件为真时便执行语句，直到条件为假才结束循环，并继续执行循环程序外的后续语句。

#### 例 4-2:

```
#include <stdio.h>
main( )
{
    char c;
    c='\0';          /* 初始化 c */
    while(c!='\X0D') /* 回车结束循环 */
        c=getche( ); /* 带回显的从键盘接收字符 */
}
```

本例中，while 循环以检查 c 是否为回车符开始，因其事先被初始化为空，所以条件为真，进入循环后等待键盘输入字符；一旦输入回车，则 c='\X0D'，While 条件为假，循环便告结束。

与 for 循环一样，while 循环也总是在循环的头部检验条件，这就意味着循环可能什么也不执行就退出。

注：

- while 循环体也允许是空语句。

例如：

```
while((c=getche( ))!='\X0D');
```

这个循环直到键入回车为止。

- 可以有多层循环嵌套。
- 语句可以是语句体，此时必须用“{”和“}”括起来。

#### 例 4-3:

```
#include <stdio.h>
main( )
{
    char c, fname[13];
    FILE *fp;          /* 定义文件指针 */
    printf("File name:"); /* 提示输入文件名 */
    scanf("%s",fname);  /* 等待输入文件名 */
    fp=fopen(fname, "r"); /* 打开文件只读 */
    while((c=fgetc(fp))!=EOF) /* 读取一个字符并判断是否到文件结束 */
        putchar(c);      /* 文件未结束时显示该字符 */
}
```

### 4.2.3 do-while 循环

do-while 循环的一般格式为:

```
do
    语句;
while(条件);
```

这个循环与 while 循环的不同在于: 它先执行循环中的语句, 然后再判断条件是否为真, 如果为真则继续循环; 如果为假, 则终止循环。因此, do-while 循环至少要执行一次循环语句。

同样当有许多语句参加循环时, 也要用“{”和“}”把它们括起来。

### 4.3 开关语句

在编写程序时, 经常会碰到按不同情况分转的多路问题, 这时可用嵌套 if-else-if 语句来实现, 但 if-else-if 语句使用并不方便, 并且容易出错。对于这种情况, Turbo C 提供了一种开关语句。开关语句的格式为:

```
switch(变量)
{
    case 常量 1:
        语句 1 或空;
    case 常量 2:
        语句 2 或空;
        .
        .
        .
    case 常量 n:
        语句 n 或空;
    default:
        语句 n+1 或空;
}
```

执行 switch 开关语句时, 将变量与逐个 case 后的常量进行比较, 若与其中一个相等, 则执行该常量后面的语句, 若不与任何一个常量相等, 则执行 default 后面的语句。

注:

- switch 后的变量可以是数值型, 也可以是字符型。
- 可以省略一些 case 和 default。
- 每个 case 或 default 后的语句可以是语句体, 但不需要使用“{”和“}”括起来。

下例的 switch 变量为整数型。

**例 4-4:**

```
main( )
```



```

{
    int test;
    for(test=0;test<=10;test++) /* test 作为循环控制变量的 for 循环 */
    {
        switch(test) /* 变量为整型数的开关语句 */
        {
            case 1:
                printf("%d\n",test);
                break;
            case 2:
                printf("%d\n",test);
                break;
            case 3:
                printf("%d\n",test);
                break;
            default:
                puts("Error");
                break;
        }
    }
}

```

下例的 switch 变量为字符型。

#### 例 4-5:

```

#include <stdio.h>
main( )
{
    char c;
    while(c!=27) /* 循环直到按 Esc 键结束 */
    {
        c=getch( ); /* 从键盘不回显接收一个字符 */
        switch(c)
        {
            case 'A': /* 接收的字符为'A' */
                putchar(c);
                break; /* 退出开关语句 */
            case 'B': /* 接收的字符为'B' */
                putchar(c);
                break;
            default: /* 接收的字符非'A'非'B' */
                puts("Error");
                break;
        }
    }
}

```

```

    }
}
}

```

## 4.4 break、continue 和 goto 语句

### 4.4.1 break 语句

break 语句通常用在循环语句和开关语句中。当 break 用于 switch 开关语句中时，可以使程序跳出 switch 语句而执行它后面的语句；如果没有 break 语句，则将成为一个死循环而无法退出。break 在 switch 语句中的用法已在前面介绍开关语句时的例子中碰到，这里不再举例。

当 break 语句用于 do-while、for、while 循环语句中时，可使程序终止循环而执行循环后面的语句，通常 break 语句总是与 if 语句结合使用，在满足条件时便跳出循环。

例 4-6:

```

main( )
{
    int i=0;
    char c;
    while(1)          /* 设置循环 */
    {
        c='\0';          /* 给循环控制变量赋空字符 */
        while(c!=13&& c!=27) /* 从键盘接收字符直到按回车或 Esc 键 */
        {
            c=getch( );
            printf("%c\n",c);
        }
        if(c==27) break;    /* 判断若按 Esc 键则退出循环 */
        i++;
        printf("The No. is %d\n",i);
    }
    printf("The end.");
}

```

注:

- break 语句对 if-else 的条件语句不起作用。
- 在多层循环中，一个 break 语句只能向外跳一层。

### 4.4.2 continue 语句

continue 语句的作用是跳过循环体中剩余的语句而强制执行下一次循环。

continue 语句只用在 for、while、do-while 等循环体中，常与 if 条件语句一起使用，用来加速循环。

例 4-7:

```

main( )
{
    char c;
    while(c!=0X0D)      /* 不是回车符循环 */
    {
        c=getch( );
        if(c==0X1B)     /* 若按 Esc 键不输出便进行下次循环 */
            continue;
        printf("%c\n",c);
    }
}

```

#### 4.4.3 goto 语句

goto 语句是一种无条件转移语句，与 BASIC 中的 goto 语句相似。goto 语句的使用格式为：

goto 标号;

其中标号是 Turbo C 中一个有效的标识符，这个标识符加上一个“:”一起出现在函数内某处，执行 goto 语句后，程序将跳转到该标号处并执行其后的语句。另外标号必须与 goto 语句同处于一个函数中，但可以不在一个循环层中。通常 goto 语句与 if 条件语句连用，在满足某一条件时，程序跳到标号处运行。

goto 语句通常不用，主要因为它将使程序层次不清，且不易读，但需要从多层嵌套退出时，用 goto 语句则比较合理。

例 4-6 用 goto 语句时变为：

例 4-8:

```

main( )
{
    int i=0;
    char c;
    while(1)          /* 设置循环 */
    {
        c='\0';
        while(c!=13)  /* 设置循环直到判断是回车符 */
        {
            c=getch( );      /* 等待键盘输入一字符 */
            if(c==27) goto quit; /* 若按 Esc 键跳转到 quit 标号处 */
            printf("%c\n",c); /* 输出键盘接收的字符 */
        }
        i++;
        printf("The No. is %d\n",i);
    }
}

```

```
quit:
    printf("The end");
}
```

## 第五章 结构、联合和枚举

本章介绍 Turbo C 新的数据类型：结构、联合和枚举，其中结构和联合是以前讲过的五种基本数据类型的组合，枚举是一个被命名为整型常数的集合。本章对这些数据类型分别进行阐述。

### 5.1 结构(struct)

结构是由基本数据类型构成的，并用一个标识符来命名的各种变量的组合。结构中可以使用不同的数据类型。

#### 5.1.1 结构说明和结构变量定义

在 Turbo C 中，结构也是一种数据类型，可以使用结构变量。因此，象其它类型的变量一样，在使用结构变量时要先对其定义。

定义结构变量的一般格式为：

```
struct 结构名
{
    类型 变量名;
    类型 变量名;
    .
    .
    .
} 结构变量;
```

结构名是结构的标识符，不是变量名。

类型为第二章中所讲述的数据类型。

构成结构的每一个类型变量称为结构成员，它象数组的元素一样，但数组中元素是以下标来访问的，而结构是按变量名字来访问成员的。

下面举一个例子来说明怎样定义结构变量。

```
struct wage
{
    char name[8];
    int age;
    char sex[2];
    char depart[20];
    float wage1,wage2,wage3,wage4,wage5;
} person;
```

这个例子定义了一个结构名为 wage 的结构变量 person。如果省略变量名 person，则

变成对结构的说明。用已说明的结构名也可定义结构变量。这样定义时上例变成:

```
struct wage
{
    char name[8];
    int age;
    char sex[2];
    char depart[20];
    float wage1,wage2,wage3,wage4,wage5;
};
struct wage person;
```

如果需要定义多个具有相同形式的结构变量,则采用这种方法比较方便,它先作结构说明,再用结构名来定义变量。

例如:

```
struct wage Tianyr,Wangjz, .....;
```

如果省略结构名,则称之为无名结构,这种情况常常出现在函数内部,用这种结构时上述的例子又变成:

```
struct {
    char name[8];
    int age;
    char sex[2];
    char depart[20];
    float wage1,wage2,wage3,wage4,wage5;
} Qianyr,Wangjz;
```

### 5.1.2 结构变量的使用

结构是一个新的数据类型,因此结构变量也可以象其它类型的变量一样赋值、运算,不同的是结构变量以成员作为基本变量。

结构成员的表示方式为:

结构变量.成员名

如果将“结构变量.成员名”看成一个整体,则这个整体的数据类型与结构中该成员的数据类型相同,这样就可象前面所讲的变量那样使用。

下面这个例子定义了一个结构变量,其中每个成员都从键盘接收数据,然后对结构中的浮点数求和,并显示运算结果,同时将数据以文本方式存入一个名为 wage.dat 的磁盘文件中。请注意这个例子中对不同结构成员的访问。

例 5-1:

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

```

struct {                                /* 定义一个结构变量 */
    char name[8];
    int age;
    char sex[2];
    char depart[20];
    float wage1,wage2,wage3,wage4,wage5;
} w;
FILE * fp;                             /* 定义文件指针 */
float wage;
char c='y';
fp=fopen("wage.dat","w"); /* 创建一个文件只写 */
while(c=='Y' || c=='y')      /* 判断是否继续循环 */
{
    printf("\nName: ");
    scanf("%s",w.name);      /* 输入姓名 */
    printf("Age: ");
    scanf("%d",&w.age);     /* 输入年龄 */
    printf("Sex: ");
    scanf("%s",w.sex);       /* 输入性别 */
    printf("Dept: ");
    scanf("%s",w.depart);    /* 输入部门 */
    printf("Wage1: ");
    scanf("%f",&w.wage1);   /* 输入单项工资 1 */
    printf("Wage2: ");
    scanf("%f",&w.wage2);   /* 输入单项工资 2 */
    printf("Wage3: ");
    scanf("%f",&w.wage3);   /* 输入单项工资 3 */
    printf("Wage4: ");
    scanf("%f",&w.wage4);   /* 输入单项工资 4 */
    printf("Wage5: ");
    scanf("%f",&w.wage5);   /* 输入单项工资 5 */
    wage=w.wage1+w.wage2+w.wage3+w.wage4+w.wage5; /* 求和 */
    printf("The sum of wages is %6.2f\n",wage); /* 屏幕显示结果 */
    fprintf(fp,"%10s%4d%4s%30s%10.2f\n",      /* 文件写入 */
        w.name,w.age,w.sex,w.depart,wage);
    while(1) /* 设置循环 */
    {
        printf("Continue ? <Y/N>");
        c=getche();
        if(c=='Y', c=='y', c=='N', c=='n')break; /* 判断是否退出循环 */
    }
}

```

```
fclose(fp);  
}
```

### 5.1.3 结构数组和结构指针

结构是一种新的数据类型，同样也可以有结构数组和结构指针。

#### 一、结构数组

结构数组就是具有相同结构类型的变量集合。假如要定义一个班级 32 个学生的姓名、性别、年龄和住址，可以定义成一个结构数组。如下所示：

```
struct{  
    char name[8];  
    char sex[2];  
    int age;  
    char addr[40];  
} student[32];
```

也可定义为：

```
struct info  
{  
    char name[8];  
    char sex[2];  
    int age;  
    char addr[40];  
};  
struct info student[32];
```

需要指出的是结构数组成员的访问是以数组元素为结构变量的，其形式为：

结构数组元素.成员名

例如：

```
student[0].name  
student[22].age
```

实际上，结构数组相当于一个二维构造，第一维是结构数组元素，每个元素是一个结构变量，第二维是结构成员。

注：

● 结构数组的成员也可以是数组变量。

例如：

```
struct x  
{  
    int m[2][5];  
    float f;  
    char s[20];  
}
```



```
}y[4];
```

为了访问结构 x 中结构变量 y[2]的 m[1][4]这个变量,可写成

```
y[2].m[1][4]
```

## 二、结构指针

结构指针是指向结构的指针。它由一个加在结构变量名前的“\*”操作符来定义,例如用前面已说明的结构定义一个结构指针如下:

```
struct wage{
    char name[8];
    int age;
    char sex[2];
    char depart[20];
    float wage1,wage2,wage3,wage4,wage5;
} * person__p;
```

也可省略结构指针名只作结构说明,然后再用下面的语句定义结构指针。

```
struct wage * person__p;
```

使用结构指针对结构成员的访问,与结构变量对结构成员的访问在表达方式上有所不同。结构指针对结构成员的访问表示为:

结构指针名->结构成员

其中“->”是两个符号“-”和“>”的组合,好象一个箭头指向结构成员。例如要给上面定义的结构中 name 和 age 赋值,可以用下面语句:

```
strcpy(person__p->name,"Qian Y.R");
person__p->age = 22;
```

实际上, person\_\_p->name 就是(\* person\_\_p).name 的缩写形式。

需要指出的是结构指针是指向结构的一个指针,即结构中第一个成员的首地址,因此在使用之前应该对结构指针初始化,即分配整个结构长度的字节空间,这可用下面的函数来完成,仍以上例来说明如下:

```
person__p=(struct wage *) malloc(sizeof(struct wage));
```

sizeof(stuct wage)自动求取 wage 结构的字节长度, malloc( )函数定义了一个大小为结构长度的内存区域,然后将其首地址作为结构指针返回。

注:

- 由于结构也是一种数据类型,因此定义的结构变量或结构指针变量同样也有局部变量和全程变量之分,视定义的位置而定。
- 结构变量名并不是指向该结构的地址,这与数组名的含义不同,因此若需要结构第一个成员的首地址时应该用&[结构变量名]。

### 5.1.4 结构的复杂形式

#### 一、嵌套结构

嵌套结构是指在一个结构成员中可以包含另一个结构, Turbo C 允许这种嵌套。

例如: 下面是一个有嵌套的结构

```
struct info{
    char name[8];
    int age;
    struct addr address;
} classmate;
```

其中: addr 为另一个结构的结构名, 必须先进行说明, 即

```
struct addr{
    char city[20];
    unsigned long zip;
    char tel[14];
}
```

如果要给 classmate 结构中的成员 address 结构中的 zip 赋值, 则可写成:

```
classmate.address.zip = 100800;
```

每个结构成员名从最外层直到最内层逐个被列出, 即嵌套式结构成员的表达方式是:

结构变量名.嵌套结构变量名.结构成员名

其中: 嵌套结构可以有多个, 结构成员名为最内层结构中不是结构的成员名。

#### 二、位结构

位结构是一种特殊的结构, 在需要按位访问一个字节或字的多个位时, 使用位结构比按使用位运算符更加方便。

下面先来看一下位结构定义的一般格式:

```
struct 位结构名{
    数据类型 变量名: 整型常数;
    数据类型 变量名: 整型常数;
} 位结构变量;
```

其中: 数据类型必须是 int(unsigned 或 signed), 整型常数必须是非负的整数, 范围是 0~15, 表示二进制位的个数, 即表示有多少位。

变量名是可选项, 可以省略而不命名, 上面这样规定是为了排列需要。

例如: 下面定义了一个位结构。

```
struct {
    unsigned icon: 8;
    unsigned txcolor: 4;
    unsigned bgcolor: 3;
```

```

    unsigned blink: 1;
} ch;

```

位结构成员的访问与结构成员的访问相同。

例如：访问上例位结构中的 bgcolor 成员可写成：

```
ch.bgcolor
```

为了便于理解，下图给出例子中位结构成员的内存分配情况。

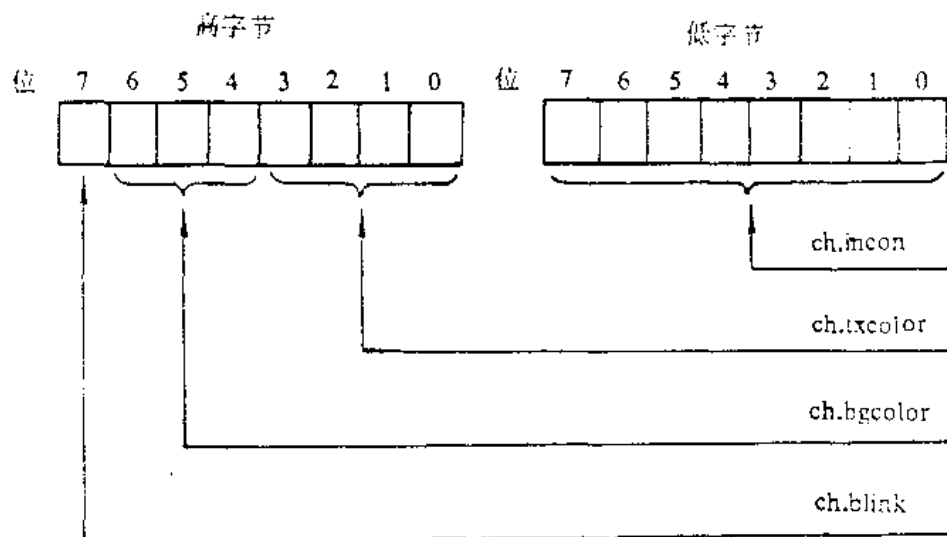


图 5-1 位结构成员内存分配情况

注：

- 位结构中的成员可以定义为 signed，也可以定义为 unsigned，但当成员长度为 1 时，会被认为是 unsigned 类型。因为单个位不可能具有符号。
- 位结构中的成员不能使用数组和指针，但位结构变量可以是数组和指针，如果是指针，其成员访问方式同结构指针。
- 位结构总长度(位数)，是各个位成员定义的位数之和，可以超过两个字节。
- 位结构成员可以与其它结构成员一起使用。

例如：

```

struct info {
    char name[8];
    int age;
    struct addr address;
    float pay;
    unsigned state: 1;
    unsigned pay: 1;
} worker;

```

上例的结构定义了关于一个工人的信息。其中有两个位结构成员，每个位结构成员只有一位，因此虽只各占一个字节但保存了两个信息，该字节中第一位表示工人的状态，第

二位表示工资是否已发放。由此可见使用位结构可以节省存储空间。

## 5.2 联合(union)

### 5.2.1 联合说明和联合变量定义

联合也是一种新的数据类型，它是一种特殊形式的变量。

联合说明和联合变量定义与结构十分相似。其格式为：

```
union 联合名 {  
    数据类型 成员名;  
    数据类型 成员名;  
    .  
    .  
    .  
} 联合变量名;
```

联合表示几个变量共用一个内存位置，在不同的时间保存不同的数据类型和不同长度的变量。

下例表示说明一个联合 i\_ch:

```
union i_ch {  
    int i;  
    char ch;  
};
```

再用已说明的联合可以定义联合变量。

例如用上面说明的联合定义一个名为 cnvt 的联合变量，可写成：

```
union i_ch cnvt;
```

在联合变量 cnvt 中，整型量 i 和字符 ch 共用同一内存位置。

下图表示 i 和 ch 占用共用地址的情况：

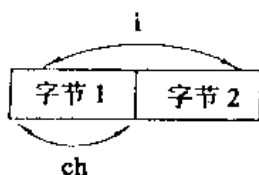


图 5-2 i 和 ch 占用共用地址情况

当一个联合被说明时，编译程序自动地产生一个变量，其长度为联合中最大的变量长度。

联合访问其成员的方法与结构相同。同样联合变量也可以定义成数组或指针，在定义为指针时，也要用“->”符号，此时联合访问成员可表示成：

联合名->成员名

另外，联合既可以出现在结构内，联合的成员也可以是结构。

例如:

```
struct {
    int age;
    char * addr;
    union {
        int i;
        char * ch;
    } x;
} y[10];
```

若要访问结构变量 y[1] 中联合 x 的成员 i, 可以写成:

y[1].x.i

若访问结构变量 y[2] 中联合 x 的字符串指针 ch 的第一个字符, 可写成

\* y[2].x.ch

若写成

\* y[2].x.\* ch

则是错误的。

### 5.2.2 结构和联合的区别

结构和联合有下列区别:

1. 结构和联合都是由多个不同数据类型的成员组成, 但在任何同一时刻, 联合中只存放了一个被选中的成员, 而结构的所有成员都存在。
2. 对联合中不同成员的赋值, 将会对其它成员重写, 成员原来的值就不存在了, 而对结构来说不同成员的赋值是互不影响的。

下面举一个例子来加深对联合的理解。

#### 例 5-2:

```
main( )
{
    union {          /* 定义一个联合 */
        int i;
        struct {     /* 在联合中定义一个结构 */
            char first;
            char second;
        } half;
    } number;
    number.i = 0x4241; /* 联合成员赋值 */
    printf("%c%c\n", number.half.first, number.half.second);
    number.half.first = 'a'; /* 联合中结构成员赋值 */
    number.half.second = 'b';
```

```

        printf("%x\n",number.i);
        getch( );
    }

```

输出结果为:

```

AB
6261

```

从本例的结果可以看出: 当给 `i` 赋值后, 其低八位和高八位的值也就是 `first` 和 `second` 的值; 当给 `first` 和 `second` 赋字符后, 这两个字符的 ASCII 码也将作为 `i` 的低八位和高八位的值。

### 5.3 枚举(enum)

枚举是一个被命名的整型常数的集合, 枚举在日常生活中很常见。

例如表示星期的 `SUNDAY`, `MONDAY`, `TUESDAY`, `WEDNESDAY`, `THURSDAY`, `FRIDAY`, `SATURDAY`, 就是一个枚举。

枚举的说明与结构和联合相似, 其格式为:

```

enum 枚举名 {
    标识符 [= 整型常数],
    标识符 [= 整型常数],
    .
    .
    .
    标识符 [= 整型常数]
} 枚举变量;

```

如果枚举没有初始化, 即省掉“= 整型常数”时, 则从第一个标识符开始, 顺次赋给标识符 `0, 1, 2, …`, 但当枚举中的某个成员赋值后, 其后的成员按依次加 1 的规则确定其值。

例如下列枚举说明后, `x1, x2, x3, x4` 的值分别为 `0, 1, 2, 3`。

```
enum test {x1,x2,x3,x4} x;
```

当定义改变成:

```
enum test {
    x1,
    x2 = 0,
    x3 = 50,
    x4,
} x;
```

则:

```
x1 = 0, x2 = 0, x3 = 50, x4 = 51
```

注:

- 枚举中每个成员(标识符)的结束符是",", 不是";", 最后一个成员可省略","。
- 初始化时可以赋予负数, 以后的标识符仍依次加 1。
- 枚举变量只能取枚举说明结构中的某个标识符常量。

例如:

```
enum test {
    x1 = 5,
    x2,
    x3,
    x4,
};
enum test x = x3;
```

此时, 枚举变量 x 实际上是 7。

## 5.4 类型说明

类型说明的格式为:

```
typedef 类型 定义名;
```

类型说明只定义了一个数据类型的新名字而不是定义一种新的数据类型。这里的类型是 Turbo C 许可的任何一种数据类型。定义名表示这个类型的新名字。

例如: 用下面语句定义整型数的新名字。

```
typedef int SIGNED_INT;
```

使用了这个说明后, SIGNED\_INT 就成为 int 的同义词了, 此时可以用 SIGNED\_INT 来定义整型变量。

例如: SIGNED\_INT i, j;

但是,

```
long SIGNED_INT L;
unsigned SIGNED_INT J;
```

却都是非法的。

typedef 同样可以用来说明结构、联合以及枚举。

说明一个结构的格式为:

```
typedef struct {
    数据类型 成员名;
    数据类型 成员名;
    .
    .
    .
} 结构名;
```

此时可以直接用结构名定义结构变量了。例如:

```
typedef struct {
    char name[8];
    int class;
    char subclass[6];
    float math,phys,chem,engl,biol;
} student;
student Tianyr;
```

则 Tianyr 被定义为结构数组和结构指针。

在第二章讲过的文件操作中，用到的 FILE 就是一个已被说明的结构，其说明如下：

```
typedef struct {
    short level;
    unsigned flags;
    char fd;
    unsigned char hold;
    short bsize;
    unsigned char * buffer;
    unsigned char * curp;
    unsigned istemp;
    short token;
} FILE;
```

这个结构说明已包含在文件 stdio.h 中，用户只要直接用 FILE 定义文件指针变量就可以。事实上，引入类型说明的目的并非为了方便，而是为了便于程序的移植。

## 5.5 预处理指令

由 ANSI 的标准规定，预处理指令主要包括：

```
#define
#error
#if
#else
#elif
#endif
#ifdef
#ifndef
#undef
#line
#pragma
```

由上述指令可以看出，每个预处理指令均带有符号“#”。下面只介绍一些常用指令。

### 1. #define 指令

#define 指令是一个宏定义指令，定义的一般格式是：



**#define** 宏替换名字符串(或数值)

由**#define** 指令定义后, 在程序中每次遇到该宏替换名时就用所定义的字符串(或数值)替换它。

例如: 可用下面语句定义 TRUE 表示数值 1, FALSE 表示 0。

```
#define TURE 1
#define FALSE 0
```

一旦在源程序中使用了 TRUE 和 FALSE, 编译时会自动的用 1 和 0 值代替。

注:

- 在宏定义语句后没有";"。
- 在 Turbo C 程序中习惯上用大写字符作为宏替换名, 而且常放在程序开头。
- 宏定义还有一个特点, 就是宏替换名可以带有形式参数, 在程序中用到时, 实际参数会替换这些形式参数。

例如:

```
#define MAX(x,y) (x>y)?x:y
main( )
{
    int i=10,j=15;
    printf("The Maximum is %d",MAX(i,j));
}
```

例中宏定义语句的含义是用宏替换名 MAX(x,y)代替 x,y 中较大者, 同样也可定义:

```
#define MIN(x,y) (x<y)?x:y
```

表示用宏替换名 MIN(x,y)代替 x,y 中较小者。

## 2. **#error** 指令

该指令用于程序的调试, 当编译中遇到**#error** 指令就停止编译。其一般格式为:

**#error** 出错信息

出错信息不加引号, 当编译器遇到这个指令时, 显示下面信息并停止编译。

Fatal: filename lineno error directive: 出错信息

## 3. **#include** 指令

**#include** 指令的作用是指示编译器将该指令所指出的另一个源文件嵌入**#include** 指令所在的程序中, 文件名应使用双引号或尖括号括起来。Turbo C 库函数的头文件一般用**#include** 指令在程序开头说明。

例如:

```
#include <sbo>
```

C 库函数的头文件一般用**#include** 指令在程序开头说明。

例如:

```
#include <stdio.h>    #include "io.h"
```

程序也允许嵌入其它文件，例如：

```
main( )
{
    #include <one.c>
}
```

其中 one.c 为另一个文件，内容为

```
printf("This is from the include file");
```

上例编译时将按集成开发环境的 Options / Directories / Include directories 中指定的包含文件路径查找被嵌入文件。

#### 4. #if、#else、#endif 指令

#if、#else 和 #endif 指令为条件编译指令，它的一般格式为：

```
#if 常数表达式
    语句段;
#else
    语句段;
#endif
```

上述结构的含义是：若 #if 指令后的常数表达式为真，则编译 #if 到 #else 之间的程序段；否则编译 #else 到 #endif 之间的程序段。

例如：

```
#define MAX 1000
main( )
{
    #if MAX > 999
        printf("compiled for bigger\n");
    #else
        printf("compiled for smaller\n");
    #endif
}
```

#### 5. #undef 指令

#undef 指令用来删除此前定义的宏定义，其一般格式为：

```
#undef 宏替换名
```

例如：

```
#define TURE 1
.....
#undef TURE
```

#undef 主要用来使宏替换名只限定在需要使用它们的程序段中。

## 第六章 函 数

结构化程序设计是目前流行的程序设计方法, 因为结构化程序是将一个大型的程序分为若干个具有不同功能的子模块, 这样便于软件的开发和维护。Turbo C 是典型的结构式语言, 它提供的运行程序库含有 400 多个函数, 每个函数都完成一定的功能, 可由用户随意调用。这些函数总的可分为输入输出函数、数学函数、字符串和内存函数、与 BIOS 和 DOS 有关的函数、字符屏幕和图形功能函数、过程控制函数、目录函数等。对这些库函数应熟悉其功能, 只有这样才可省去很多不必要的重复劳动。

用户编制 Turbo C 语言源程序, 就是使用 Turbo C 的库函数。可以把所有使用的库函数放在一个庞大的主函数里, 也可以按不同功能设计成一个个用户函数而被其它函数调用。Turbo C 建议用户采用后者, 当用户编制了一些较常用的函数时, 只要将其保存在函数库里, 在以后的编程中可被方便地调用而不需要再去编译它们, 链接时将会自动从相应的库中调出装配成所需程序。

本章主要介绍有关函数的说明和定义、函数的调用以及函数的返回等内容。

### 6.1 函数的说明和定义

Turbo C 的所有函数与变量一样, 在使用之前必须先予以说明。所谓说明是指说明函数是什么类型的函数。一般, 库函数的说明都包含在相应的头文件 `<*.h>` 中, 例如标准输入输出函数包含在 `stdio.h` 中, 非标准输入输出函数包含在 `io.h` 中, 以后在使用库函数时必须先知道该函数包含在什么样的头文件中, 在程序的开头用 `#include <*.h>` 或 `#include "*.h"` 说明。只有这样, 程序在编译、链接时 Turbo C 才知道它是提供的库函数。否则将认为是用户自己编写的函数而不能装配 (关于 `#include` 指令第五章已作过介绍)。

#### 6.1.1 函数说明

有两种方式的函数说明:

##### 1. 经典方式

其格式为:

函数类型 函数名( );

##### 2. ANSI 规定方式

其格式为:

函数类型 函数名(数据类型 形式参数, 数据类型 形式参数, ...);

其中: 函数类型是该函数返回值的数据类型, 可以是以前介绍的整型(int)、长整型(long)、字符型(char)、单浮点型(float)、双浮点型(double)以及无值型(void), 也可以是指针, 包括结构指针。无值型表示函数没有返回值。

函数名是 Turbo C 的标识符, 小括号中的内容为该函数的形式参数说明。可以只有

数据类型而没有形式参数，也可以两者都有。经典方式的函数说明没有参数信息。

下面给出一些函数说明的例子。

```
int putthz(intx,inty,intz,int color,char *p); /* 说明一个整型函数 */
char *name(void); /* 说明一个字符串指针函数 */
void student(int n,char *str); /* 说明一个不返回值的函数 */
float calculate( ); /* 说明一个浮点型函数 */
```

注:

● 如果一个函数没有说明就被调用，编译程序并不认为出错，而将此函数默认为整型(int)函数。因此当一个函数返回其它类型，又没有事先说明，编译时将会出错。

### 6.1.2 函数定义

函数定义就是确定该函数完成什么功能以及怎样运行，相当于其它编程语言的子程序。Turbo C 对函数的定义采用 ANSI 规定方式。即:

```
函数类型 函数名(数据类型 形式参数, 数据类型 形式参数, ...)
{
    函数体;
}
```

其中函数类型和形式参数的数据类型为 Turbo C 的基本数据类型。函数体为 Turbo C 提供的库函数和语句以及其它用户自定义函数调用语句的组合，并包含在一对花括号“{”和“}”中。

需要指出的是一个程序必须有一个主函数，其它的用户定义子函数则可以是任意多个，这些函数的位置也没有什么限制，可以在 main( ) 函数前，也可以在其后。Turbo C 将所有函数都认为是全局性的，而且是外部的，即可以被另一个文件中的任何一个函数调用。

## 6.2 函数的调用

### 6.2.1 函数的简单调用

上一节讲了函数的说明和定义，怎样使用函数就属于函数的调用问题。不象 FORTRAN 语言中调用子程序用 call 语句，Turbo C 调用函数时直接使用函数名和实参的方法，也就是将要赋给被调用函数的参量，按该函数说明的参数形式传递过去，然后进入子函数运行，运行结束后再按子函数规定的数据类型返回一个值给调用函数。使用 Turbo C 的库函数就是函数简单调用的方法。

下面用一个例子来介绍如何调用用户自己编写的子函数。

例 6-1:

```
#include <stdio.h>
int maximum(int, int, int); /* 说明一个用户自定义函数 maximum( ) */
main( )
{
```

```

    int i,j,k;
    printf("i,j,k=?\n");
    scanf("%4d,%4d,%4d",&i,&j,&k);    /* 从键盘接收 3 个整数 */
    maximum(i,j,k);    /* 调用子函数 */
    getch( );    /* 等待按任一键退出 */
}

maximum(int l, int m, int n)    /* 定义子函数 maximum( ) */
{
    int max;
    max=l>m?l:m;    /* 求出 l 和 m 中较大者送 max */
    max=max>n?max:n;    /* 求出 max 和 n 中较大者反送 max */
    printf("The maximum value is %d\n",max);
}

```

### 6.2.2 函数的参数传递

函数的参数传递包括以下几个方面，下面分别进行讨论。

#### 一、调用函数向被调用函数以形式参数传递

用户编写的函数一般在对其说明和定义时就规定了形式参数类型，因此调用这些函数时参量必须与子函数中形式参数的数据类型、顺序和个数完全相同，否则在调用中将会出错，得到意想不到的结果。

例 6-1 就是按形式参数传递变量的例子，可供参考。

注：

- 当数组作为形式参数向被调用函数传递时，只传递数组的地址，而不是将整个数组元素都复制到函数中去，亦即用数组名作为实参调用子函数，调用时指向该数组第一个元素的指针就被传递给子函数，因为在 Turbo C 中，没有下标的数组名就是一个指向该数组第一个元素的指针。当然，两个函数中数组变量的类型必须相同。

用下述方法传递数组形参。

例 6-2：

```

#include<stdio.h>
void disp( );
main( )
{
    int m[10],i;
    for(i=0;i<10;i++)
        m[i]=i;
    disp(m);    /* 按指针方式传递数组 */
    getch( );
}
void disp(int *n)
{

```

```

int j;
for(j=0;j<10;j++)
    printf("%3d",*(n++));
}

```

另外，在传递数组的某个元素时，数组元素作为实参，此时按使用其它简单变量的方法使用数组元素。例 6-2 按传递数组元素的方法传递时变为：

```

#include <stdio.h>
void disp( );
main( )
{
    int m[10],i;
    for(i=0;i<10;i++)
    {
        m[i]=i;
        disp(m[i]);    /* 逐个传递数组元素 */
    }
    getch( );
}
void disp(int n)
{
    printf("%3d",n);
}

```

这时一次只传递了数组的一个元素。

## 二、被调用函数向调用函数返回值

一般使用 return 语句由被调用函数向调用函数返回值，该语句有下列用途：

1. 它能立即从所在的函数中退出，返回到调用它的程序中去。
2. 返回一个值给调用它的函数。

有两种方法可以终止子函数的运行并返回到调用它的函数中：一是执行到函数的最后一条语句后返回；一是执行到语句 return 时返回。前者当子函数执行完后仅返回给调用函数一个 0。若要返回一个值，就必须用 return 语句。只需在 return 语句中指定返回的值即可。例 6-1 返回最大值时变为：

### 例 6-3:

```

#include <stdio.h>
int maximum(int, int, int);
main( )
{
    int i,j,k,max;
    printf("i,j,k=?\n");
    scanf("%4d,%4d,%4d",&i,&j,&k);
    max=maximum(i,j,k);    /* 调用子函数，并将返回值赋给 max */
}

```

```

        printf("The maximum value is %d\n",max);
    getch( );
}

maximum(int l,int m,int n)
{
    int max;
    max = l > m ? l : m;    /* 求最大值 */
    max = max > n ? max : n;
    return(max);    /* 返回最大值 */
}

```

return 语句可以向调用函数返回值，但这种方法只能返回一个参数，而在许多情况下却要返回多个参数，这时用 return 语句就不能满足要求。Turbo C 提供了另一种参数传递的方法，就是调用函数向被调用函数传递的形式参数不是传递变量本身，而是传递变量的地址，在子函数运行中向相应的地址写入不同的数值之后，也就改变了调用函数中相应变量的值，从而达到了返回多个变量的目的。下面举例说明。

#### 例 6-4:

```

#include <stdio.h>
void subfun(int *, int *);    /* 说明了子函数 */
main( )
{
    int i,j;
    printf("i,j=? ");
    scanf("%d,%d",&i,&j);    /* 从键盘输入 2 个整数 */
    printf("In main before calling\n")    /* 输出这 2 个数及乘积 */
        "i=%-4d j=%-4d i*j=%-4d\n",i,j,i*j);
    subfun(&i,&j);    /* 以传送地址的方式调用子函数 */
    printf("In main after calling\n")    /* 调用子函数后输出变量值 */
        "i=%-4d j=%-4d i*j=%-4d\n",i,j,i*j);
    getch( );
}

void subfun(int *i, int *j)
{
    *i = *i + 2;
    *j = *i - *j;
    printf("In subfun after calling\n")    /* 子函数中输出变量值 */
        "i=%-4d j=%-4d i*j=%-4d\n",*i,*j,*i*j);
}

```

上例中，\*i \* \*j 表示指针 i 和 j 所指的两个整型数 \*i 和 \*j 之乘积。

另外，return 语句也可以返回一个指针，现举例说明之。

下列例程先等待输入一个字符串，再等待输入要查找的字符，然后调用 match 函数

在字符串中查找该字符。若有相同字符，则返回一个指向该字符串中这一位置的指针，如果没有找到，则返回一个空(NULL)指针。

#### 例 6-5

```
#include <stdio.h>
char * match(char, char *); /* 说明一个子函数 */
main( )
{
    char s[40], c, * str;
    str = malloc(40);        /* 为字符串指针分配内存空间 */
    printf("Please input character string: ");
    gets(s);                 /* 键盘输入字符串 */
    printf("Please input one character: ");
    c = getch( );            /* 键盘输入字符 */
    str = match(c, s);        /* 调用了函数 */
    putchar('\n');           /* 换行 */
    puts(str);                /* 输出子函数返回的指针所指向的字符串 */
    getch( );
}
char * match(char c, char * s)
{
    int i = 0;
    while(c != s[i] && s[i] != '\0') /* 在字符串中查找指定的字符 */
        i++;
    return(&s[i]);             /* 返回所找字符的地址 */
}
```

#### 三、用全程变量实现参数互传

以上两种办法可以在调用函数和被调用函数间传递参数，但使用都不太方便。如果将所要传递的参数定义为全程变量，可使这种变量在整个程序中对所有函数都是可见的，这也相当于在调用函数和被调用函数之间实现了参数的传递和返回。这也是实践中经常使用的方法。但定义全程变量势必长久地占用了内存。因此，全程变量的数目受到限制，特别对于较大的数组更是如此。当然，对于绝大多数程序来说内存都是够用的。下面看一个使用全程变量传递参数的例子。

#### 例 6-6:

```
#include <stdio.h>
void disp( );
int m[10]; /* 定义全程变量 */
main( )
{
    int i;
    printf("In main before calling\n");
```



```

    for(i=0;i<10;i++)
    {
        m[i]=i;
        printf("%3d",m[i]);    /* 输出调用子函数前数组的值 */
    }
    disp( );    /* 调用子函数 */
    printf("\n\n main after calling\n");
    for(i=0;i<10;i++)
        printf("%3d",m[i]);    /* 输出调用子函数后数组的值 */
    getch( );
}
void disp( )
{
    int j;
    printf("\n\n subfunc after calling\n");
    for(j=0;j<10;j++)
    {
        m[j]=m[j]*10;
        printf("%3d",m[j]);    /* 子函数中输出数组的值 */
    }
}

```

#### 四、主函数的命令行参数

可以通过参数传递的方法在调用函数和被调用函数之间建立联系，对于主函数同样也可以用规定形式参数的方法来传递命令行参数。所谓命令行参数是向主函数传递的参数，它是 DOS 状态下运行程序时在文件名后直接输入主函数所要求的参数。Turbo C 规定主函数的形式参数只能用以下格式：

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

其中，整型数 `argc` 表示命令行中输入参数的个数(包括程序名)，它至少是 1，因为程序名就是一个参数；`argv[ ]` 是指向各个输入参数字符串的指针，这个数组里的每个元素都指向相应的命令行参数，所有的命令行参数都是字符串，任何数字都必须由程序变成适当的格式。下面的例程说明了使用命令行参数的用法。如果在程序名后直接键入一个字符串和一个数字，则程序的运行结果为连续重复显示数字所规定次数的“Hello,<字符串>”。

#### 例 6-7:

```

main(int argc,char *argv[ ])    /* 定义含命令行参数的主函数 */
{
    int inti;    /* 定义局部变量 */
    if(argc!=3)    /* 判断是否 3 个命令行参数 */
    {
        puts("You forget type your name"); /* 不是 3 个显示提示信息 */
        getch( );    /* 等待按任一键 */
    }
}

```

```

        exit(0);          /* 退出 */
    }
    for(i=0;i<atoi(argv[2]);i++)    /* 控制下列语句的执行次数 */
        printf("Hello,%s\n",argv[1]); /* 是3个时输出第二个参数 */
}

```

如果命名该程序为 name.c, 运行这个程序时键入 name Mr.Wang. 3, 则程序输出

```

Hello,Mr.Wang.
Hello,Mr.Wang.
Hello,Mr.Wang.

```

注:

- 命令行参数都必须用空格(space)或制表符(Tab)分隔, 逗号和分号都不认为是分隔符。
- 程序的名字总是作为第一个字符串 \* argv[0], 然后顺次是 \* argv[1], \* argv[2], ...
- 所有的命令行参数都被认为是字符串。

### 6.2.3 函数的递归调用

Turbo C 允许函数自己调用自己, 即函数的递归调用。递归调用可以使程序简洁、代码紧凑, 但要牺牲一些内存空间用作处理时的堆栈。

如要计算一个  $n!$  ( $n$  的阶乘) 的值可用下面递归调用。

例 6-8:

```

unsigned long mul(int n);
main( )
{
    int m;
    puts("Calculate n! n=?");
    scanf("%d",&m);    /* 键盘输入数据 */
    printf("%d!= %ld\n",m,mul(m)); /* 调用子程序计算并输出 */
    getch( );
}
unsigned long mul(int n)
{
    unsigned long p;
    if(n>1)
        p=mul(n-1)*n;    /* 递归调用计算n! */
    else
        p=1L;
    return(p);           /* 返回结果 */
}

```

运行结果:

```
calculate  n!  n=?
```

输入 5 时结果为

```
5! = 120
```

### 6.3 函数作用范围

Turbo C 中每个函数都是独立的代码块, 函数的代码归该函数所有, 除了对函数的调用以外, 其它任何函数中的任何语句都不能访问它。例如使用跳转语句 `goto` 就不能从一个函数跳进其它函数内部。除非使用全程变量, 否则一个函数内部定义的程序代码和数据不与另一个函数内的程序代码和数据不会相互影响。

Turbo C 中所有函数的作用域都处于同一嵌套层次, 即不能在一个函数内再说明或定义另一个函数。

Turbo C 中一个函数对其它子函数的调用是全程的, 即使函数在不同的文件中, 也不必附加任何说明语句而被另一个函数调用, 也就是说一个函数对于整个程序都是可见的。

## 第七章 字符屏幕和图形函数

象 BASIC 语言一样, Turbo C 提供了许多关于屏幕输出的函数, 主要分为文本字符屏幕函数和图形功能函数。本章详细介绍这两类函数的用法。

### 7.1 字符屏幕函数

Turbo C 的字符屏幕函数主要包括文本窗口大小的设定、窗口颜色的设置、窗口文本的清除和输入输出等函数。

#### 7.1.1 文本窗口的定义

Turbo C 默认定义的文本窗口为整个屏幕, 共有 80 列(或 40 列)25 行的文本单元, 每个单元包括一个字符和一个属性, 字符即 ASCII 码字符, 属性规定该字符的颜色和亮度。

Turbo C 可以使用 window( ) 函数定义屏幕上的一个矩形区域作为窗口。窗口定义之后, 用有关窗口的输入输出函数就可以只在此窗口内进行操作而不越出窗口的边界。

window( ) 函数的调用格式为:

```
void window(x1,y1,x2,y2);
```

该函数的原型在 conio.h 中(关于文本窗口的所有函数其头文件均为 conio.h, 以后不再说明)。函数中形式参数(x1,y1)是窗口左上角的坐标, (x2,y2)是窗口右下角的坐标, 其中: (x1,y1)和(x2,y2)是相对于整个屏幕而言的。Turbo C 规定整个屏幕的左上角坐标为(1,1), 右下角坐标为(80,25), 并规定沿水平方向为 X 轴, 正向朝右; 沿垂直方向为 Y 轴, 正向朝下。若 window( ) 函数中的坐标超过了屏幕坐标的界限, 则窗口的定义就失去了意义, 也就是说定义将不起作用, 但程序编译链接时并不出错。

另外, 一个屏幕可以定义多个窗口, 但现行窗口只能有一个(因为 DOS 为单任务操作系统), 当需要用另一窗口时, 可将定义该窗口的 window( ) 函数再调用一次, 此时该窗口便成为现行窗口了。

如要定义一个窗口的左上角在屏幕(20,5)处, 大小为 30 列 15 行, 则可写成:

```
window(20,5,50,20);
```

#### 7.1.2 文本窗口颜色的设置

文本窗口颜色的设置包括背景颜色的设置和字符颜色的设置, 使用的函数及其调用格式为:

```
设置背景颜色:    void textbackground(int color);  
设置字符颜色:    void textcolor(int color);
```

有关颜色的定义见表 7-1。

表 7-1 有关颜色的定义

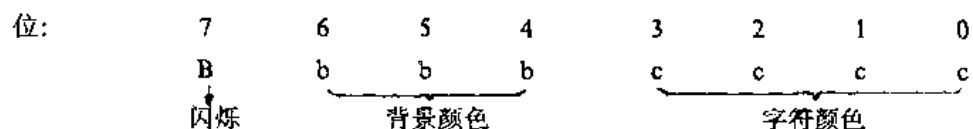
符号常数	数值	含 义	字符或背景
BLACK	0	黑	两者均可
BLUE	1	蓝	两者均可
GREEN	2	绿	两者均可
CYAN	3	青	两者均可
RED	4	红	两者均可
MAGENTA	5	洋红	两者均可
BROWN	6	棕	两者均可
LIGHTGRAY	7	淡灰	两者均可
DARKGRAY	8	深灰	只用于字符
LIGHTBLUE	9	淡蓝	只用于字符
LIGHTGREEN	10	淡绿	只用于字符
LIGHTCYAN	11	淡青	只用于字符
LIGHTRED	12	淡红	只用于字符
LIGHTMAGENTA	13	淡洋红	只用于字符
YELLOW	14	黄	只用于字符
WHITE	15	白	只用于字符
BLINK	128	闪烁	只用于字符

上表中的符号常数与相应的数值等价，两者可以互换。例如设定蓝色背景，可以使用 `textbackground(1)`，也可以使用 `textbackground(BLUE)`，两者没有任何区别，只不过后者比较容易记忆，一看就知道是蓝色。

Turbo C 另外还提供了一个函数，可以同时设置文本的字符和背景的颜色，这个函数的调用格式为：

```
void textattr(int attr);
```

其中：attr 的值表示颜色形式编码的信息，每一位代表的含义如下：



字节的低四位 cccc 设置字符颜色(0 到 15)，4~6 三位 bbb 设置背景颜色(0 到 7)，第 7 位 B 设置字符是否闪烁。假如要设置蓝底黄字，定义方法如下：

```
textattr(YELLOW+(BLUE<<4));
```

若再要求字符闪烁，则定义变为：

```
textattr(128+YELLOW+(BLUE<<4)); 或
textattr(BLINK YELLOW BLUE);
```

注：

● 由于背景只有 0 到 7 共八种颜色，若取大于 7 小于 15 的数，则代表的颜色与减 8 后的值所对应的颜色相同。

● 用 `textbackground()` 和 `textcolor()` 函数设置了窗口的背景与字符颜色后, 在没有用 `clrscr()` 函数清除窗口之前, 颜色不会改变, 直到使用了函数 `clrscr()`, 整个窗口和随后输出到窗口中的文本字符才会变成新的颜色。

下面的例程使用了关于窗口大小的定义、颜色的设置等函数。

#### 例 7-1:

```
#include <conio.h>
main( )
{
    int i;
    textbackground(0);    /* 设置屏幕背景色 */
    clrscr( );            /* 清除文本屏幕 */
    for(i=1;i<8;i++)
    {
        window(10+i*5,5+i,30+i*5,15+i); /* 定义文本窗口 */
        textbackground(i); /* 定义窗口背景色 */
        clrscr( );        /* 清除窗口 */
    }
    getch( );
}
```

该程序在一帧屏幕的不同位置定义了 7 个窗口, 其背景色分别使用了 7 种不同的颜色。

### 7.1.3. 窗口内文本的输入输出函数

#### 一、窗口内文本的输出函数

```
int cprintf("<格式化字符串>", <变量表>);
int cputs(char *string);
int putch(int ch);
```

`cprintf()` 函数输出一个格式化的字符串或数值到窗口中。它与 `printf()` 函数的用法完全一样, 区别在于 `cprintf()` 函数的输出受窗口限制, 而 `printf()` 函数的输出为整个屏幕。

`cputs()` 函数输出一个字符串到屏幕上, 它与 `puts()` 函数用法完全一样, 只是受窗口大小的限制。

`putch()` 函数输出一个字符到窗口内。

注:

● 使用以上几种函数, 当输出越出窗口的右边界时会自动转到下一行的开始处继续输出。当窗口内填满内容仍没有结束输出时, 窗口屏幕将会自动逐行上卷直到输出结束为止。

#### 二、窗口内文本的输入函数

```
int getche(void);
```

该函数在第三章中已经讲过,这里需要说明的是, `getche()` 函数从键盘上获得一个字符,在屏幕上显示的时候,如果字符越出了窗口右边界,则会被自动转移到下一行的开始位置。

下面的程序给例 7-1 中加入了一些文本的输出函数。

#### 例 7-2:

```
#include <conio.h>
main( )
{
    int i;
    char *c[]={"BLACK","BLUE","GREEN","CYAN","RED","MAGENTA",
               "BROWN","LIGHTGRAY"}; /* 定义数组指针并初始化 */
    textbackground(0); /* 设置屏幕背景色 */
    clrscr( ); /* 清除屏幕 */
    printf("%s",c[0]); /* 输出字符串 */
    for(i=1;i<8;i++) /* 设置 7 次循环 */
    {
        window(10+i*5,5+i,30+i*5,15+i); /* 定义窗口 */
        textbackground(i); /* 设置窗口背景色 */
        clrscr( ); /* 清除窗口 */
        textcolor(7+i); /* 设置字符颜色 */
        cputs(c[i]); /* 向当前窗口输出字符串 */
    }
    getch( );
}
```

#### 7.1.4 有关屏幕操作的函数

`void clrscr(void);` 清除当前窗口中的文本内容,并把光标定位在窗口的左上角(1,1)处。

`void clreol(void);` 清除当前窗口中从光标位置到行尾的所有字符,光标位置不变。

`void gotoxy(x,y);` 该函数很有用处,它用来将光标定位在当前窗口中的位置。这里  $x, y$  是指光标要定位处的坐标(相对于窗口而言),当  $x, y$  越出了窗口的范围时,该函数就不起作用了。

```
int gettext(int x1,int y1,int x2,int y2,void *buffer);
```

```
int puttext(int x1,int y1,int x2,int y2,void *buffer);
```

`gettext()` 函数是将屏幕上指定的矩形区域内文本内容存入 `buffer` 指针指向的一个内存空间。内存的大小用下式计算:

所用字节大小 = 行数 \* 列数 \* 2

其中:

行数 =  $y2 - y1 + 1$       列数 =  $x2 - x1 + 1$

puttext( )函数则是将 gettext( )函数存入内存 buffer 中的文字内容拷贝到屏幕上指定的位置。

```
int movetext(int x1,int y1,int x2,int y2,int x3,int y3);
```

movetext( )函数将屏幕上左上角为(x1,y1), 右下角为(x2,y2)的一个矩形窗口内的文本内容拷贝到左上角为(x3,y3)的新的位置。该函数的坐标也是相对于整个屏幕而言的。

注:

- gettext( )函数和 puttext( )函数中的坐标是相对整个屏幕而言的, 即是屏幕的绝对坐标, 而不是相对窗口的坐标。

- movetext( )函数是拷贝而不是移动窗口区域内容, 也就是使用该函数后, 原来位置区域里的文本内容仍然存在。

### 例 7-3:

```
#include <conio.h>
main( )
{
    int i;
    char *f[] = {" Load      F3 ",
                 " Pick  Alt-F3 ",
                 " New          ",
                 " Save      F2 ",
                 " Write to    ",
                 " Directory   ",
                 " Change dir  ",
                 " Os shell    ",
                 " Quit  Alt-X  "};
    char buf[9 * 14 * 2];      /* 定义字符型数组 */
    clrscr( );
    textcolor(YELLOW);
    textbackground(BLUE);
    clrscr( );
    gettext(10,2,24,11,buf);  /* 保存文本内容于数组中 */
    window(10,2,24,11);
    textbackground(RED);
    textcolor(YELLOW);
    clrscr( );
    for(i=0;i<9;i++)
    {
        gotoxy(1,i+1);        /* 定位光标 */
        cprintf("%s",f[i]);    /* 在当前窗口的光标位置输出字符串 */
    }
}
```



```

    }
    getch( );
    movetext(10,2,24,11,40,10); /* 拷贝屏幕上一个矩形区内容到另一位置 */
    puttext(10,2,24,11,buf);    /* 在指定位置释放内存中保存的文本内容 */
    getch( );
}

```

下面再介绍一些函数:

```

void highvideo(void);    设置显示器高亮度显示字符。
void lowvideo(void);     设置显示器低亮度显示字符。
void normvideo(void);    使显示器恢复到程序运行前的显示方式。
int wherex(void);        这两个函数返回当前窗口下光标的x,y坐标。
int wherey(void);

```

## 7.2 图形函数

Turbo C 提供了非常丰富的图形函数,所有图形函数的原型均在 `graphics.h` 中,本节主要介绍图形模式的初始化、独立图形程序的建立、基本图形功能、图形窗口以及图形模式下的文本输出等函数。另外,使用图形函数时要确保有显示器图形驱动程序 `*.BGI`,同时将集成开发环境 Options/Linker 中的 Graphics lib 选为 on,只有这样才能保证正确使用图形函数。

### 7.2.1 图形模式的初始化

不同的显示器适配器有不同的图形分辨率。即使是同一种显示器适配器,在不同的模式下也有不同的分辨率。因此,在屏幕作图之前,必须根据显示器适配器的种类将显示器设置成为某种图形模式。在未设置图形模式之前,微机系统默认屏幕为文本模式(80列、25行字符模式),此时所有的图形函数均不能工作。如要设置屏幕为图形模式,可用下列图形初始化函数:

```
void far initgraph(int far *gdriver,int far *gmode,char *path);
```

其中 `gdriver` 和 `gmode` 分别表示图形驱动器和模式, `path` 是指图形驱动程序所在的目录路径。有关图形驱动器、图形模式的符号常数及对应的分辨率见表 7-2。

图形驱动程序由 Turbo C 出版商提供,文件扩展名为 `.BGI`。对应不同的图形适配器有不同的图形驱动程序。例如对于 EGA、VGA 图形适配器,就应该调用驱动程序 `EGAVGA.BGI`。

**例 7-4:** 使用图形初始化函数设置 VGA 高分辨率图形模式。

```

#include <graphics.h>
main( )
{
    int gdriver,gmode;
    gdriver = VGA;                /* VGA 图形驱动器 */
}

```

```

gmode=VGAHI;                      /* 选 VGA 高分辨率图形模式 */
initgraph(&gdriver,&gmode,"c:\\tc"); /* 图形初始化 */
bar3d(200,200,400,350,50,1);      /* 画一长方体 */
getch( );
closegraph( );                     /* 关闭图形模式 */
}

```

表 7-2 图形驱动器、模式的符号常数及数值

图形驱动器(gdriver)		图形模式(gmode)		色调	分辨率
符号常数	数值	符号常数	数值		
CGA	1	CGAC0	0	C0	320×200
		CGAC1	1	C1	320×200
		CGAC2	2	C2	320×200
		CGAC3	3	C3	320×200
		CGAHI	4	2 色	640×200
MCGA	2	MCGAC0	0	C0	320×200
		MCGAC1	1	C1	320×200
		MCGAC2	2	C2	320×200
		MCGAC3	3	C3	320×200
		MCGAMED	4	2 色	640×200
		MCGAHI	5	2 色	640×480
EGA	3	EGALO	0	16 色	640×200
		EGAHI	1	16 色	640×350
EGA64	4	EGA64LO	0	16 色	640×200
		EGA64HI	1	4 色	640×350
EGAMON	5	EGAMONHI	0	2 色	640×350
HERC	7	HERCMONHI	0	2 色	720×348
ATT400	8	ATT400C0	0	C0	320×200
		ATT400C1	1	C1	320×200
		ATT400C2	2	C2	320×200
		ATT400C3	3	C3	320×200
		ATT400MED	4	2 色	640×200
		ATT400HI	5	2 色	640×400
VGA	9	VGALO	0	16 色	640×200
		VGAMED	1	16 色	640×350
		VGAHI	2	16 色	640×480
PC3270	10	PC3270HI	0	2 色	720×350
IBM8514	6	IBM8514LO	1	256 色	640×480
		IBM8514HI	2	256 色	1024×768
DETECT	0	用于硬件测试			

有时编程者并不知道所用的图形显示器适配器的种类，或者需要将编写的程序用于多种不同图形驱动器，Turbo C 提供了一个自动检测显示器硬件的函数，其调用格式为：

```
void far detectgraph(int *gdriver, *gmode);
```

其中 gdriver 和 gmode 的意义与图 7-2 中相同。

**例 7-5:** 自动进行硬件测试后的图形初始化程序。

```
#include <graphics.h>
main( )
{
    int gdriver,gmode;
    detectgraph(&gdriver,&gmode);          /* 自动测试硬件 */
    printf("Detect graphics driver is %d,mode is %d\n",
           gdriver,gmode);                /* 输出测试结果 */
    getch( );
    initgraph(&gdriver,&gmode,"c:\\tc"); /* 根据测试结果进行图形初始化 */
    bar3d(50,50,250,150,20,1);
    getch( );
    closegraph( );
}
```

本例程序中先对图形显示器自动检测，然后再用图形初始化函数进行初始化设置。但 Turbo C 提供了一种更为简单的方法，即在 `gdriver=DETECT` 语句后再调用 `initgraph( )` 函数就行了。采用这种方法后，例 7-5 将变成：

**例 7-6:**

```
#include <graphics.h>
main( )
{
    int gdriver=DETECT,gmode;
    initgraph(&gdriver,&gmode,"c:\\tc");
    bar3d(50,50,250,150,20,1);
    getch( );
    closegraph( );
}
```

另外，Turbo C 还提供了退出图形状态的函数 `closegraph( )`，其调用格式为：

```
void far closegraph(void);
```

调用该函数后可退出图形状态而进入文本方式(Turbo C 的默认方式)，并释放用于保存图形驱动程序和字体的系统内存。

### 7.2.2 独立图形运行程序的建立

Turbo C 对用 `initgraph( )` 函数直接进行图形初始化的程序，在编译和链接时并没有将相应的驱动程序(\*.BGI)装入到执行程序，当程序运行到 `initgraph( )` 语句时，再从该函数中第三个形式参数 `char * path` 中所规定的路径中去寻找相应的驱动程序。若没有找到，则在 C:\TC 中去找，如 C:\TC 中仍没有或 TC 不存在，将会报告下列错误：

```
BGI Error: Graphics not initialized (use 'initgraph')
```

因此，为了使用方便，应该建立一个不需要驱动程序就能独立运行的可执行图形程序，Turbo C 中规定用下述步骤(这里以 EGA、VGA 显示器适配器为例)：

1. 在 C:\TC 子目录下输入命令：BGIOBJ EGAVGA

此命令将驱动程序 EGAVGA.BGI 转换成目标文件 EGAVGA.OBJ。

2. 在 C:\TC 子目录下输入命令：TLIB LIB\GRAPHICS.LIB+EGAVGA

此命令的意思是将 EGAVGA.OBJ 的目标模块装入库文件 GRAPHICS.LIB 中。

3. 在程序中调用 initgraph( )函数之前加上一句：

```
registerbgidriver(EGAVGA__driver);
```

该函数告诉链接程序在链接时把 EGAVGA 驱动程序装入到用户的执行程序中。

经过上面处理，编译链接后的执行程序可在任何目录或其它兼容机上运行。假定已作了前两个步骤，若再向例 7-6 中加入 registerbgidriver( )函数则变成：

**例 7-7:**

```
#include <graphics.h>
main( )
{
    int gdriver=DETECT,gmode;
    registerbgidriver(EGAVGA__driver); /* 建立独立图形运行程序 */
    initgraph(&gdriver,&gmode,"c:\tc");
    bar3d(50,50,250,150,20,1);
    getch( );
    closegraph( );
}
```

本例的程序编译链接后产生的执行程序可以独立运行。

如不要初始化成 EGA 或 VGA 的分辨率，而想初始化为 CGA 的分辨率，则只需要将上述步骤中有 EGAVGA 的地方用 CGA 代替即可。

### 7.2.3 屏幕颜色的设置和清屏函数

对于图形模式的屏幕颜色设置，同样分为背景色的设置和前景色的设置。在 Turbo C 中分别用下面两个函数。

设置背景色：void far setbkcolor(int color);

设置作图色：void far setcolor(int color);

其中 color 为图形方式下颜色的规定数值，对于 EGA 和 VGA 显示器适配器，有关颜色的符号常数及数值见下表所示。

表 7-3 有关屏幕颜色的符号常数表

符号常数	数值	含义	符号常数	数值	含义
BLACK	0	黑色	DARKGRAY	8	深灰
BLUE	1	蓝	LIGHTBLUE	9	淡蓝
GREEN	2	绿	LIGHTGREEN	10	淡绿
CYAN	3	青	LIGHTCYAN	11	淡青
RED	4	红	LIGHTRED	12	淡红
MAGENTA	5	洋红	LIGHTMAGENTA	13	淡洋红
BROWN	6	棕	YELLOW	14	黄
LIGHTGRAY	7	淡灰	WHITE	15	白

对于 CGA 适配器,背景色可以为表 7-3 里 16 种颜色中的一种,但前景色则取决于不同的调色板。共有四种调色板,每种调色板上有四种颜色可供选择。不同调色板所对应的颜色见表 7-4。

表 7-4 CGA 调色板与颜色值表

调色板		颜色值			
符号常数	数值	0	1	2	3
C0	0	背景	绿	红	黄
C1	1	背景	青	洋红	白
C2	2	背景	淡绿	淡红	黄
C3	3	背景	淡青	淡洋红	白

清除图形屏幕内容使用清屏函数,其调用格式如下:

```
void far cleardevice(void);
```

有关颜色设置和清屏函数的使用请看例 7-8。

例 7-8:

```
#include <graphics.h>
main( )
{
    int gdriver,gmode,i;
    gdriver=DETECT;
    registerbgidriver(EGAVGA_driver); /* 建立独立图形运行程序 */
    initgraph(&gdriver,&gmode,"c:\\tc"); /* 图形初始化 */
    setbkcolor(0); /* 设置背景为黑色 */
    cleardevice( ); /* 清屏 */
    for(i=0;i<=15;i++) /* 设置 16 次循环 */
    {
        setcolor(i); /* 设置不同作图色 */
        circle(320,240,20+i*10); /* 画半径不同的圆 */
        delay(100); /* 延时 100 毫秒 */
    }
    for(i=0;i<=15;i++) /* 设置 16 次循环 */
    {
        setbkcolor(i); /* 设置不同背景色 */
        cleardevice( );
        circle(320,240,20+i*10);
        delay(100);
    }
    closegraph( ); /* 关闭图形模式 */
}
```

另外, Turbo C 还提供了几个检测当前颜色设置情况的函数。

<code>int far getbkcolor(void);</code>	返回当前背景颜色值。
<code>int far getcolor(void);</code>	返回当前作图颜色值。
<code>int far getmaxcolor(void);</code>	返回最高可用的颜色值。

#### 7.2.4 基本图形函数

基本图形函数包括画点、线以及其它一些基本图形的函数。本节对这些函数作一全面的介绍。

##### 一、画点

##### 1. 画点函数

画点函数的调用格式为:

```
void far putpixel(int x,int y,int color);
```

该函数表示在指定的象素处画一个颜色按 `color` 所确定的点。至于颜色 `color` 的值可从表 7-3 中获得, 而 `x`, `y` 对则是指图形象素的坐标。

在图形模式下, 是按象素来定义坐标的。对于 VGA 适配器, 它的最高分辨率为 640 × 480, 其中 640 为整个屏幕从左到右所有象素的个数, 480 为整个屏幕从上到下所有象素的个数。屏幕的左上角坐标为(0,0), 右下角坐标为(639,479), 水平方向从左到右为 `x` 轴正向, 垂直方向从上到下为 `y` 轴正向。Turbo C 的图形函数都是相对于图形屏幕坐标, 即象素来说的。

关于点的另外一个函数是:

```
int far getpixel(int x,int y);
```

它获得当前点(`x`,`y`)的颜色值。

##### 2. 有关坐标位置的函数

```
int far getmaxx(void);
```

返回 `x` 轴向的最大值。

```
int far getmaxy(void);
```

返回 `y` 轴向的最大值。

```
int far getx(void);
```

返回游标在 `x` 轴向的位置。

```
int far gety(void);
```

返回游标在 `y` 轴向的位置。

```
void far moveto(int x, int y);
```

移动游标到点(`x`,`y`), 但不是画点, 在移动过程中亦不画点。

```
void far moverel(int dx, int dy);
```

将游标从现行位置(`x`,`y`)移动到位置(`x+dx`, `y+dy`), 移动过程中不画点。

##### 二、画线

##### 1. 画线函数

Turbo C 提供了一系列画线函数, 下面分别叙述:

```
void far line(int x0, int y0, int x1, int y1);
```

画一条从点(x0, y0)到(x1, y1)的直线。

void far lineto(int x, int y);

画一条从现行游标到点(x, y)的直线。

void far linerel(int dx, int dy);

画一条从现行游标(x,y)到按相对增量确定的点(x+dx, y+dy)的直线。

void far circle(int x, int y, int radius);

以(x, y)为圆心, radius 为半径画一个圆。

void far arc(int x, int y, int stangle, int endangle, int radius);

以(x, y)为圆心, radius 为半径, 从 stangle 开始到 endangle 为止(用度表示)画一段圆弧线。在 Turbo C 中规定 x 轴正向为 0 度, 逆时针方向旋转一周, 顺次为 90、180、270 和 360 度(其它有关函数也按此规定, 不再重述)。

void ellipse(int x, int y, int stangle, int endangle, int xradius, int yradius);

以(x, y)为中心, xradius、yradius 为 x 轴和 y 轴半径, 从角 stangle 开始到 endangle 为止画一段椭圆弧。当 stangle=0, endangle=360 时, 画出一个完整的椭圆。

void far rectangle(int x1, int y1, int x2, int y2);

以(x1, y1)为左上角, (x2, y2)为右下角画一个矩形框。

void far drawpoly(int numpoints, int far \* polypoints);

画一个顶点数为 numpoints, 各顶点坐标由 polypoints 给出的多边形。polypoints 整型数组必须至少有 2 倍于顶点个数的元素。每一个顶点的坐标都定义为 x,y, 并且 x 在前。值得注意的是在画一个封闭多边形时, numpoints 的值取实际多边形的顶点数加一, 并且数组 polypoints 中第一个和最后一个点的坐标相同。

下面举一个用 drawpoly( )函数画箭头的例子。

例 7-9:

```
#include <graphics.h>
main( )
{
    int gdriver,gmode,i;
    int arw[16]={200,102,300,102,300,107,          /* 定义顶点坐标值 */
                330,100,300,93,300,98,200,98,200,102};
    gdriver = DETECT;
    registerbgidriver(EGAVGA_driver);              /* 建立独立图形程序 */
    initgraph(&gdriver,&gmode,"c:\\tc");            /* 图形初始化 */
    setbkcolor(BLUE);                               /* 设置蓝色背景 */
    cleardevice( );                                 /* 清屏 */
    setcolor(12);                                   /* 设置作图颜色 */
    drawpoly(8,arw);                                /* 画一箭头 */
    getch( );
    closegraph( );
}
```

注:

- 上述函数按 `setcolor()` 函数设置的顏色画线。

## 2. 设定线型函数

在没有对线的特性进行设定之前, Turbo C 用其默认值, 即一点宽的实线。但 Turbo C 也提供了可以改变线型的函数。线型包括宽度和形状。其中宽度只有两种选择: 一点宽和三点宽。而线的形状则有五种。下面介绍有关线型的设置函数。

`void far setlinestyle(int linestyle, unsigned upattern, int thickness);`

该函数用来设置线的有关信息, 其中 `linestyle` 是线形状的规定, 见表 7-5。

表 7-5 线的形状(linestyle)

符号常数	数 值	含 义
SOLID__LINE	0	实线
DOTTED__LINE	1	点线
CENTER__LINE	2	中心线
DASHED__LINE	3	点划线
USERBIT__LINE	4	用户定义线

`thickness` 是线的宽度, 见表 7-6。

表 7-6 线宽(thickness)

符号常数	数 值	含 义
NORM__WIDTH	1	一点宽
THICK__WIDTH	3	三点宽

至于 `upattern`, 只有在 `linestyle` 选 `USERBIT__LINE` 时才有意义(选其它线型时 `upattern` 取 0 即可)。此时 `upattern` 的 16 位二进制数的每一位代表一个象素, 如果某位为 1, 则该象素打开, 否则该象素关闭。

`void far getlinesettings(struct linesettingstype far * lincinfo);`

该函数将有关线的信息存放到由 `lincinfo` 指向的结构中, 表中 `linesettingstype` 的结构如下:

```
struct linesettingstype{
    int linestyle;
    unsigned upattern;
    int thickness;
};
```

例如下面两行程序可以读出当前线的特性:

```
struct linesettingstype * info;
getlinesettings(info);
```

`void far setwrite mode(int mode);`

该函数规定画线的方式。如果 `mode=0`, 则表示画线时将所画位置上原来的信息覆



盖掉(这是 Turbo C 的默认方式)。如果 mode=1, 则表示画线时用现在特性的线与所画之处原有的线进行异或(XOR)操作, 实际上画出的线是原有线与现在规定的线进行异或后的结果。因此, 在线的特性不变的情况下, 进行两次画线操作相当于没有画线。

有关线型设定和画线函数的例子如下所示。

#### 例 7-10:

```
#include <graphics.h>
main( )
{
    int gdriver,gmode,i;
    gdriver = DETECT;
    registerbgidriver(EGAVGA_driver);    /* 建立独立图形运行程序 */
    initgraph(&gdriver,&gmode,"c:\\tc"); /* 图形初始化 */
    setbkcolor(BLUE);                    /* 设置蓝色背景 */
    cleardevice( );                      /* 清屏幕 */
    setcolor(GREEN);                     /* 设置绿色作图 */
    circle(320,240,98);                  /* 画圆 */
    setlinestyle(0,0,3);                 /* 设置三点宽实线线型 */
    setcolor(12);                         /* 设置淡红作图色 */
    rectangle(220,140,420,340);          /* 画一矩形框 */
    setcolor(WHITE);                     /* 设置白色作图 */
    setlinestyle(4,0xaaaa,1);           /* 设置一点宽用户定义线 */
    line(220,240,420,240);               /* 画线 */
    line(320,140,320,340);
    getch( );
    closegraph( );
}
```

### 7.2.5 封闭图形的填充

填充就是用规定的颜色和图模填满一个封闭图形。

#### 一、先画轮廓再填充

Turbo C 提供了一些先画出基本图形轮廓, 再按规定的图模和颜色填满整个封闭图形的函数。在没有改变填充方式时, Turbo C 以默认方式填充。下面介绍这些函数。

```
void far bar(int x1, int y1, int x2, int y2);
```

确定一个以(x1, y1)为左上角, (x2, y2)为右下角的矩形窗口, 再按规定图模和颜色填充。

注:

- 此函数不画出边框, 即以填充色为边框。

```
void far bar3d(int x1,int y1,int x2,int y2,int depth, int topflag);
```

当 topflag 为非 0 时, 画出一个三维的长方体, 如图 7-1 所示。图中的阴影部分代表按规定的图模和颜色填充。

注:

● `bar3d()` 函数中, 长方体第三维的方向不随任何参数而改变, 即始终为约 45 度的方向。depth 如图 7-1 中所示。

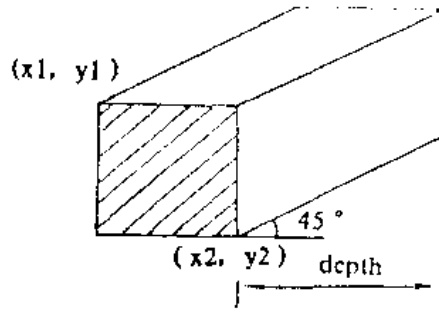


图 7-1 `bar3d()` 函数中 `topflag` 为非 0 时所作图形

当 `topflag` 为 0 时, 三维图形不封顶, 此时变为图 7-2 所示。

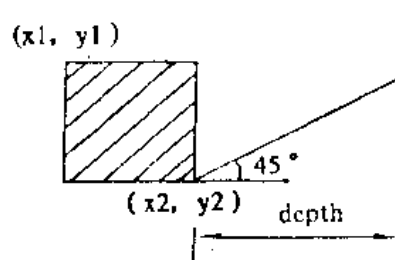


图 7-2 `bar3d()` 函数中 `topflag` 为 0 时所作图形

实际上很少这样使用。

`void far pieslice(int x,int y,int stangle,int endangle,int radius);`

画一个以  $(x,y)$  为圆心, `radius` 为半径, `stangle` 为起始角度, `endangle` 为终止角度的扇形, 再按规定方式填充。当 `stangle=0`, `endangle=360` 时成为一个实心圆, 并在圆内从圆心沿 X 轴正向画一条半径。

`void far sector(int x,int y,int stangle,int endangle,int xradius,int yradius);`

画一个以  $(x,y)$  为圆心分别以 `xradius`、`yradius` 为 x 轴和 y 轴半径, `stangle` 为起始角, `endangle` 为终止角的椭圆扇形, 再按规定方式填充。

二、设定填充方式

Turbo C 有四个与填充方式有关的函数。下面分别介绍:

`void far setfillstyle(int pattern, int color);`

`color` 的值是当前屏幕图形模式时颜色的有效值。 `pattern` 的值及与其等价的符号常数

如表 7-7 所示。

除 USER\_FILL(用户定义填充式样)以外, 其它填充式样均可由 setfillstyle() 函数设置。当选用 USER\_FILL 时, 该函数对填充图模和颜色不作任何改变。之所以定义 USER\_FILL 主要因为在获得有关填充信息时用到此项。

表 7-7 填充图模 (pattern)

符号常数	数 值	含 义
EMPTY_FILL	0	以背景颜色填充
SOLID_FILL	1	以实填充
LINE_FILL	2	以直线填充
LTSLASH_FILL	3	以斜线(阴影线)填充
SLASH_FILL	4	以粗斜线(粗阴影线)填充
BKSLASH_FILL	5	以粗反斜线(粗阴影线)填充
LTBKSLASH_FILL	6	以反斜线(阴影线)填充
HATCH_FILL	7	以直方网格填充
XHATCH_FILL	8	以斜网格填充
INTERLEAVE_FILL	9	以间隔点填充
WIDE_DOT_FILL	10	以稀疏点填充
CLOSE_DOT_FILL	11	以密集点填充
USER_FILL	12	以用户定义式样填充

```
void far setfillpattern(char * upattern, int color);
```

设置用户定义的填充图模和颜色以供填充封闭图形。

其中 upattern 是一个指向 8 个字节的指针。这 8 个字节定义了 8×8 点阵的图形。每个字节的 8 位二进制数表示水平 8 点, 共 8 个字节表示 8 行, 然后以此为图模填充整个封闭区域。

```
void far getfillpattern(char * upattern);
```

该函数将用户定义的填充图模存入 upattern 指针指向的内存区域。

```
void far getfillsettings(struct fillsettingstype far * fillinfo);
```

获得现行图模和颜色并将其存入结构指针变量 fillinfo 中。其中 fillsettingstype 结构定义如下:

```
struct fillsettingstype{
    int pattern;    /* 现行填充图模 */
    int color;      /* 现行填充颜色 */
};
```

有关图形填充图模和颜色的选择, 请看下面例程。

例 7-11:

```
#include <graphics.h>
main( )
{
    char str[8]={10,20,30,40,50,60,70,80}; /* 用户定义图模 */
```

```

int gdriver,gmode,i;
struct fillsettingstype save; /* 定义一个用来存储填充信息的结构变量 */
gdriver=DETECT;
initgraph(&gdriver, &gmode, "c:\\tc");
setbkcolor(BLUE);
cleardevice( );
for(i=0;i<13;i++)
{
    setcolor(i+3);
    setfillstyle(i,2+i);          /* 设置填充类型 */
    bar(100,150,200,50);          /* 画矩形并填充 */
    bar3d(300,100,500,200,70,1); /* 画长方体并填充正面 */
    pieslice(200,300,90,180,90); /* 画扇形并填充 */
    sector(500,300,180,270,200,100); /* 画椭圆扇形并填充 */
    dclay(1000);                  /* 延时 1 秒 */
}
cleardevice( );
setcolor(14);
setfillpattern(str,RED);          /* 设置用户自定义填充图模和颜色 */
bar(100,150,200,50);
bar3d(300,100,500,200,70,0);
pieslice(200,300,0,360,90);
sector(500,300,0,360,100,50);
getch( );
getfillsettings(&save);          /* 获得用户定义的填充图模信息 */
closegraph( );
clrscr( );
printf("The pattern is %d,and the color of filling is %d",
    save.pattern,save.color);      /* 输出目前填充图模和颜色值 */
getch( );
}

```

以上程序运行结束后，在屏幕上显示出现行填充图模和颜色的常数值。

### 三、任意封闭图形的填充

迄今为止，我们只能对一些特定形状的封闭图形进行填充，暂时还不能对任意封闭图形进行填充。为此，Turbo C 提供了一个可以填充任意封闭图形的函数，它的调用格式如下：

```
void far floodfill(int x,int y, int border);
```

其中：x, y 为封闭图形内的任意一点。border 为边界的颜色，也就是封闭图形轮廓的颜色。调用了该函数后，将用规定的颜色和图模填满整个封闭图形。

注：

- 如果 x, y 取在边界上, 则不进行填充。
- 如果不是封闭图形, 则填充时会从没有封闭的地方溢出去, 填到其它地方。
- 如果 x, y 在图形外面, 则填充封闭图形外的屏幕区域。
- 由 border 指定的颜色值必须与图形轮廓的颜色值相同, 但填充色可选任意颜色。

下例是有关 floodfill( )函数的用法, 该程序填充了 bar3d( )所画长方体中其它两个未填充的面。

例 7-12:

```
#include <graphics.h>
main( )
{
    int gdriver,gmode;
    struct fillsettingstype save;
    gdriver = DETECT;
    initgraph(&gdriver, &gmode, "c:\\ \\ tc");
    setbkcolor(BLUE);          /* 设置图形背景色 */
    cleardevice( );
    setcolor(LIGHTRED);        /* 设置作图颜色 */
    setlinestyle(0,0,3);       /* 设置线型 */
    setfillstyle(1,14);        /* 设置填充方式 */
    bar3d(100,200,400,350,200,1); /* 画长方体并填充正面 */
    floodfill(450,300,LIGHTRED); /* 填充长方体另外两个面 */
    floodfill(250,150,LIGHTRED);
    rectangle(450,400,500,450); /* 画一矩形 */
    floodfill(470,420,LIGHTRED); /* 填充矩形 */
    getch( );
}
```

## 7.2.6 图形窗口和图形屏幕操作函数

### 一、图形窗口操作

象文本方式下可以设定屏幕窗口一样, 图形方式下也可以在屏幕上某一区域设定窗口, 只是设定的为图形窗口而已, 其后的有关图形操作都将以这个窗口的左上角(0,0)作为坐标原点, 而且可以通过设置使窗口之外的区域为“不可接触”。这样, 所有的图形操作就被限定在窗口内进行。

```
void far setviewport(int x1,int y1,int x2,int y2,int clipflag);
```

设定一个以(x1,y1)象素点为左上角, (x2,y2)象素点为右下角的图形窗口, 其中 x1、y1、x2、y2 是相对于整个屏幕的坐标。若 clipflag 为非 0, 则设定的图形窗口以外部分不可接触; 若 clipflag 为 0, 则图形窗口以外可以接触。

```
void far clearviewport(void);
```

清除现行图形窗口的内容。

```
void far getviewsettings(struct viewporttype far *viewport);
```

获得关于现行窗口的信息，并将其存于 viewporttype 定义的结构变量 viewport 中，其中 viewporttype 的结构说明如下：

```
struct viewporttype{
    int left,top,right,bottom;
    int clipflag;
};
```

注：

- 窗口颜色的设置与前述屏幕颜色设置相同，但屏幕背景色和窗口背景色只能是同一种颜色，如果窗口背景色改变，整个屏幕的背景色也将改变，这与文本窗口不同。
- 可以在同一个屏幕上设置多个窗口，但只能有一个现行窗口工作，如要对其它窗口进行操作，再调用一次定义那个窗口的 setviewport( )函数即可。
- 前面讲过的对图形屏幕操作的函数均适合于对窗口的操作。

## 二、屏幕操作

除了清屏函数以外，关于屏幕操作还有以下函数：

```
void far setactivepage(int pagenum);
void far setvisualpage(int pagenum);
```

这两个函数只用于 EGA、VGA 以及 HERCULES 图形适配器。setactivepage( )函数是为图形输出选择激活页。所谓激活页是指后续图形的输出被写到该函数选定的 pagenum 页面，该页面并不一定可见。setvisualpage( )函数才使 pagenum 所指定的页面变成可见页。页面从 0(Turbo C 默认页)开始。如果先用 setactivepage( )函数在不同页面上画出一幅幅图像，再用 setvisualpage( )函数交替显示，就可以实现一些动画的效果。

```
void far getimage(int x1,int y1,int x2,int y2,void far *mapbuf);
void far putimage(int x,int y,void *mapbuf,int op);
unsigned far imagesize(int x1,int y1,int x2,int y2);
```

这三个函数用于将屏幕上的图像复制到内存，然后再将内存中的图像送回到屏幕上。首先通过函数 imagesize( )测算要保存左上角为(x1,y1)，右下角为(x2,y2)的图形屏幕区域内的全部内容需用多少个字节，然后再给 mapbuf 分配一个所测得字节数内存空间的指针。通过调用 getimage( )函数就可将该区域内的图像保存在内存中，需要时可用 putimage( )函数将该图像输出到左上角为点(x,y)的位置上，其中 getimage( )函数中的参数 op 规定如何释放内存中的图像。

关于这个参数的定义参见表 7-8。

对于 imagesize( )函数，只能返回小于 64K 字节的图像区域，否则将会出错，出错时返回-1。

本节介绍的函数在图像动画处理和菜单设计中非常有用。

表 7-8 putimage( )函数中的 op 值

符号常数	数 值	含 义
COPY_PUT	0	复制
XOR_PUT	1	与屏幕图像异或后复制
OR_PUT	2	与屏幕图像或后复制
AND_PUT	3	与屏幕图像与后复制
NOT_PUT	4	复制图形的反像

例 7-13: 下面的程序模拟两个小球动态碰撞的过程。

```
#include <graphics.h>
main( )
{
    int i,gdriver,gmode,size;
    void *buf;          /* 定义一个指针 */
    gdriver = DETECT;
    initgraph(&gdriver, &gmode, "c:\\tc");
    setbkcolor(BLUE);    /* 设置蓝色背景 */
    cleardevice( );      /* 清除屏幕 */
    setcolor(LIGHTRED);  /* 设置淡红作图色 */
    setlinestyle(0,0,1); /* 设置一点宽实线 */
    setfillstyle(1,10);  /* 设置淡绿色实填充 */
    circle(100,200,30);  /* 画圆 */
    floodfill(100,200,12); /* 填充圆 */
    size = imagesize(69,169,131,231); /* 返回存储屏幕图形所需的字节长度 */
    buf = malloc(size);   /* 动态分配内存空间 */
    getimage(69,169,131,231,buf); /* 保存指定区域的圆饼图 */
    putimage(500,169,buf,COPY_PUT); /* 在另一位置释放圆饼图 */
    for(i=0;i<185;i++)    /* 设置循环, 两球相对运动直到相撞 */
    {
        putimage(70+i,170,buf,COPY_PUT); /* 左边球向右移 */
        putimage(500-i,170,buf,COPY_PUT); /* 右边球向左移 */
    }
    for(i=0;i<185;i++)    /* 设置循环, 两球反向运动 */
    {
        putimage(255-i,170,buf,COPY_PUT); /* 左边球向左移 */
        putimage(315+i,170,buf,COPY_PUT); /* 右边球向右移 */
    }
    getch( );
    closegraph( );
}
```

### 7.2.7 图形模式下的文本输出

在图形模式下,只能使用标准输出函数,如 `printf()`、`puts()`、`putchar()` 函数输出文本到屏幕,除此之外的其它输出函数(如窗口输出函数)则不能使用,即使可以输出的标准函数,也只能以前景色为白色,按 80 列,25 行的文本方式输出。

Turbo C 也提供了一些专门用于图形显示模式下使用的文本输出函数,下面将分别进行介绍。

#### 一、文本输出函数

`void far outtext(char far *textstring);`

该函数输出字符串指针 `textstring` 所指向的文本在现行位置。

`void far outtextxy(int x,int y,char far *textstring);`

该函数输出字符串指针 `textstring` 所指向的文本在规定的  $(x,y)$  位置。其中  $x$  和  $y$  为象素的坐标。

注:

● 这两个函数都是输出字符串,但经常会遇到输出数值或其它类型的数据,此时就必须使用格式化输出函数 `sprintf()`。

`sprintf()` 函数的调用格式为:

`int sprintf(char *str,char *format,variable-list);`

它与 `printf()` 函数不同之处是将按格式化规定的内容写入 `str` 指向的字符串中,返回值等于写入的字符个数。

例如:

`sprintf(s,"Your TOEFL score is %d",toefl);`

这里的 `s` 应是字符串指针或数组, `toefl` 为整型变量。

#### 二、有关文本字体、字型和输出方式的设置

有关图形方式下的文本输出函数,可以通过 `setcolor()` 函数设置输出文本的颜色。另外,也可以改变文本字体、大小以及选择是水平方向输出还是垂直方向输出。

`void far setttextjustify(int horiz,int vert);`

该函数用于定位输出字符串。

对使用 `outtextxy(int x,int y,char far *textstring)` 函数所输出的字符串,其中哪个点对应于定位坐标  $(x,y)$  在 Turbo C 中是有规定的。如果把一个字符串看成一个长方形的图形,在水平方向显示时,字符串长方形按垂直方向可分为顶部、中部和底部三个位置,水平方向可分为左、中、右三个位置,两者结合就有 9 个位置。

`setttextjustify()` 函数的第一个参数 `horiz` 指出水平方向三个位置中的一个,第二个参数 `vert` 指出垂直方向三个位置中的一个,二者就确定了其中一个位置。当规定了这个位置后,用 `outtextxy()` 函数输出字符串时,字符串长方形的这个规定位置就对准函数中的  $(x,y)$  位置。而对用 `outtext()` 函数输出字符串时,这个规定的位置就位于现行游标的位置。有关参数 `horiz` 和 `vert` 的取值参见表 7-9。



表 7-9 参数 horiz 和 vert 的取值

符号常数	数 值	用 于
LEFT__TEXT	0	水平
RIGHT__TEXT	2	水平
BOTTOM__TEXT	0	垂直
TOP__TEXT	2	垂直
CENTER__TEXT	1	水平或垂直

void far settextstyle(int font,int direction,int charsize);

该函数用来设置输出字符的字体(由 font 确定)、输出方向(由 direction 确定)和字符大小(由 charsize 确定)等特性。Turbo C 对函数中各个参数的规定见下列各表所示。

表 7-10 font 的取值

符号常数	数 值	含 义
DEFAULT__FONT	0	8*8 点阵字(缺省值)
TRIPLEX__FONT	1	三倍笔划字体
SMALL__FONT	2	小号笔划字体
SANSSERIF__FONT	3	无衬线笔划字
GOTHIC__FONT	4	黑体笔划字

表 7-11 direction 的取值

符号常数	数 值	含 义
HORIZ__DIR	0	从左到右
VERT__DIR	1	从底到顶

有关图形屏幕下文本输出和字体字型设置函数的用法请看例 7-14。

例 7-14:

```
#include <graphics.h>
main( )
{
    int i,gdriver,gmode;
    char s[30];
    gdriver = DETECT;
    initgraph(&gdriver, &gmode, "c:\\ \\tc");
    setbkcolor(BLUE);
    cleardevice( );
    setviewport(100,100,540,380,1);    /* 定义一个图形窗口 */
    setfillstyle(1,2);                  /* 设置绿色以实填充 */
    setcolor(YELLOW);                  /* 设置黄色作图 */
    rectangle(0,0,439,279);             /* 画一矩形框 */
    floodfill(50,50,14);                /* 填充此矩形框 */
    setcolor(12);                       /* 改变作图颜色为淡红色 */
    settextstyle(1,0,8);                /* TRIPLEX__FONT 字体, 水平输出放大 8 倍 */
}
```

```

outtextxy(20,20,"Good news");      /* 在指定位置输出文本 */
setcolor(15);                       /* 改变作图颜色为白色 */
settextstyle(3,0,5);                /* SANSERIF_FONT 字体, 水平输出放大 5 倍 */
outtextxy(120,120,"Good news");
setcolor(14);                       /* 改变作图颜色为黄色 */
settextstyle(2,0,8);                /* SMALL_FONT 字体, 水平输出放大 8 倍 */
i=617;
sprintf(s,"Your score of TOEFL is %d",i); /* 将数字转化成字符串 */
outtextxy(30,200,s);                /* 在指定位置输出字符串 */
setcolor(1);                       /* 改变作图颜色为蓝色 */
settextstyle(4,0,3);                /* GOTHIC_FONT 字体, 水平输出放大 3 倍 */
outtextxy(70,240,s);                /* 在指定位置输出字符串 */
getch( );
closegraph( );
}

```

表 7-12 charsize 的取值

符号常数或数值	含 义
1	8×8 点阵
2	16×16 点阵
3	24×24 点阵
4	32×32 点阵
5	40×40 点阵
6	48×48 点阵
7	56×56 点阵
8	64×64 点阵
9	72×72 点阵
10	80×80 点阵
USER_CHAR_SIZE=0	用户定义的字符大小

### 三、用户对文本字符大小的设置

前面介绍的 `settextstyle()` 函数, 可以设定图形方式下输出文本字符的字体和大小, 但对于笔划型字体(除 8×8 点阵字以外的字体), 只能在水平和垂直方向以相同的放大倍数放大。为此 Turbo C 又提供了另外一个函数 `setusercharsize()`, 对笔划字体可以分别设置水平和垂直方向的放大倍数。该函数的调用格式为:

```
void far setusercharsize(int mulx,int divx,int muly,int divy);
```

该函数用来设置笔划型字的放大系数, 它只有在 `settextstyle()` 函数中的 `charsize` 为 0(或 `USER_CHAR_SIZE`)时才起作用, 并且字体为函数 `settextstyle()` 规定的字体。调用函数 `setusercharsize()` 后, 每个显示在屏幕上的字符都以其缺省大小乘以  $mulx / divx$  为输出字符宽, 乘以  $muly / divy$  为输出字符高。该函数的用法见下例。

#### 例 7-15:

```
#include <graphics.h>
```

```

main( )
{
    int gdriver,gmode;
    gdriver=DETECT;
    initgraph(&gdriver, &gmode, "c:\\tc");
    setbkcolor(BLUE);
    cleardevice( );
    setfillstyle(1,2);          /* 设置填充方式 */
    setcolor(YELLOW);          /* 设置黄色作图 */
    rectangle(100,100,540,380); /* 画一黄色方框 */
    floodfill(50,50,14);      /* 种子点不在方框内, 填充方框以外的屏幕区 */
    setcolor(12);              /* 改变作图色为淡红 */
    settextstyle(1,0,8);        /* 用三倍笔划字体, 放大8倍 */
    outtextxy(120,120,"Good news"); /* 输出字符串 */
    setusercharsize(2,1,4,1);   /* 水平放大2倍, 垂直放大4倍 */
    settextstyle(1,0,0);        /* 选字体和用户定义放大倍数 */
    outtextxy(150,200,"Good news"); /* 输出字符串 */
    setcolor(15);              /* 改用白色作图色 */
    settextstyle(3,0,5);        /* 用无衬线笔划字体, 放大5倍 */
    outtextxy(220,200,"Good news"); /* 输出字符串 */
    setusercharsize(4,1,1,1);   /* 水平放大4倍, 垂直放大1倍 */
    settextstyle(3,0,0);        /* 选字体和用户定义放大倍数 */
    outtextxy(180,320,"Good news"); /* 输出字符串 */
    getch( );
    closegraph( );
}

```

## 第八章 Turbo C 实用编程

通过前面章节的学习,读者对 Turbo C 语言已有了全面、基本的了解,但在进行实际编程时,还会遇到不少问题。本章介绍一些实用编程技术,主要包括:怎样编写 Turbo C 源程序使其能在汉字系统下输入输出汉字、Turbo C 的 BIOS、DOS 系统调用函数、字符串和内存管理函数、数学运算函数等。最后介绍在集成开发环境下程序的调试和源程序命令行编译与链接的方法。

### 8.1 含汉字输入输出的程序的编制

如果不是汉化的 Turbo C,则不能在汉字操作系统下加载,不然将会发生系统死机或屏幕不显示 Turbo C 集成开发环境中的菜单。但是用户常有需要编制汉化的应用程序,即能在汉字操作系统运行并可输入输出汉字。下面介绍有关这方面的知识。

在汉字操作系统下,屏幕将处于图形模式(但未进行图形初始化时不能直接使用图形函数),此时只有标准文本输入输出函数(如: `printf()`、`puts()`、`putchar()`、`scanf()`、`gets()`、`getche()`等函数)可以使用。也可用 `window()` 函数定义一个文本窗口, `gotoxy()` 函数定位输入输出汉字的位置, `textbackground()` 设置窗口背景色, `textcolor()` 设置输出汉字的颜色,但有关窗口的输入输出函数将不能使用。另外,保存屏幕内容和释放到屏幕上的函数 `gettext()` 和 `puttext()` 也不能使用。

有关汉字程序的编制请遵循下述步骤:

1. 在 Turbo C 集成开发环境中编制程序,在需要输出汉字的地方,暂且用西文字符串代替,对程序进行编译和链接,保证运行正确。
2. 退出 Turbo C 集成开发环境。
3. 进入汉字操作系统,用有关文字编辑软件将程序中要输出汉字的地方临时写入的西文字符用相应汉字代替。
4. 退出汉字操作系统,进入 Turbo C 集成开发环境中重新编译链接程序,生成可执行文件。
5. 再次进入汉字操作系统,直接运行生成的 EXE 文件,就可按程序的要求输入输出汉字了。

注:

- 可以用 `textbackground()` 函数设置窗口的背景色,用 `textcolor()` 函数设置窗口的前景色,但在这两个函数后必须有一条 `clrscr()` 函数,才可改变颜色。否则汉字以白色输出。

有关汉字的定位输入输出和颜色的变化,请仔细阅读下面这段程序就可明确。

**例 8-1:**

```
#include <stdio.h>
```

```

#include <conio.h>
void sub(void);          /* 子函数说明 */
unsigned char s[12];     /* 定义全程变量 */
main( )
{
    unsigned char *str;   /* 定义局部变量 */
    textbackground(BLACK); /* 设置文本背景色 */
    clrscr( );           /* 清除文本屏幕 */
    textcolor(LIGHTRED);  /* 设置文本字符色 */
    window(3,3,40,10);   /* 定义文本窗口 */
    textbackground(BLUE); /* 改变文本背景色 */
    clrscr( );
    gotoxy(3,2);          /* 在当前窗口中定位光标 */
    printf("您贵姓? ");  /* 输出汉字 */
    scanf("%s",str);      /* 键盘输入字符或汉字 */
    gotoxy(3,4);
    printf("您好!, %s 先生",str);
    sub( );               /* 调用子函数 */
    exit(0);              /* 退出 */
}

void sub( )
{
    unsigned char *str[]={"工作单位: ","职业: "}; /* 定义汉字字符串 */
    char c;
    unsigned s1[40], *s2;
    window(30,9,70,15); /* 重新定义文本窗口 */
    textbackground(GREEN); /* 设置新窗口背景色为绿色 */
    textcolor(YELLOW);    /* 改变输出字符颜色 */
    clrscr( );           /* 清除窗口 */
    gotoxy(5,2);
    puts("请您自我介绍一下好吗? (Y/N)"); /* 在当前窗口中输出汉字 */
    gotoxy(32,2);
    while(1)             /* 设置循环 */
    {
        c=getch( );      /* 从键盘接收任一字符 */
        if(c=='Y'||c=='y') /* 判断接收的是否为字母 Y 或 y */
        {
            /* 是则执行此循环体 */
            gotoxy(9,4);
            puts(str[0]);
            gotoxy(19,4);
            scanf("%s",s1);
            gotoxy(9,6);
        }
    }
}

```

```

        printf("%s",str[1]);
        gotoxy(15,6);
        scanf("%s",s2);
        break;
    }
    else if(c=='N'|c=='n') /* 判断接收的是否为字母 N 或 n */
        break; /* 是则退出循环, 不是重新循环 */
}
}

```

以上介绍的是屏幕为文本方式时汉字的输入输出, 但当屏幕在图形方式时, 这种方法仍然可以使用, 例 8-1 加入图形初始化的几条图形函数后就变成下面的例 8-2。

#### 例 8-2:

```

#include <stdio.h>
#include <conio.h>
#include <graphics.h>
void sub(void);
unsigned char s[12];
main( )
{
    int gdriver=DETECT,gmode;
    unsigned char * str;
    registerbgidriver(EGAVGA__driver); /* 建立独立图形运行程序 */
    initgraph(&gdriver,&gmode,"c:\\tc"); /* 图形初始化 */
    setbkcolor(BLACK); /* 设置图形屏幕背景色 */
    cleardevice( ); /* 清除图形屏幕 */
    setcolor(YELLOW); /* 设置作图颜色 */
    setlinestyle(0,0,3); /* 设置线型 */
    rectangle(15,34,321,181); /* 作一矩形框 */
    window(3,3,40,10); /* 定义文本窗口 */
    textbackground(BLUE); /* 设置文本背景色 */
    textcolor(LIGHTRED); /* 设置文本字符颜色 */
    clrscr( ); /* 清除文本屏幕 */
    gotoxy(3,2);
    printf("您贵姓? ");
    scanf("%s",str);
    gotoxy(3,4);
    printf("您好!, %s 先生",str);
    sub( );
    exit(0);
}
void sub( )

```

```

{
    unsigned char *str[]={"工作单位:", "职业:"};
    char c;
    unsigned s1[40], *s2;
    setcolor(LIGHTRED);          /* 改变作图颜色 */
    rectangle(230,140,560,300);  /* 画矩形框 */
    setfillstyle(1, GREEN);      /* 设置填充方式 */
    window(32,9,68,15);         /* 定义文本窗口 */
    textbackground(GREEN);       /* 设置文本窗口背景色 */
    textcolor(YELLOW);           /* 设置文本字符颜色 */
    clrscr( );                   /* 清除文本窗口 */
    bar(232,142,558,298);       /* 画一矩形框并按设定方式填充 */
    gotoxy(1,2);                 /* 确定文本输出位置 */
    puts("请您自我介绍一下好吗? (Y/N)"); /* 输出汉字 */
    gotoxy(28,2);
    while(1)
    {
        c=getch( );
        if(c == 'Y' || c == 'y')
        {
            gotoxy(5,4);
            puts(str[0]);
            gotoxy(15,4);
            scanf("%s", s1);
            gotoxy(5,6);
            printf("%s", str[1]);
            gotoxy(11,6);
            scanf("%s", s2);
            break;
        }
        else if(c == 'N' || c == 'n')
            break;
    }
    closegraph( );              /* 退出图形方式 */
}

```

注:

- 在图形方式下, 文本或文本背景的颜色与图形或图形屏幕背景的颜色无关。如要改变颜色, 应分别按相应的函数设置。

- 图形模式时, 不能用有关图形方式下文本输出函数输出汉字, 即不能用下面函数输出汉字:

```
outtext( );
```

outtextxy( );

## 8.2 Turbo C 提供的 BIOS 和 DOS 系统调用函数

Turbo C 提供了一些与操作系统有关的函数, 这些函数使用起来比较灵活, 可以实现许多高级功能。下面介绍一些常用的函数。

### 8.2.1 键盘操作函数 bioskey( )

bioskey( )函数直接调用了 BIOS int16 中断对键盘进行管理, 该中断包括两个功能: 一是判断是否按了键并返回所按之键的 ASCII 码(字符键)或扩充码(功能键)值; 二是返回键盘的状态。

虽然标准输入函数可以返回所输入的字符, 但对于一些特殊键如 F1、F2、...、F10、箭头键等就无能为力了, 然而 bioskey( )函数能够完成对这些键的管理功能。该函数的调用格式为:

```
int bioskey(int cmd);
```

函数的头文件为 bios.h。

如果参数 cmd=0, 则表示函数返回一个键盘输入的键的值, 该值为一个两字节 16 位二进制整型数。如果没有进行键盘操作, 将一直处于等待状态。若按下的是一般的 ASCII 码字符("普通键"), 则返回的两字节整数中的低 8 位为所按字符的 ASCII 码值。若按下的是特殊功能键, 则返回的两字节整数中的低 8 位为 0, 高 8 位为所按键的扩充码, 有关微机键盘的 ASCII 码和扩充码的表见附录。

如果参数 cmd=1, bioskey( )函数将用来查询是否按下了一个键, 是则返回非 0 值, 不是则返回 0。

如果参数 cmd=2, 则 bioskey( )函数返回键盘上控制键的状态字, 该状态字以编码方式放在返回值的低 8 位字节中。如果某一位为 1 则该位代表的键被按下, 各位的含义如下:

字节位	对应的十六进制数	含 义
0	0X01	右边的上档键 Shift 被按下
1	0X02	左边的上档键 Shift 被按下
2	0X04	Ctrl 键被按下
3	0X08	Alt 键被按下
4	0X10	ScrollLock 已开通
5	0X20	NumLock 已开通
6	0X40	CapsLock 已开通
7	0X80	Insert 已开通

例如: 如果 bioskey( )函数当 cmd=2 时返回的数为 0X64, 则表示在 CapsLock 和 NumLock 开通的同时又按下了 Ctrl 键。

例 8-3: 下面的程序利用箭头键控制一个实心方块在屏幕上移动, 并设定回车键和空格键作为两种重复键。即当程序开始运行时, 移动光标将在屏幕上画出条形图, 此时若



按回车键则以后移动光标将不画出条形图；再按一次回车键，移动光标又可画出。空格键用来控制是否擦除已画的条形图。按 Esc 键时可退出程序。

```
#include <conio.h>
#include <stdio.h>
#include <stdlib.h>
main( )
{
    char c;
    unsigned char test=0,test1=0,buf[2];
    int x=1,y=1,key1=0,key2=0;
    textbackground(1);      /* 设置文本背景色 */
    textcolor(12);          /* 设置文本字符色 */
    clrscr( );              /* 屏幕着色 */
    gettext(1,1,1,buf);     /* 存屏幕上一个字符 */
    putchar('\xdb');         /* 输出一实心方块 */
    gotoxy(1,1);             /* 定位光标 */
    while(key1!=27)          /* 设置循环直到按 Esc 键 */
    {
        while(bioskey(1)==0); /* 等待直到按任一键 */
        key1=key2=bioskey(0); /* 返回所按键的键码 */
        key1=key1&0xff;        /* 取普通键的 ASCII 码 */
        if(key1==13)           /* 如果按回车键 */
            test=-test;        /* 变量 test 位取反 */
        if(key1==32)           /* 如果按空格键 */
            test1=-test1;      /* 变量 test1 位取反 */
        key2=key2&0xff?0:key2>>8; /* 取特殊键的扩充码 */
        if(key2==72||key2==80||key2==75||key2==77) /* 判断是否为箭头键 */
        {
            if(test==0xff)     /* 根据 test 的值判断移动光标时是否画条形图 */
            {
                if(test1==0xff) /* 根据 test1 的值判断是擦除还是保留原图 */
                    putchar(' '); /* 擦除原位置的图 */
                else
                    puttext(x,y,x,y,buf); /* 保留原位置的图 */
            }
            if(key2==72)y=y<=1?1:y-1; /* 按向上箭头 y 坐标减 1 */
            if(key2==80)y=y>=25?25:y+1; /* 按向下箭头 y 坐标加 1 */
            if(key2==75)x=x<=1?1:x-1; /* 按向左箭头 x 坐标减 1 */
            if(key2==77)x=x>=80?80:x+1; /* 按向右箭头 x 坐标加 1 */
            gettext(x,y,x,y,buf); /* 保存新位置的字符 */
            gotoxy(x,y);          /* 定位光标到新位置 */
        }
    }
}
```

```

        putchar("\xdb");          /* 画出一实心方块 */
        gotoxy(x,y);
    }
}
}

```

本例中, if(key2 == 72)y=y<=1?1:y-1 的含义是: 当 y<=1 时如果按了向上的箭头键, 则 y 恒为 1; 当 y>1 时如果按了向上的箭头键, 则 y=y-1, y-1 可用--y 代替, 但不能用 y--。其它三个箭头键判断语句含义相同。

有时希望程序在执行一个循环时, 对键盘扫描一次, 检查是否按了程序设定的某些键而决定是否转去执行相应的子程序来完成不同的功能, 这种情况在巡回检测中经常用到。下面的程序在执行时, 屏幕每秒钟显示一次系统时间, 在任何时刻只要按回车键, 将清除屏幕并显示 "It is <时间> now", 暂停 2 秒后又重新进入起始画面; 若按 Esc 键则可终止程序运行。

#### 例 8-4:

```

#include <graphics.h>
#include <stdio.h>
#include <dos.h>
void sub(void);
unsigned char s[12];
int i;
main( )
{
    int gdriver=DETECT,gmode,key;
    registerbgidriver(EGAVGA__driver);
    initgraph(&gdriver, &gmode, "c:\\tc");    /* 图形初始化 */
    setbkcolor(BLUE);
    cleardevice( );
    setcolor(12);
    setfillstyle(1,LIGHTGREEN);              /* 设置填充方式 */
    bar(100,100,540,350);                    /* 画矩形框并填充 */
    setttextstyle(1,0,4);                    /* 设置输出文本字体字形 */
    outtextxy(110,130,"Welcome to use this book!"); /* 输出文本 */
    setcolor(1);
    setttextstyle(4,0,3);
    outtextxy(180,270,"The best choice for you");
    gotoxy(28,24);
    printf("Press <Esc> key to exit");
    setcolor(15);
    while(1)                                  /* 设置循环 */
    {
        while(bioskey(1)!=0) /* 扫描键盘检测是否按过键, 若是则执行循环体 */

```

```

{
    key = bioskey(0);    /* 取所按键的键码 */
    key = key & 0x0ff;   /* 取普通键的 ASCII 码 */
    if(key == 27) break; /* 判断若是按了 Esc 键则退出内层循环 */
    if(key == 13)        /* 判断若是按了回车键则执行以下程序体 */
    {
        clrscr( );      /* 清除文本字符 */
        cleardevice( ); /* 清除屏幕图形 */
        gotoxy(30,10);
        printf("It is %s now",s); /* 输出此刻时间 */
        sleep(2);        /* 延迟 2 秒 */
        main( );         /* 递归调用重新运行 main( ) 函数 */
    }
}
if(key == 27) break;    /* 再判断若是按 Esc 键则退出外层循环 */
sub( );
}
closegraph( );         /* 退出图形方式 */
exit(0);               /* 结束程序运行 */
}
void sub( )
{
    struct time * dos__time; /* 定义存放系统时间的结构指针 */
    gettimeofday(dos__time); /* 取系统时间 */
    if(! = dos__time->ti__sec) /* 判断是否过了 1 秒, 不到 1 秒返回主函数 */
    {
        sprintf(s,"%02d:%02d:%02d",dos__time->ti__hour,
        dos__time->ti__min,dos__time->ti__sec); /* 到 1 秒保存时间 */
        i = dos__time->ti__sec;
        gotoxy(72,i);
        puts(s); /* 输出时间 */
    }
}

```

注:

● 在例 8-4 中, 当外层循环较长时, 等待一次键盘扫描的时间间隔就长, 在按键时刻如果程序没有扫描键盘, 就可能对按键不予理睬。解决的办法是在循环体中不同的位置多设几条扫描键盘的语句。

### 8.2.2 打印机操作函数 biosprint( )

用打印机打印西文文本文件时, 可以将“PRN”作为文件名(“PRN”代表打印机)然后象对磁盘文件写内容一样, 可从打印机输出文本。

### 例 8-5:

```
#include <stdio.h>
main( )
{
    FILE *fpo, *fpi;           /* 定义文件指针 */
    char c,s[13];
    clrscr( );
    printf("Please input file name: ");
    scanf("%s",s);             /* 键盘输入要打印的文件名 */
    fpo = fopen(s,"r");         /* 打开输入的文件只读 */
    fpi = fopen("PRN","w");     /* 打开打印机文件只写 */
    while((c = fgetc(fpo)) != EOF) /* 从输入文件读一字符并判断文件是否结束 */
    {
        putchar(c);           /* 屏幕输出该字符 */
        if(c == '\n') fputc('\r',fpi); /* 若为换行符则加一回车符送打印机 */
        fputc(c,fpi);          /* 向打印机输出该字符 */
    }
}
```

也可直接使用 Turbo C 的保留字 `stdin`、`stdout`、`stdprn`。本身`stdprn`就是打印机流。顺便说明一下, Turbo C 对其标准设备都规定了相应的保留字, 在头文件 `stdio.h` 中说明。这些保留字为:

<code>stdin</code>	标准输入设备(键盘)
<code>stdout</code>	标准输出设备(屏幕)
<code>stdprn</code>	标准打印设备(打印机)
<code>stdaux</code>	标准辅助设备
<code>stderr</code>	标准出错设备

以上几种设备在调入 `stdio.h` 后便处于开通状态。

### 例 8-6:

```
#include <stdio.h>
main( )
{
    FILE *fpo;
    char c,s[13];
    clrscr( );
    printf("Please input file name: ");
    scanf("%s",s);             /* 键盘输入要打印的文件名 */
    fpo = fopen(s,"r");         /* 打开文件只读 */
    while((c = fgetc(fpo)) != EOF) /* 从文件读一字符并判断文件是否结束 */
    {
        putchar(c);           /* 屏幕输出该字符 */
    }
}
```

```

        if(c == '\n') fputc('\r', stdprn); /* 若为换行符则加一回车符送打印机 */
        fputc(c, stdprn);                /* 向打印机输出该字符 */
    }
}

```

但以上两种方法只能打印西文字符，而且速度较慢。Turbo C 提供了一个管理打印机的函数 `biosprint()`，该函数直接调用 BIOS 的 17 类中断。

`biosprint()` 函数的调用格式为：

```
int biosprint(int cmd, int byte, int port);
```

其中 `port` 是打印机并行口，当 `port=0` 时，是指 1 号打印机 LPT1；当 `port=1` 时是指 2 号打印机 LPT2。

`byte` 为向打印机输出的字符的 ASCII 码或汉字的内码。

`cmd` 的取值为：

当 <code>cmd=0</code> 时	向打印机输出一个字符 <code>byte</code>
当 <code>cmd=1</code> 时	初始化打印机
当 <code>cmd=2</code> 时	返回打印机的状态(返回值的低 8 位有效)

有关打印机的状态如下：

字节位	对应的十六进制数	含 义
0	0X01	超时错误
1	0X02	未使用
2	0X04	未使用
3	0X08	I/O 错误
4	0X10	联机
5	0X20	缺纸
6	0X40	认可
7	0X80	打印机不忙

例如：当运行了打印机驱动程序后，或自带汉字库的打印机处于中文打印模式时，可用下面例程打印出汉字。

例 8-7：

```

#include <stdio.h>
#include <bios.h>
main( )
{
    FILE * fpo;                /* 定义文件指针 */
    char c,s[13];
    clrscr( );
    printf("Please input file name: ");
    scanf("%s",s);             /* 键盘输入要打印的文件名 */
    fpo = fopen(s,"r");         /* 打开输入的文件只读 */
    while((c = fgetc(fpo)) != EOF) /* 从输入文件读一字符并判断文件是否结束 */

```

```

    {
        if(c == '\n')biosprint(0, '\r', 0); /* 若为换行符则加一回车符送打印机 */
        biosprint(0, c, 0);                /* 向打印机输出字符或汉字 */
    }
}

```

有关打印机的使用技巧请见本书第二部分第十一章“Turbo C 的高级打印技术”。

### 8.2.3 DOS 软中断功能调用函数 `intdos()`

DOS 的软中断(21 中断)具有强大的功能, Turbo C 用函数 `intdos()` 可直接访问这些系统调用。 `intdos()` 的调用格式为

```
int intdos(union REGS *in__regs, union REGS *out__regs);
```

该函数表示用 `in__regs` 指向的联合中的内容所确定的 DOS 系统调用, 执行一次 DOS 21 中断, 并将结果放入 `out__regs` 指向的联合中, 该函数的头文件为 `dos.h`。

联合 REGS 的定义如下:

```

union REGS{
    struct WORDREGS x;
    struct BYTEREGS n;
};

```

其中结构 WORDREGS 和 BYTEREGS 的结构如下:

```

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

```

下面举例来说明该函数的用法:

**例 8-8:** 用 `intdos()` 函数直接调用 DOS 21 中断的功能 42 读取系统日期。该功能调用前 `ah=0x2a`; 调用后, 返回值 `cx`=年, `dh`=月, `dl`=日, `al`=星期几(0=星期天, 1=星期一, ...), 有关这方面的知识请参阅《DOS 功能调用大全》。

```

#include <dos.h>
main( )
{
    union REGS in, out; /* 定义结构变量 */
    in.h.ah = 0x2a;      /* 设置 DOS 21 类中断的功能 42 */
    intdos(&in, &out);   /* 执行中断 */
    printf("Year = %4d\n", out.x.cx); /* 输出年 */
    printf("Month = %2d\n", out.h.dh); /* 输出月 */
    printf("Day = %2d\n", out.h.dl);  /* 输出日 */
}

```

```

    printf("Weekday = %ld\n",out.h.ai);    /* 输出星期几 */
    getch( );
}

```

此程序的运行结果得到系统的日期。

## 8.2.4 BIOS 和 DOS 软中断调用函数 int86( )

BIOS 除了打印机 ( 17H 中断 ) 和键盘 ( 16H 中断 ) 中断之外, 还有显示器驱动 (10H 中断)、软盘服务 ( 13H 中断 ) 等中断。Turbo C 提供了关于这些中断调用的函数 int86 ( )。其使用格式为:

```
int int86(int int_num,union REGS *in_regs,union REGS *out_regs);
```

其中 int\_num 为 BIOS 的软中断, 联合 REGS 与上节中所述相同。该函数用 in\_regs 指向的联合变量中的内容所确定的系统调用, 执行一次 int\_num 规定的软中断, 并将结果放入 out\_regs 指向的联合中, 该函数的头文件在 dos.h 中。

显然只要将 int\_num 定为 0x21, 则该函数与 intdos( ) 函数功能相同。

例 8-9: 将例 8-8 的程序用 int86( ) 函数调用变成:

```

#include <dos.h>
main( )
{
    union REGS in,out;
    in.h.ah=0x2a;
    int86(0x21,&in,&out);    /* 执行中断 */
    printf("Year = %4d\n",out.x.cx);
    printf("Month = %2d\n",out.h.dh);
    printf("Day = %2d\n",out.h.dl);
    printf("Weekday = %ld\n",out.h.ai);
    getch( );
}

```

## 8.2.5 其它的系统调用函数

这类函数的头文件都是 dos.h。

一、delay( ) 和 sleep( ) 函数

delay( ) 函数的调用格式为:

```
void delay(unsigned int milliseconds);
```

sleep( ) 函数的调用格式为:

```
void sleep(unsigned int seconds);
```

这两个函数均表示将系统挂起, 暂停一段时间。delay( ) 函数中 milliseconds 以毫秒为单位, sleep( ) 函数中 seconds 以秒为单位。如要暂停执行程序 1 秒, 可写成

```
delay(1000);
```

或

```
sleep(1);
```

## 二、sound( )和nosound( )函数

这两个函数的调用格式为:

```
void sound(unsigned int frequency);
```

```
void nosound(void);
```

sound( )函数按 frequency 所规定的频率(单位为 Hz), 接通扬声器发声。

nosound( )函数关闭扬声器。

### 例 8-10:

```
#include <dos.h>
```

```
main( )
```

```
{
```

```
    int i;
```

```
    for(i = 100; i < 200; i = i+2)
```

```
    {
```

```
        sound(i);          /* 发出不同频率的声音 */
```

```
        delay(100);        /* 延时 100 毫秒 */
```

```
    }
```

```
    nosound( );           /* 关掉声音 */
```

```
}
```

注:

- 用了 sound( )函数后, 必须用 nosound( )函数关掉。否则扬声器将一直响下去, 即使退出程序也不会停止。

- 一般在 sound( )函数后加一个 delay( )函数, 可以获得连续的声音。

## 三、getdate( )和 gettime( )函数

这两个函数用来读取系统的日期和时间, 其调用格式为:

```
void getdate(struct date *d);
```

```
void gettime(struct time *t);
```

结构 date 和 time 的定义包含在 dos.h 中, 它们的定义如下:

```
struct date {
```

```
    int da__year;          /* 年 */
```

```
    char da__day;          /* 日 */
```

```
    char da__mon;          /* 月 */
```

```
};
```

```
struct time {
```

```
    unsigned char ti__min;  /* 分 */
```

```
    unsigned char ti__hour; /* 时 */
```



```

        unsigned char ti_hund;    /* 百分秒 */
        unsigned char ti_sec;    /* 秒 */
    };

```

下面程序读取系统日期和时间并在屏幕上显示。

#### 例 8-11:

```

#include <conio.h>
#include <dos.h>
main( )
{
    char stime[20],sdate[20];
    struct date *dos__date;    /* 定义日期结构变量指针 */
    struct time *dos__time;    /* 定义时间结构变量指针 */
    getdate(dos__date);        /* 取系统日期 */
    gettime(dos__time);        /* 取系统时间 */
    sprintf(sdate,"%d-%d-%d",dos__date->da__year,
        dos__date->da__mon,dos__date->da__day);    /* 存系统日期 */
    printf("Today is %s\n",sdate);    /* 输出日期 */
    sprintf(stime,"%d:%d:%d",dos__time->ti__hour,
        dos__time->ti__min,dos__time->ti__sec);    /* 存系统时间 */
    printf("Time is %s\n",stime);    /* 输出时间 */
    getch( );
}

```

### 8.3 字符串函数和数字字符串与数值的转换函数

#### 8.3.1 字符串函数

本节介绍的函数在未指明时，其头文件均为 string.h。

```

char *strcpy(char *str1,char *str2);
char *strcat(char *str1,char *str2);
int strcmp(char *str1, char *str2);

```

函数 strcpy( )把字符串指针或数组 str2 的内容复制到 str1 中。str2 必须是一个指向空(NULL)结尾的指针。函数返回指向 str1 的指针。一般给字符串数组赋值均使用此函数。

例如:

```

char str[20];
strcpy(str,"M1724 打印机");

```

函数 strcat( )把字符串 str2 连接到 str1 的后面重新存入 str1，并以空(NULL)结束。其中 str1 和 str2 必须以空结尾。连接之后，原来作为 str1 结尾的空结束符被 str2 的第一个字符覆盖了。在整个操作中原来的 str2 的内容未被修改。

#### 例 8-12:

```

include <stdio.h>
include <string.h>
main( )
{
    char s[20], *str;      /* 定义字符串变量 */
    strcpy(s, "你好!");    /* 给数组字符串变量赋值 */
    str = "王先生";        /* 给指针字符串变量赋值 */
    strcat(s, str);        /* 连接两个字符串内容重新存入 s 中 */
    puts(s);               /* 将字符串 s 输出 */
}

```

注:

● 该函数在操作时不作边界检查, 因此事先应保证 str1 有足够大的空间以存放它的原来内容加上 str2 的内容。

函数 strcmp( ) 比较两个以空结尾的字符串 str1 和 str2 的内容, 可根据返回值判断这两个字符串的大小。所谓字符串的大小是指按字典顺序进行比较。

返回值	含 义
< 0	str1 小于 str2
= 0	str1 等于 str2
> 0	str1 大于 str2

```

char *strncpy(char *str1, char *str2, int count);
char *strncat(char *str1, char *str2, int count);
int strncmp(char *str1, char *str2, int count);

```

函数 strncpy( ) 用于把 str2 所指向的字符串中的 count 个字符, 拷贝到由 str1 所指向的字符串中, 其中 str2 必须是一个指向以空结束的字符串指针。若 str2 所指字符串少于 count 个字符, 则在 str1 结尾处加空, 直到拷贝完 count 个字符。相反若 str2 所指的字符串多于 count 个字符, 则 str1 字符串不以空结束。

函数 strncat( ) 把 str2 所指向的字符串中不超过 count 个字符, 连接到 str1 所指向的字符串后面。字符串 str2 未作变化。在使用这个函数之前, 必须保证 str1 足够长, 以便能够存放它的原来内容加上 str2 的内容。因为该函数在操作时不作边界检查。

函数 strncmp( ) 比较两个以空结尾长度均不超过 count 个字符的字符串。若其中某个字符串少于 count 个字符, 则当遇到第一个 NULL 时比较结束。根据返回值判断比较结果。

返回值	含 义
< 0	str1 小于 str2
= 0	str1 等于 str2
> 0	str1 大于 str2

```

unsigned strlen(char *str);

```

函数 strlen( ) 用来计算指针 str 所指向的, 以空(NULL)结束的字符串的长度, 返回的长度值结束符 NULL 不计在内。

```
char *strlwr(char *str);
char *strupr(char *str);
int tolower(int ch);
int toupper(int ch);
```

函数 `strlwr()` 把 `str` 所指向的字符串取成小写字母,并将返回的小写字母字符串重新存入 `str` 中。

函数 `strupr()` 把 `str` 所指的字符串取成大写字母,并将返回的大写字母字符串重新存入 `str` 中。

函数 `tolower()` 的头文件为 `cctype.h`。它返回 `ch` 的小写字母,并将返回的小写字母重新存入 `ch` 中,如果 `ch` 不是字母则相当于未操作。

函数 `toupper()` 的头文件为 `cctype.h`。它返回 `ch` 中的大写字母,并将返回的大写字母重新存入 `ch` 中,如果 `ch` 不是字母则相当于未操作。

### 8.3.2 数字字符串与数值的转换函数

对于数字,可以是数值形式,也可以是字符串形式。在实际应用中,两者常常要进行转换。`Turbo C` 提供了一些函数可以实现这种转换,这些函数的头文件为 `stdlib.h`,下面分别进行介绍。

```
int atoi(char *str);
long atol(char *str);
double atof(char *str);
```

函数 `atoi()` 将 `str` 所指向的字符串转换成整型数并返回。该字符串必须包含一个有效的整型数,否则返回 0。

函数 `atol()` 将 `str` 所指向的字符串转换成长整型数并返回。该字符串中必须包含一个有效的长整型数,否则返回 0。

函数 `atof()` 将 `str` 所指向的字符串转变成为一个双精度浮点数并返回,该字符串必须包含一个有效的浮点数,否则返回 0。

```
char *itoa(int num, char *str, int radix);
char *ltoa(long num, char *str, int radix);
```

函数 `itoa()` 把整型数 `num` 按 `radix` 规定的进位制转换成等价的字符串,并把结果存放在由 `str` 所指向的字符串中,其中 `radix` 的范围为 2~36。

函数 `ltoa()` 把长整型数 `num` 按 `radix` 规定的进位制转换成等价的字符串,并把结果存放在由 `str` 所指向的字符串中,其中 `radix` 的范围为 2~36。

注:

- 这两个函数在使用之前,字符串 `str` 一定要足够长,这样方可保存转换后的结果。

- `Turbo C` 并未提供将浮点数转换成字符串的函数,但提供了 `sprintf()` 函数。该函数可以将不同数据类型(包括整型数、浮点数和字符串等)按格式化规定转换成一个字符串拷贝到某一变量中。关于 `sprintf()` 函数的用法请看第七章 7.2.8 节,这里不再重复。

下面的例子运行时在屏幕上显示 ASCII 码字符的八进制、十进制、十六进制 ASCII

值。

#### 例 8-13:

```
#include <stdio.h>
main( )
{
    char s8[4],s10[4],s16[4];
    int i;
    for(i=0;i<=255;i++)
    {
        itoa(i,s8,8);      /* 将 i 按八进制转换成字符串 */
        itoa(i,s10,10);     /* 将 i 按十进制转换成字符串 */
        strcpy(itoa(i,s16,16)); /* 将 i 按十六进制转换成字符串并将其中
                                的字母转换成大写 */
        printf("%c%10s%10s%10s\n", i,s8,s10,s16);
    }
}
```

### 8.4 动态内存分配、过程控制和数学运算函数

#### 8.4.1 动态内存分配函数

有时数组变量定义怎样的长度或者指针变量初始化时定多大的空间事先并不知道，而 Turbo C 又不能象 BASIC 那样动态定义数组，即先求出数组长度再定义，而必须在函数执行语句前就定义变量。往往定义的范围总是比实际使用的大，这样势必造成内存空间的浪费。Turbo C 提供了一些有关动态分配内存的函数，可以根据需要分配内存空间，而且在使用完毕时又可用相应函数将这些内存释放，这样使内存空间得到充分利用。本节介绍几个较常用的函数，其头文件均为 stdlib.h。

```
void * malloc(unsigned size);
void * calloc(unsigned num, unsigned size);
void * realloc(void * ptr, unsigned newsize);
void free(void * ptr);
```

函数 malloc( ) 得到指向大小为 size 个字节的内存空间的首字节指针，由它分配内存区域供用户使用。当返回值不为空指针时，分配成功，否则说明没有足够的内存，如果试图使用该指针将会破坏系统。所以最好在分配完之后测试是否分配成功。

函数 calloc( ) 得到指向大小为 num \* size 个字节的内存空间的首字节指针，由它分配的内存区域供用户使用。该函数常用来为具有 num 个元素，每个元素 size 个字节长度的数组分配空间。当返回值不为空指针时，分配成功，否则说明没有足够的内存，如果试图使用该指针将会破坏系统。所以最好在分配完之后测试是否分配成功。

函数 realloc( ) 把由 ptr 所指向的已分配的内存大小变成由 newsize 所确定的新的大小。newsize 可以大于或小于原先的值。当用 realloc( ) 函数来增加内存大小时，最好检测返回指针是否为空。如果是空指针，说明没有 newsize 大小的内存空间。否则若不为空，

原来的信息将会拷贝到新块中。当用 `realloc()` 函数减小内存时不会出现上述问题。

函数 `free()` 用来释放 `ptr` 所指向的内存以使这些内存成为再分配时的可用内存。`free()` 函数只能释放以前由动态内存分配函数，如函数 `malloc()` 和 `calloc()` 所分配的内存。

有关动态分配内存和释放内存的例子本书第二部分将有许多，这里就不举例了。

#### 8.4.2 过程控制函数

所谓过程控制函数是指那些用来控制程序执行、终止或调用其它执行程序的函数。

`exit()`

`exit()` 函数的头文件是 `process.h`，其调用格式为：

```
void exit(int status);
```

该函数使程序的运行立即正常终止。同时将状态值 `status` 传递到调用过程。一般认为：如果状态 `status` 为 0，则程序正常终止；若为非 0 值，则说明存在执行错误。调用了 `exit()` 函数之后，将自动清除和关闭所有打开的文件。

`system()`

`system()` 函数的头文件是 `stdlib.h`，其调用格式为：

```
int system(char *str);
```

该函数把由 `str` 所指向的字符串作为命令传给 DOS 系统。当 DOS 系统执行完所规定的命令后又返回到调用函数。

利用 `system()` 函数可以运行其它扩展名为 `.EXE` 或 `.COM` 的文件以及 DOS 的命令。但当运行该函数中指定的程序或执行 DOS 的命令操作时，调用程序是被挂起的。因此如果程序较大时将会出错，但并不毁坏系统，只是不去执行罢了。例如下面一段程序先查看当前目录下的所有 `.EXE` 文件。等待输入可执行文件名。然后由 `system()` 函数去执行之，执行完毕后显示一些提示。

##### 例 8-14:

```
#include <stdio.h>
#include <stdlib.h>
main()
{
    char str[9];
    clrscr();
    system("dir /w *.exe");    /* 调用 DOS 系统的显示命令 */
    printf("Please input filename:");
    gets(str);                /* 键盘输入可执行文件名 */
    system(str);              /* 转去执行输入的可执行文件 */
    printf("The end");        /* 执行完后再返回到程序并显示提示 */
    getch();
}
```

利用 `system()` 函数可以执行 BASIC 程序(解释 BASIC)和 dBASE 程序。其方法如下所示:

调用 BASIC 程序: `system("GWBasic 文件名");`

调用 dBASE 程序: `system("dBASE 文件名");`

若要调用 FOXBASE, 由于 FOXBASE 本身占用内存较大, 一般不能运行。鉴于目前使用 FOXBASE 的用户非常多, 本书提出了一个简便的方法可以完成 Turbo C 和 FOXBASE 两者之间的调用。有关这部分内容请参阅第二部分第十章。

`spawnl()`

`spawnl()` 函数的头文件是 `process.h`, 其调用格式为:

`spawnl(int mode, char *fname, char *arg0,...,char *argN, NULL);`

该函数用于从调用函数(父过程)中执行另一个程序(子过程), 子过程的文件名由 `fname` 给出。如果子过程有命令行参数, 则分别由 `arg0` 到 `argN` 给出。另外, 参数 `mode` 决定了子过程的执行方式, `mode` 的取值如下所示。

符号常数	数值	含 义
P_WAIT	0	把父过程挂起直到子过程执行完毕
P_NOWAIT	1	子过程和父过程同时执行(Turbo C 未提供)
P_OVERLAY	2	子过程在内存中覆盖掉父过程

注:

- 该函数的最后一个参数必须是 `NULL`, 如果子过程文件无命令行参数, 则在参数项用一个 `NULL` 即可。
- 一个子过程还可以调用另一个子过程, 子过程允许嵌套的层数受程序大小和 RAM 容量的限制。
- 当进行 `spawnl()` 函数调用时, 已打开的文件在子过程中仍然保持打开。  
例如运行一个可执行文件 `menu.exe`, 执行完后返回父过程, 可写成:

`spawnl(P_WAIT, "menu.exe", NULL);`

`execcl()`

`execcl()` 函数的头文件是 `process.h`, 其调用格式为:

`execcl(char *fname, char *arg0,...,char *argN, NULL);`

该函数从调用程序(父过程)执行另一程序(子过程)。调用时子过程将覆盖父过程。  
将例 8-13 中的第二个 `system()` 函数改为:

`execcl(str, NULL);`

则变成:

**例 8-15:**

```
#include <stdio.h>
#include <process.h>
#include <stdlib.h>
```

```

main( )
{
    char str[9];
    clrscr( );
    system("dir / w *.exe");    /* 调用 DOS 系统的显示命令 */
    printf("Please input filename:");
    gets(str);                  /* 键盘输入可执行文件名 */
    execl(str,NULL);           /* 执行输入的可执行文件 */
    printf("The end");
    getch( );
}

```

执行完后屏幕上并不显示 The end，而直接退出，说明执行 execl( ) 函数指定的子过程时父过程被覆盖了。

### 8.4.3 数学运算函数

本书介绍一些常用的数学运算函数，其头文件均为 math.h。

```

double sin(double arg);
double cos(double arg);
double tan(double arg);

```

分别求出 arg(以弧度表示)的正弦值、余弦值和正切值。

```

int abs(int num);
int labs(long num);
double fabs(double num);

```

上面三个函数分别求整型数、长整型数和浮点数的绝对值。

```

double log10(double num);
double log(double num);

```

这两个函数分别返回 num 以 10 为底的对数和自然对数的值。

```

double poly(double x,int n, double c[]);

```

该函数计算一个系数为 c[0]到 c[n]的 x 的 n 次多项式。即计算

$$c[n]x^n + c[n-1]x^{n-1} + \dots + c[0]$$

的值。

```

double pow(double base, double exp);

```

该函数计算以 base 为底，以 exp 为指数的幂的值。

## 8.5 Turbo C 集成开发环境下程序的调试

Turbo C 的集成开发环境给用户编程和调试带来了极大方便，大大缩短了软件的开发周期。有关编译、链接、单步运行、断点设置等常用的调试方法在本书第一部分第一章已作过介绍，需要时可查看。本节主要讲一些 Turbo C 程序调试时的一般性错误，以供

读者在碰到这些错误时采取相应的措施纠正，从而完成整个软件的开发。

### 8.5.1 编译时的常见错误

1. 忘记函数后面的";"。

此时错误提示色条将停留在该语句下的一行，并显示：

Statement missing : in function <函数名>

2. 在宏指令如#include、#define等语句尾多加了";"号。

3. "{"和"}"、"("和")"、"/ \* "和" \* /"不成对匹配

色条将位于错误所在的行，并提示出有关括号缺漏的信息。

4. 没有用#include指令说明头文件。

对于库函数，在使用前必须说明其头文件，否则将出现有关该函数所使用的参数未定义的错误。

5. 误用了Turbo C保留的关键字作为标识符，此时将提示定义数据类型太多的错误。

6. 误将变量定义语句放在了执行语句后面。此时会提示语法错误。

7. 使用了未定义的变量，此时屏幕显示：

Undefined symbol '<变量名>' in function <函数名>

8. 警告性错误太多。

忽略这些警告性错误并不影响程序的执行和结果。编译时当警告性错误数目大于某一规定值时(缺省为100)便退出编译器，这时应将集成开发环境Options/Compiler/Errors中的有关警告性错误的检查开关为改置off。

9. 将关系符"=="误用作赋值号"="。此时屏幕显示

Lvalue required in function <函数名>

### 8.5.2 链接时的常见错误

1. 将Turbo C库函数名写错。

这种情况在链接时将会认为此函数是用户自定义函数。此时屏幕显示：

Undefined symbol '<函数名>' in module <程序名>

2. 多个文件链接时，没有在Project/Project name中指定项目文件(.prj文件)，此时出现找不到函数的错误。

3. 用户函数在说明和定义时类型不一致。

4. 由程序调用的用户函数没有定义。

### 8.5.3 运行时的常见错误

1. 格式化输入输出时，规定的类型与变量本身的类型不一致。

例如：

```
float f;  
printf("%d",f);
```

2. scanf( )函数中将变量地址写成变量。



例如:

```
int i;  
scanf("%d",&i);
```

3. 循环语句中的循环控制变量在每次循环里未进行修改,使循环成为无限循环。

4. switch 语句中没有使用 break 语句。

5. 多层条件语句的 if 与 else 不配对。

6. 将赋值号 "=" 误用作关系符 "=="。

7. 用动态内存分配函数 malloc( )或 calloc( )分配的内存区使用以后,未用 free( )函数释放,会导致函数前几次调用正常,而后而调用时发生死机现象,不能返回操作系统。其原因是因为没有空间可供分配,而占用了操作系统在内存中的某些空间。

8. 指针型变量使用不当。有时对于指针型变量未分配空间就加以使用,这将会给系统带来潜在的危险。常常在程序运行时前几次调用该函数正确,再调用就会发生死机现象。

9. 使用了动态分配内存不成功的指针,造成系统破坏。

10. 对文件操作时,没有在操作结束后及时关闭打开的文件。

11. 误认为数组的下标是以 1 开始,或者数组下标越界。当越界时会引起子函数不能正常返回,也会影响 malloc( ), free( )等内存管理函数的正常使用,有时可能发生死机现象。

12. 定义全程变量太多或局部变量太多。

13. 路径表达错误。例如将"C:\TC"误写成"C\TC"。

14. 字符串结束符使用不当。一般字符数组中存放的字符串应以空符(NULL)结束,如果没有结束符,将不认为字符串结束,使用时不注意就会引起数据错误。例如定义一个字符数组 char s[30],第一次使用到下标 25,而第二次只使用到下标 20,如果在正常的字符操作后没有语句 s[21]='\0',这样该字符串中的值就不正确,因为它保持了上次下标从 21 到 25 的字符。另外,还应注意 Turbo C 对数组变量并不作边界检查。

本节所述各种错误只是一些很常见的,即便是这些常见的错误,如果有许多种同时出现,也很难找到出错的原因,对一些意想不到的错误就更不好判断了。因此就需要读者反复练习,自己亲自动手上机调试。在调试过程中,多用几个 printf( )函数, getch( )函数和集成开发环境中提供的运行到光标处及单步运行等方法,使出错误区域逐渐缩小,有利于发现并改正错误。

## 8.6 Turbo C 的命令行编译

Turbo C 提供了一个编译、运行和调试一体化的集成开发环境,使用户开发程序非常方便。但也有例外,如要将 Turbo C 程序与由它调用的其它语言(如汇编)程序进行编译和链接,在集成开发环境下进行就不太方便了。另外,对于在行间嵌入汇编语句的 Turbo C 程序就不能用集成开发环境进行编译,这时必须使用命令行编译。

所谓命令行编译是指在 DOS 状态下,用 Turbo C 的编译程序(TCC.EXE)生成一个可执行文件(.EXE)。

Turbo C 提供的命令行编译程序为 TCC.EXE,其使用格式为:

TCC [选择项 1 选择项 2...选择项 N] 文件名 1 文件名 2...文件名 N

这里的选项是编译程序或链接程序的选项。文件名是源文件.C、目标文件.OBJ或库文件.LIB。若未指定扩展名，Turbo C 将按源文件处理。

在没有指定只进行编译不进行链接时，TCC 编译完成后将自动链接。

在链接过程中，Turbo C 将自动链接其标准库，用户不必给出标准库名。另外，所有编译程序的选项前都有一个“-”号，并且选项对大小写是有区别的。下面是一些常用的选项及其意义。

选择项	含义
-c	只编译成.OBJ文件
-B	编译行间嵌有汇编语句的程序
-f	使用浮点仿真
-L	指定库的路径(xxx为路径)
-S	输出一个调用汇编模块的格式
-c	只编译不链接
-Ixxx	指定包含文件的路径(xxx为路径)
-ms *	使用小型存储模式
-mt	使用微型存储模式
-w	显示警告错误
-w-	不显示警告错误
-nxxx	指定输出目录(xxx为路径)
-exxx	指定执行文件名(xxx为文件名)

例如：

```
tcc -ms -cFILE -Lc:\tc\lib FILE1 FILE2.OBJ graphics.lib
```

该命令含义是：编译 FILE1.c 并与 FILE2.OBJ 和 graphics.lib 链接生成小型存储模式的 FILE.EXE 文件。其中 graphics.lib 的路径为 c:\tc\lib。

注：

- -ms 选项为 Turbo C 的缺省选择。
- 选项-B 和-S 的用法将在第十三章讲述 Turbo C 与汇编语言的接口时介绍。
- 如果使用了选项-c，则只将源文件编译成.OBJ 目标文件，若要链接成可执行文件，还应使用 Turbo C 提供的链接程序 TLINK.EXE。链接程序的使用格式为：

tlink[/选择项] OBJ 文件名,EXE 输出文件名,MAP 映象文件名,LIB 库文件名  
鉴于这种方法不常用，因此本书不对 TLINK.EXE 的使用方法作详细介绍。

如果要将产生的 OBJ 目标模块放入 Turbo C 相应的库中，供以后直接使用，可用 Turbo C 提供的库管理程序 TLIB.EXE。例如向库中增加一个目标模块可写成：

```
tlb LIB 库文件名+OBJ 文件名
```

从库中去掉一个目标模块可写成：

```
tlb LIB 库文件名-OBJ 文件名
```

## 第二部分 Turbo C 2.0 应用技术专题

本部分主要讲述 Turbo C 2.0 的应用技术, 这些技术非常实用, 包括: 西文方式下汉字的显示、与 FOXBASE(dBASE)数据库的接口、高级打印、菜单设计以及与汇编语言的接口等内容。

### 第九章 西文状态下的汉字显示技术

第八章已经讲过, 在汉字操作系统下用 `printf()`、`puts()` 等函数可以在屏幕上显示汉字。但有时用户并不希望在汉字操作系统下运行程序, 因为在汉字操作系统下, 汉字库将占用较大的内存, 如果应用程序比较大, 则可能导致程序无法加载, 这对于内存较小的计算机更为严重。另外, 汉字操作系统下显示的汉字仅为  $16 \times 16$  点阵, 而且色彩和显示方式都比较单调。针对这个问题本章介绍在西文操作系统下的汉字显示技术。当然, 这种技术在中文操作系统下同样可以使用。使用这种方法输出汉字, 可以达到尽善尽美的程度。

本章所介绍的技术包括在西文状态下显示  $16 \times 16$  点阵汉字和  $24 \times 24$  点阵汉字, 对  $24 \times 24$  点阵汉字按任意倍数放大显示。同时可以使汉字在屏幕上任意位置, 以各种不同的颜色、水平方向或垂直方向显示。另外, 也可以选择不同字体和是否将汉字逆时针旋转  $90$  度显示。下面详细介绍这些功能的实现方法。

#### 9.1 西文状态下显示 $16 \times 16$ 点阵汉字

##### 9.1.1 $16 \times 16$ 点阵汉字字模的存储格式

在汉字操作系统中, 有一个  $16 \times 16$  点阵的汉字库, 主要用于屏幕显示。在汉字系统加载时, 该字库一般驻留在内存(有的汉字系统可以由用户选择是否驻留, 但 UC DOS 固定驻留)。字库中的汉字按  $16 \times 16$  点阵模式存储, 即每个汉字由  $16 \times 16 = 256$  个点组成, 占用  $16 \times 2 = 32$  个连续的字节单元。字节的每一个位表示一个点的属性(1 表示亮点, 0 表示暗点), 连续的 2 个字节组成该汉字字模的一个行。32 个字节的排列顺序如图 9-1 所示。

##### 9.1.2 西文状态下显示 $16 \times 16$ 点阵汉字的实现

计算机是以编码的方式来处理和使用字符的。西文字符采用一个字节表示, 即 ASCII 码, 一般只用七位来表示 128 个字符, 而把最高位用作奇偶校验(或者不用)。我国国标规定汉字用内码表示, 内码为两个字节。为了保证中西文兼容, 也就是说汉字系统的

内码必须同时允许 ASCII 码和汉字的使用，两者之间不应发生冲突。目前规定每个字节只用七位，若两个字节的最高位均为 1，则该字符为汉字。

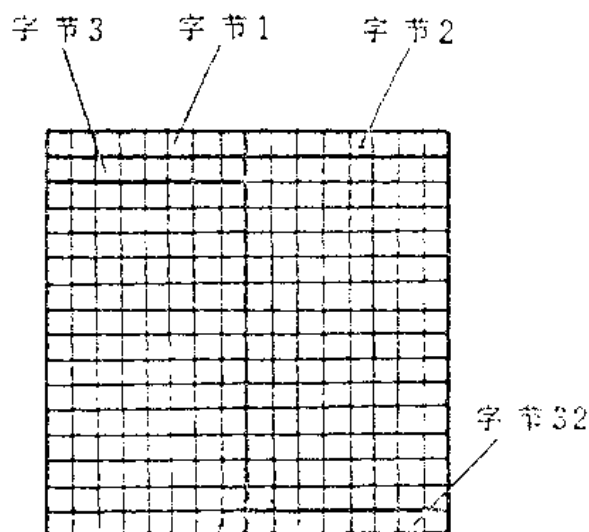


图 9-1 16×16 点阵汉字字模的结构

国标对汉字库的结构作了统一规定，即将字库分成若干个区，每个区有 94 个汉字，每一个汉字在字库中有确定的区和位，因此每一个汉字各有一个区位码，知道了区位码也就相当于知道了汉字在字库中的位置。由于汉字的内码与区位码有一定的关系，所以只要通过汉字的内码就可得到该汉字的区位码，从而也可以获得该汉字的字模，然后在西文状态的图形模式下，读取字模中每一个字节的每一位，按画点的方式就能在屏幕上显示出汉字。而内码在中文操作系统下输入汉字时，就已存入程序，即使退出汉字系统，它也存在。

设某一汉字的内码为  $ddff$ ，其中  $dd$  表示区内码， $ff$  表示位内码，则  $dd-0xa1$  为该汉字的区码， $ff-0xa1$  为该汉字的位码。如果把组成一个 16×16 点阵汉字的 32 个字节作为一条记录，确定汉字在字库中的位置也就相当于确定汉字的记录号  $rec$ 。计算汉字的记录号可用下面的公式：

$$rec = (dd - 0xa1) * 94 + (ff - 0xa1)$$

得到记录号后乘以 32 则为该汉字在 16×16 点阵字库中字模第一个字节的位置。只要用 Turbo C 有关文件的函数定位文件位置指示器于按二进制方式打开的字库文件中该位置，连续读取 32 个字节，就可以得到这个汉字的字模。再用有关位操作和循环的语句，对每个字节的每一个位进行判断，如果某位为 1，则按设置的颜色在屏幕上相应位置画一个点；若某位为 0，则该位置不画点，这样就可以显示出汉字。

下面的例程在屏幕上以红色显示 16×16 点阵汉字。程序中使用的字库为希望电脑公司出版的汉字操作系统 UC DOS 2.0 所提供的 16×16 点阵字库(以后使用的字库也均为 UC DOS 2.0 字库,不再说明)。字库名为 `cclib.dat`，存放在硬盘的根目录下。

例 9-1:

```
#include <graphics.h>
```

```

#include <stdio.h>
#include <fcntl.h>
int openhz(void);          /* 打开字库文件函数的说明 */
int puthz16(int,int,int,int,char *); /* 显示汉字函数的说明 */
int getbit(unsigned char,int); /* 读字节位函数的说明 */
int handle;               /* 定义文件描述字为全程变量 */
main( )
{
    int gdriver,gmode;
    gdriver = DETECT;
    registerbgidriver(EGAVGA_driver); /* 建立独立图形运行程序 */
    initgraph(&gdriver,&gmode,"c:\\tc"); /* 图形模式初始化 */
    setbkcolor(BLUE); /* 设置屏幕背景色 */
    setcolor(LIGHTRED); /* 设置作图色 */
    openhzk( ); /* 调用了函数打开字库 */
    puthz16(230,200,10,12,"欢迎使用本书"); /* 调用了函数显示汉字 */
    close(handle); /* 关闭打开的字库文件 */
    getch( ); /* 等待按任一键 */
    closegraph( ); /* 退出图形模式 */
    exit(0); /* 退出程序 */
}
int openhzk( )
{
    handle = open("c:\\cclib.dat",O_RDONLY|O_BINARY); /* 以二进制方式打开
                                                    16×16点阵字库文件只读 */
    if(handle == -1) /* 文件打开不成功显示信息并退出 */
    {
        cputs("Error on open cclib.dat");
        getch( );
        closegraph( );
        exit(1);
    }
}
int puthz16(int x,int y,int z,int color,char *p)
{
    unsigned int i,c1,c2,f=0; /* 定义局部变量 */
    int i1,i2,i3,rec;
    long l;
    char by[32];
    while((i = *p++) != 0) /* 设置循环直到汉字显示完为止 */
    {
        if(i > 0xa1) /* 判断是否为汉字内码 */

```

```

        if(f == 0)                                /* 若是汉字内码再判断是否为区内码 */
        {
            c1 = (i-0xa1)&0x07f;                  /* 得到汉字区码 */
            f = 1;
        }
        else
        {
            c2 = (i-0xa1)&0x07f;                  /* 得到汉字位码 */
            f = 0;
            rec = c1 * 94 + c2;                    /* 得到汉字的记录号 */
            l = rec * 32L;                          /* 得到汉字在字库中的位置 */
            lseek(handle,l,SEEK_SET);               /* 文件位置指针定位到汉字字模的首字节 */
            read(handle,by,32);                    /* 读取该汉字字模的 32 个字节 */
            for(i1 = 0; i1 < 16; i1++)              /* 显示字模垂直方向 16 个点 */
                for(i2 = 0; i2 < 2; i2++)          /* 字模水平方向两个字节 */
                    for(i3 = 0; i3 < 8; i3++)      /* 水平方向每个字节的 8 位 */
                        if(getbit(by[i1 * 2 + i2], 7 - i3)) /* 判断该位是否为 1 */
                            putpixel(x + i2 * 8 + i3, y + i1, color); /* 是为 1 则在屏幕相应位置画点 */
                        x = x + 16 + z;              /* 显示下一个汉字的 X 坐标 */
        }
    }
    return(x);                                     /* 返回显示结束时的 X 坐标 */
}

int getbit(unsigned char c, int n)
{
    return((c >> n) & 1); /* 将字节中的某位移到字节最低位并屏蔽掉其它 7 位 */
}

```

该程序运行后在屏幕上以 16×16 点阵汉字显示“欢迎使用本书”。

上述程序的关键是 puthz16( ) 子函数，该函数有 5 个形式参数，它的调用格式为：

```
int puthz16(int x, int y, int z, int color, char *p);
```

其中 x, y 分别为汉字字模左上角对应的屏幕上象素的坐标；z 为两个汉字的间距，以象素为单位；color 为显示汉字的颜色值；p 为指向要显示汉字的字符串指针。函数返回显示完汉字串后游标的 x 坐标。

函数中首先根据读取的字符编码是否大于 0xa1 判断该字符是不是为汉字，若是汉字则读取两次才能得到一个汉字内码。第一次读取汉字的区内码，第二次读取汉字的位内码，再由汉字的内码确定该汉字记录号，用 lseek( ) 函数定位文件位置指针，用 read( ) 函数读取 32 个字节可得到该汉字的字模，最后对 32 个字节逐一判断每一位是否为 1，来确定是否在屏幕上相应位置用 putpixel( ) 函数画点。判断位时采用字节右移的办法，将该位移到字节的最低位再与 1 相与而屏蔽掉其它 7 位，由所得字节的值是否为 1 就可知道该位是否为 1。

上面的程序只能水平方向显示汉字,若对其略加改进,就可以选择是按水平方向显示还是按垂直方向显示。例9-2还可以选择是否将汉字逆时针方向旋转90度显示。

**例9-2:**

```
#include <graphics.h>
#include <stdio.h>
#include <fcntl.h>
int openhz(void);
int puthz16(int,int,int,int,char *,char,int);
int getbit(unsigned char, int);
int handle;
main( )
{
    int gdriver,gmode;
    unsigned char *f="欢迎使用本书";
    gdriver=DETECT;
    registerbgidriver(EGAVGA_driver);
    initgraph(&gdriver,&gmode,"c:\\tc");
    setbkcolor(BLUE);
    setcolor(LIGHTRED);
    openhzk( );
    puthz16(230,200,10,12,f,'v',0);
    puthz16(230,200,10,12,f,'h',0);
    puthz16(230,180,10,10,f,'v',90);
    puthz16(230,180,10,10,f,'h',90);
    close(handle);
    getch( );
    closegraph( );
    exit(0);
}
int openhzk( )
{
    handle=open("c:\\cclib.dat",O_RDONLY|O_BINARY);
    if(handle == -1)
    {
        cputs("Error on open cclib.dat");
        getch( );
        closegraph( );
        exit(1);
    }
}
int puthz16(int x,int y,int z,int color,char *p,char d,int angle)
```

```

{
    unsigned int i, c1, c2, f=0;
    int i1, i2, i3, rec;
    long l;
    char by[32];
    while((i = *p++) != 0)
    {
        if(i > 0xa1)                                /* 判断是否为汉字 */
            if(f == 0)
            {
                c1 = (i - 0xa1) & 0x07f;            /* 取汉字区码 */
                f = 1;
            }
        else
        {
            c2 = (i - 0xa1) & 0x07f;                /* 取汉字位码 */
            f = 0;
            rec = c1 * 94 + c2;
            l = rec * 32L;
            lseek(handle, l, SEEK__SET);
            read(handle, by, 32);
            if(angle == 0)                            /* 如果汉字不旋转显示 */
                for(i1 = 0; i1 < 16; i1++)
                    for(i2 = 0; i2 < 2; i2++)
                        for(i3 = 0; i3 < 8; i3++)
                            if(getbit(by[i1 * 2 + i2], 7 - i3))
                                putpixel(x + i2 * 8 + i3, y + i1, color);
            if(angle == 90)                            /* 如果汉字旋转显示 */
                for(i1 = 0; i1 < 16; i1++)
                    for(i2 = 0; i2 < 2; i2++)
                        for(i3 = 0; i3 < 8; i3++)
                            if(getbit(by[i1 * 2 + i2], 7 - i3))
                                putpixel(x + i1, y - i2 * 8 - i3, color);
            if(d == 'h')                                /* 如果汉字串按水平方向显示 */
                x = x + 16 + z;
            if(d == 'v')                                /* 如果汉字串按垂直方向显示 */
            {
                if(angle == 0)
                    y = y + 16 + z;
                if(angle == 90)
                    y = y - 16 - z;
            }
        }
    }
}

```



```

    }
}
return(x);
}
int getbit(unsigned char c,int n)
{
    return((c>>n)&1);
}

```

例 9-2 对例 9-1 的子函数 puthz16( )作了修改, 例 9-2 的 puthz16( )函数调用格式为:

```
int puthz16(int x,int y,int z,int color,char *p,char d,int angle);
```

其中 x、y、z、color、p 五个变量的意义未变, 但增加了两个变量: d 和 angle。d 用来选择汉字串按水平方向显示还是按垂直方向显示, 当 d='h'时按水平方向显示, 当 d='v'时按垂直方向显示; angle 用来选择是否以逆时针方向旋转 90 度显示汉字, 当 angle=90 时表示是旋转, 当 angle=0 时不旋转。

## 9.2 西文状态下显示 24×24 点阵汉字

### 9.2.1 24×24 点阵汉字字模的存储格式

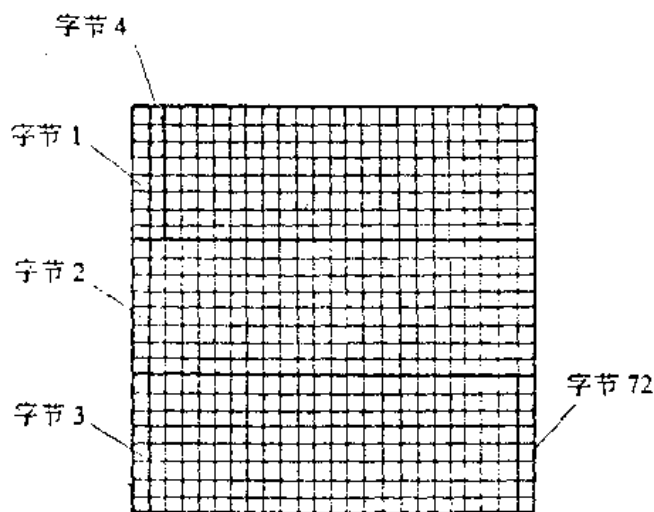


图 9-2 24×24 点阵汉字字模的结构

24×24 点阵字库主要用于打印。字库中每个汉字由  $24 \times 24 = 576$  个点组成, 占用  $24 \times 3 = 72$  个连续的字节单元, 字节的每一个位表示一个点的属性 (1 表示亮点, 0 表示暗点), 连续的 3 个字节表示该汉字字模的一个列。这 72 个字节的排列顺序如图 9-2 所示。

比较图 9-2 和图 9-1 可以看出这两种字模在存储顺序上是完全不同的。对于 16×16

点阵汉字按水平方向依次为字节 1、字节 2、.....、字节 32，而对于 24×24 点阵汉字按垂直方向依次为字节 1、字节 2、....、字节 72。

### 9.2.2 西文状态下显示 24×24 点阵汉字的实现

24×24 点阵汉字的内码与 16×16 点阵汉字的内码完全相同，也就是说汉字的区位码并不区分 16×16 点阵和 24×24 点阵字库，不同的只是每一个汉字字模的存储字节个数。因此确定 24×24 点阵汉字的记录号仍按与 16×16 点阵相同的公式计算。得到记录号后乘以 72 则为汉字字模第一个字节在字库中的位置。同样用 Turbo C 的 lseek( ) 函数定位文件位置指示器，用 read( ) 函数读取连续 72 个字节可得到该字模，再根据每个字节的每一个位是否为 1，判断在屏幕上画不画出点。这样就可以显示出 24×24 点阵汉字。请看下面例子。

#### 例 9-3:

```
#include <stdio.h>
#include <graphics.h>
#include <fcntl.h>
int openhz(void);
int puthz24(int,int,int,int,char * );
int getbit(unsigned char,int);
int handle;
main( )
{
    int gdriver,gmode;
    gdriver=DETECT;
    registerbgidriver(EGAVGA__driver);
    initgraph(&gdriver,&gmode,"c:\\tc");
    setbkcolor(BLUE);
    cleardevice( );
    openhzk( );
    puthz24(200,200,8,12,"欢迎使用本书");
    close(handle);
    getch( );
    closegraph( );
    exit(0);
}
int openhzk( )
{
    handle=open("c:\\clib",O_RDONLY|O_BINARY); /* 打开 24×24 点阵汉字库 */
    if(handle == -1)
    {
        puts("Error on open cclib.dat");
        getch( );
    }
}
```

```

        closegraph( );
        exit(1);
    }
}

int puthz24(int x,int y,int z,int color,char *p)
{
    unsigned int i,c1,c2,f=0;
    int i1,i2,i3,rec;
    long l;
    char by[72];
    while((i= *p++)!=0)
    {
        if(i>0xa1)                /* 判断是否为汉字 */
            if(f==0)
            {
                c1=(i-0xa1)&0x07f;    /* 得到汉字区码 */
                f=1;
            }
        else
        {
            c2=(i-0xa1)&0x07f;    /* 得到汉字位码 */
            f=0;
            rec=c1*94+c2;        /* 得到汉字的记录号 */
            l=rec*72L;
            lseek(handle,l,SEEK_SET); /* 文件指针定位于汉字字模第一个字节 */
            read(handle,by,72);    /* 读汉字字模的 72 个字节 */
            for(i1=0;i1<24;i1++)    /* 水平方向显示汉字 24 个点 */
                for(i2=0;i2<=2;i2++) /* 垂直方向三个字节 */
                    for(i3=0;i3<8;i3++) /* 每个字节的 8 位 */
                        if(getbit(by[i1*3+i2],7-i3))
                            putpixel(x+i1, y+i2*8+i3, color); /* 某位为 1 时画点 */
                        x=x+24+z;
        }
    }
    return(x);
}

int getbit(unsigned char c, int n)
{
    return((c>>n)&1);
}

```

本程序运行后在屏幕上以 24×24 点阵显示汉字“欢迎使用本书”。

该例子中子函数 puthz24( )的调用格式为:

```
int puthz24(int x,int y,int z,int color, char *p);
```

其中每个形式参数的含义与例 9-1 中 puthz16( )函数的形式参数完全相同。两个函数不同之处只在于该例使用的是 24×24 点阵字库,字体非常美观,而例 9-1 使用的是 16×16 点阵字库。

24×24 点阵字库有不同字体之分,UCDOS 2.0 拥有四个不同字体的字库,分别为 CLIB(宋体字)、CLIBK(楷体字)、CLIBF(仿宋体字)、CLIBH(黑体字)。只要打开不同字体的字库文件,就可按所选字体显示汉字,这一功能很容易实现。再加上能选择是水平显示还是垂直显示、是否旋转 90 度显示这些功能,就得到例 9-4 的程序。

#### 例 9-4:

```
#include<stdio.h>
#include<graphics.h>
#include<fcntl.h>
int puthz24(int,int,int,int,char *,char,int,char);
int getbit(unsigned char,int);
main( )
{
    int gdriver,gmode;
    gdriver=DETECT;
    registerbgidriver(EGAVGA__driver);
    initgraph(&gdriver,&gmode,"c:\\tc");
    setbkcolor(BLUE);
    cleardevice( );
    puthz24(230,250,8,12,"欢迎使用本书",'h',0,'s');
    puthz24(200,280,8,12,"欢迎使用本书",'v',0,'k');
    puthz24(230,230,8,10,"欢迎使用本书",'h',90,'h');
    puthz24(200,200,8,10,"欢迎使用本书",'v',90,'f');
    getch( );
    closegraph( );
    exit(0);
}
int puthz24(int x,int y,int z,int color,char *p,char d,int angle,char c)
{
    unsigned int i,c1,c2,f=0;
    int handle,i1,i2,i3,rec;
    long l;
    char by[72];
    if(c=='s')
        handle=open("c:\\clib",O_RDONLY|O_BINARY);
    if(c=='k')
```

```

        handle = open("c:\\clibk",O_RDONLY|O_BINARY);
if(c == 'h')
        handle = open("c:\\clibh",O_RDONLY|O_BINARY);
if(c == 'f')
        handle = open("c:\\clibf",O_RDONLY|O_BINARY);
while((i = *p++) != 0)
{
        if(i > 0xa1)
                if(f == 0)
                {
                        c1 = (i-0xa1)&0x07f;
                        f = 1;
                }
        else
        {
                c2 = (i-0xa1)&0x07f;
                f = 0;
                rec = c1 * 94 + c2;
                l = rec * 72L;
                lseek(handle,l,SEEK_SET);
                read(handle,by,72);
                if(angle == 0)
                        for(i1 = 0; i1 < 24; i1++)
                                for(i2 = 0; i2 < 2; i2++)
                                        for(i3 = 0; i3 < 8; i3++)
                                                if(getbit(by[i1 * 3 + i2], 7 - i3))
                                                        putpixel(x+i1, y+i2 * 8 + i3, color);
                if(angle == 90)
                        for(i1 = 0; i1 < 24; i1++)
                                for(i2 = 0; i2 < 2; i2++)
                                        for(i3 = 0; i3 < 8; i3++)
                                                if(getbit(by[i1 * 3 + i2], 7 - i3))
                                                        putpixel(x+i2 * 8 + i3, y-i1, color);
                if(d == 'h')
                        x = x + 24 + z;
                if(d == 'v')
                {
                        if(angle == 0)
                                y = y + 24 + z;
                        if(angle == 90)
                                y = y - 24 - z;
                }
        }
}

```

本程序运行后在屏幕上显示:

本例的子函数 puthz24( )的调用格式为:

除了在最后增加一个选择字型的参数以外，其它参数的意义与例 9-2 中 `puthz16()` 函数完全相同，这里不再重复。

### 9.3 西文状态下放大任意倍数显示不同字体的 24×24 点阵汉字

对汉字进行放大，包括水平方向放大和垂直方向放大。为了简单起见采用成倍数放大为例，即某一汉字水平方向放大一倍，相当于字模中每一位在水平方向变成两个点，这样水平方向的 24 个点就变成了 48 个点。同样，垂直方向放大一倍也相当于字模中每一位在垂直方向变成两个点，这样垂直方向 24 个点也变成 48 个点。

**例 9-5:**

-141-

```

main( )
{
    int gdriver,gmode,i,j,k;
    unsigned char *f="欢迎使用本书";
    unsigned char *s[]={"愿本书能给您有所帮助",
        "编者于一九九二年五月","实用编程技巧"};
    char c,*str;
    gdriver=DETECT;
    registerbgidriver(EGAVGA__driver);
    initgraph(&gdriver,&gmode,"c:\\tc"); /* 将屏幕初始化为图形模式 */
    setbkcolor(BLUE);
    cleardevice( );
    openhzk(c='k'); /* 选择打开 24×24 点阵楷体字库 */
    puthz24(10,100,8,10,4,4,f); /* 调用子函数显示汉字 */
    close(handle);
    openhzk(c='s'); /* 选择打开 24×24 点阵宋体字库 */
    puthz24(60,300,4,9,2,2,s[0]); /* 调用子函数显示汉字 */
    puthz24(180,400,4,2,1,1,s[1]);
    close(handle);
    for(i=3;i<=5;i++) /* 改变位置循环显示同一串汉字 */
    {
        openhzk(c='k');
        puthz24(15-i,95+i,8,12,4,4,f);
        close(handle);
        openhzk(c='s');
        puthz24(60,300,4,10+i,2,2,s[0]);
        puthz24(180,400,4,6+i,1,1,s[1]);
        close(handle);
        delay(500);
    }
    setbkcolor(7);
    cleardevice( );
    setcolor(RED);
    setttextstyle(1,0,9);
    outtextxy(50,70,"Turbo C 2.0");
    openhzk(c='k');
    puthz24(80,250,8,12,3,3,s[2]);
    close(handle);
    openhzk(c='o'); /* 选择打开 16×16 点阵字库 */
    puthz16(200,400,8,10,s[1]); /* 调用子函数显示汉字 */
    close(handle);
    sleep(5);
}

```

```

setbkcolor(13);
cleardevice( );
setcolor(GREEN);
settextstyle(3,0,9);
outtextxy(50,70,"Turbo C 2.0");
openhzk(c='h');
puthz24(80,250,8,14,3,3,s[2]);
close(handle);
openhzk(c='o');
puthz16(200,400,8,1,s[1]);
close(handle);
sleep(5);
setbkcolor(2);
cleardevice( );
setcolor(14);
settextstyle(4,0,9);
outtextxy(50,70,"Turbo C 2.0");
openhzk(c='f');
puthz24(80,250,8,1,3,3,s[2]);
close(handle);
openhzk(c='o');
puthz16(200,400,8,12,s[1]);
close(handle);
sleep(5);
closegraph( );
}
int openhzk(char c)      /* 打开不同字库函数 */
{
    if(c == 's')
        handle = open("c:\\\\clib",O_RDONLY|O_BINARY);
    if(c == 'f')
        handle = open("c:\\\\clibf",O_RDONLY|O_BINARY);
    if(c == 'k')
        handle = open("c:\\\\clibk",O_RDONLY|O_BINARY);
    if(c == 'h')
        handle = open("c:\\\\clibh",O_RDONLY|O_BINARY);
    if(c == 'o')
        handle = open("c:\\\\celib.dat",O_RDONLY|O_BINARY);
    if(handle == -1)
    {
        cputs("Error on open celib.dat");
        getch( );
    }
}

```



```

        closegraph( );
        exit(1);
    }
}

int puthz16(int x, int y, int z, int color, char *p)
{
    unsigned int i, c1, c2, f=0;
    int i1, i2, i3, rec;
    long l;
    char by[32];
    while((i = *p++) != 0)
    {
        if(i > 0xa1)
        {
            if(f == 0)
            {
                c1 = (i - 0xa1) & 0x07f;
                f = 1;
            }
            else
            {
                c2 = (i - 0xa1) & 0x07f;
                f = 0;
                rec = c1 * 94 + c2;
                l = rec * 32L;
                lseek(handle, l, SEEK__SET);
                read(handle, by, 32);
                for(i1 = 0; i1 < 16; i1++)
                for(i2 = 0; i2 < 2; i2++)
                for(i3 = 0; i3 < 8; i3++)
                if(getbit(by[i1 * 2 + i2], 7 - i3))
                {
                    putpixel(x + i2 * 8 + i3, y + i1, color);
                    x = x + 16 + z;
                }
            }
        }
    }
    return(x);
}

int puthz24(int x, int y, int z, int color, int m, int n, char *p)
{
    unsigned int i, c1, c2, f=0;
    int i1, i2, i3, i4, i5, rec;
    long l;
    char by[72];

```

```

while((i = *p++) != 0)
{
    if(i > 0xa1)
    if(f == 0)
    {
        c1 = (i - 0xa1) & 0x07f;
        f = 1;
    }
    else
    {
        c2 = (i - 0xa1) & 0x07f;
        f = 0;
        rec = c1 * 94 + c2;
        l = rec * 72L;
        lseek(handle, l, SEEK_SET);
        read(handle, by, 72);
        for(i1 = 0; i1 < 24 * m; i1 = i1 + m)
        for(i4 = 0; i4 < m; i4++)
        for(i2 = 0; i2 < 2; i2++)
        for(i3 = 0; i3 < 8; i3++)
        if(getbit(by[i1 / m * 3 + i2], 7 - i3))
        for(i5 = 0; i5 < n; i5++)
            putpixel(x + i1 + i4, y + i2 * 8 * n + i3 * n + i5, color);
        x = x + 24 * m + z;
    }
}
return(x);
}

int getbit(unsigned char c, int n)
{
    return((c >> n) & 1);
}

```

该例子中子函数 puthz16( ) 的调用格式为:

```
int puthz16(int x, int y, int z, int color, char *p);
```

其中每个形式参数的意义与例 9-1 中 puthz16( ) 函数的形式参数相同。

子函数 puthz24( ) 的调用格式为:

```
int puthz24(int x, int y, int z, int color, int m, int n, char *p);
```

其中形式参数 x、y、z、color、p 的意义与例 9-1 中函数 puthz16( ) 的形式参数相同。参数 m 表示显示的汉字水平放大倍数, n 表示垂直放大倍数。

例 9-5 运行时以不同大小、不同字体、不同颜色象“拉幕”一样显示汉字。如将该段

程序放到应用程序的开头，将会增色不少。

程序中最大的放大倍数为在水平方向和垂直方向各 4 倍，但是显示出的汉字可以看到较明显的锯齿，这主要因为字模点阵的组成点太少，如果改用较密的点阵字库，如  $40 \times 40$ 、 $48 \times 48$  点阵字库或更密，显示大号汉字的质量将明显改善。

## 第十章 与 FOXBASE(dBASE)接口技术

FOXBASE(或 dBASE)是一类数据库管理语言,应用非常广泛。然而,FOXBASE 却有一个非常明显的缺点就是图形功能差。另外,在使用 C 语言进行决策、模拟等过程中常常需要处理大量的数据,如果将数据的管理用 FOXBASE 语言实现,而用 Turbo C 读取 FOXBASE 数据库中的数据或直接在 FOXBASE 环境中以传递参数的方法调用 Turbo C 程序进行绘图、决策等,以充分发挥两种语言的优点。本章讲述有关这方面的技巧。

### 10.1 Turbo C 直接读取 FOXBASE 的数据库

#### 10.1.1 FOXBASE 数据库的结构

为了用 Turbo C 读取 FOXBASE 数据库中的数据,首先必须搞清楚数据库的结构。下面通过一个简单的数据库实例来进行说明。

假定要用一个数据库存储如下的表格数据:

姓名	年级	班级	数学	物理	化学	英语	生物
张三	高二	1	90.5	80	75.5	80	89
李四	高二	1	89	90	70	90	65

首先在 FOXBASE 环境下,建立一个名字为 SCORE.DBF 的数据库,其结构为:

```
Structure for database: A:\SCORE.DBF
Number of data records:      2
Date of last update   : 02/17/92

Field  Field Name      Type           Width      Dec
-----
1  NAME                Character      8
2  CLASS                Character      4
3  SUBCLAS              Character      1
4  MATH                 Numeric        4          1
5  PHYS                 Numeric        4          1
6  CHEM                 Numeric        4          1
7  ENGL                 Numeric        4          1
8  BIOL                 Numeric        4          1

* *   Total   * *                34
```

用 FOXBASE 的 LIST 命令列出数据库内容如下:

Record#	NAME	CLASS	SUBCLAS	MATH	PHYS	CHEM	ENGL	BIOL
1	张三	高二	1	90.5	80.0	75.5	80.0	89.0
2	李四	高二	1	89.0	90.0	70.0	90.0	65.0

在 DOS 系统下用 debug.com 来观察此数据库的内容, 只要键入:

debug score.dbf

就可进入 debug 状态并把 SCORE.DBF 装入到内存中。用 debug 的 R 命令显示寄存器的内容可以知道数据库文件被装入内存中的位置(由段寄存器和 IP 寄存器的显示值确定)和大小(由 BX 和 CX 寄存器的显示值确定), 再用 D 命令就可显示数据库的内容。

对于数据库 SCORE.DBF, 用 debug 命令后屏幕显示的内容如下:

屏幕显示:

```
AX=0000 BX=0000 CX=0166 DX=0000 SP=FFFF BP=0000 SI=0000 DI=0000
DS=307E ES=307E SS=307E CS=307E IP=0100 HV UP EI PL NZ NA PO NC
307E:0100 035C02 ADD BX,[SI+02] DS:0002=2F10
```

-d1001166

屏幕显示:

307E:0100	03 5C 02 11 02 00 00 00	00-21 01 22 00 00 00 00	.....
307E:0110	00 00 00 00 00 00 00 00	00-00 00 00 00 00 00 00	NAME.....C....
307E:0120	4E 41 4D 45 00 00 00 00	00-00 00 00 43 01 00 00 00	CLASS.....C....
307E:0130	08 00 00 00 00 00 00 00	00-00 00 00 00 00 00 00 00	SUBCLAS....C....
307E:0140	43 4C 41 53 53 00 00 00	00-00 00 00 43 03 00 00 00	.....C....
307E:0150	04 00 00 00 00 00 00 00	00-00 00 00 00 00 00 00 00	.....C....
307E:0160	53 55 42 43 4C 41 53 00	00-00 00 00 43 0D 00 00 00	.....C....
307E:0170	01 00 00 00 00 00 00 00	00-00 00 00 00 00 00 00 00	.....C....
307E:0180	4D 41 54 48 00 00 00 00	00-00 00 00 4E 0E 00 00 00	.....C....
307E:0190	04 01 00 00 00 00 00 00	00-00 00 00 00 00 00 00 00	.....C....
307E:01A0	50 48 59 53 00 00 00 00	00-00 00 00 4E 12 00 00 00	.....C....
307E:01B0	04 01 00 00 00 00 00 00	00-00 00 00 00 00 00 00 00	.....C....
307E:01C0	43 48 45 4D 00 00 00 00	00-00 00 00 4E 16 00 00 00	.....C....
307E:01D0	04 01 00 00 00 00 00 00	00-00 00 00 00 00 00 00 00	.....C....
307E:01E0	45 4E 47 4C 00 00 00 00	00-00 00 00 4E 1A 00 00 00	.....C....
307E:01F0	04 01 00 00 00 00 00 00	00-00 00 00 00 00 00 00 00	.....C....
307E:0200	42 49 4F 4C 00 00 00 00	00-00 00 00 4E 1E 00 00 00	.....C....
307E:0210	04 01 00 00 00 00 00 00	00-00 00 00 00 00 00 00 00	.....C....
307E:0220	0D 20 D5 C5 C8 FD 20 20	20-20 20 B8 DF B6 FE 31 39	.....C....
307E:0230	30 2E 35 38 30 2E 30 37	37-35 2E 35 33 30 2E 30 33	0.530.075.530.00
307E:0240	39 2E 30 20 C0 EE CB C4	20 20 20 B8 DF B6 FE 31 39	9.0
307E:0250	31 38 39 2E 30 39 30 2E	30 37 30 2E 30 39 30 2E	139.000.070.000
307E:0260	30 36 35 2E 30 1A		005.0.

其中 D 命令的显示结果为数据库的物理存储格式。

一个完整的 FOXBASE 数据库由两个部分组成: 一个部分为数据库结构描述, 另一个部分为数据库数据内容, 而且结构描述在前, 数据内容在后。数据库结构描述又可分成两个部分: 数据库的整体结构描述与字段结构描述。

数据库的整体结构描述是从数据库的第一个字节开始, 共 32 个字节。这 32 个字节的含义分别如下:

字节	含 义
0	该字节为03H时表示无Memory字段，为80H时表示含.DBT文件(有Memory时产生的文件)。上例中第一个字节为03，表示无Memory字段。
1~3	数据库最近修改的日期，这三个字节分别表示年、月、日。日期是以二进制来表示的，并不是ASCII码。例如本例中5C表示92年、02表示2月、11表示17日。
4~7	文件记录数，低位字节在前，高位字节在后，数据用二进制表示。
8~9	结构描述部分长度，长度=(第9字节)×256+(第8字节)。结构描述结束符为0DH。本例结构描述部分长度为1×256+33=289。
10~11	记录长度。记录长度=(第11字节)×256+(第10字节)。本例的记录长度为34个字节。
12~31	保留，一般为0。

从第32字节开始，每32个字节为一个字段描述：

字节	含 义
0~9	字段名，以ASCII码存放。
10	保留，一般为0。
11	字段类型。根据不同的数据类型，分别是字母C、D、L、N等的ASCII码值。
12~15	首记录中该字段对应的内存地址，12~13为偏移量，14~15为段地址。
16	字段长度。即字段的字节数，最多不超过256个字节。
17	数字型(N型)字段的小数位位数。
18	保留，一般为0。

字段描述部分结束符(0DH)以后，存储数据文件各个记录的内容。记录以定长格式顺序存储，每个记录的第一个字节是删除标记位，被删除的记录第一个字节为2AH，即‘\*’，否则为空格(20H)。每个记录的字段之间没有分隔符，记录尾也无终止符。各种类型的数据均以ASCII码存放，文件结束符为1AH。

### 10.1.2 Turbo C 读取数据库的实现

明确了数据库的结构后，就可以用Turbo C的文件操作函数读取数据库中的数据。因为数据库中有些内容是按二进制方式存储的，有些内容是按ASCII码方式存储的，为了兼顾两者，Turbo C应以二进制只读方式打开数据库文件。对用户来说只需要读取数据库的记录内容，不需要读数据库的结构描述部分，但结构描述部分的长度不是固定不变的，而是随数据库结构的不同而不同。要定位文件位置指针就必须知道结构描述部分的长度，而结构描述部分的长度就存放在这个部分的第8~9两个字节中，故只要读取这两个字节的内容就可知道数据库结构描述部分的长度，再将Turbo C的文件位置指示器从头

开始跳过此长度的字节数，就能读取数据库记录的内容。在读取之前，应先在 Turbo C 中定义一个结构，因为数据库中记录的内容不论其类型均以 ASCII 码存储，所以结构的所有成员应定义为字符型数组，其中每个数组的大小与数据库中相应字段的长度相同。下面的例程读取 SCORE.DBF 中的记录内容，然后求出每个学生的平均分数，并将其各科成绩及平均分数显示在屏幕上。

#### 例 10-1:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
void input(void);
typedef struct {          /* 说明一个与数据库相对应的结构类型 */
    char begin;
    char name[8];
    char class[4];
    char subclas[1];
    char math[4];
    char phys[4];
    char chem[4];
    char engl[4];
    char biol[4];
} orig;
main( )
{
    FILE * fp;
    orig * spo;          /* 定义结构变量指针 */
    float f, aver;
    int i=0, j;
    char c;
    char name__a[9], class__a[5], subclas__a[2];
    char math__a[5], phys__a[5], chem__a[5], engl__a[5], biol__a[5];
    clrscr( );
    printf("姓名      年级      班级      数学  物理  "
           "化学  英语  生物      总平\n");
    fp = fopen("a:score.dbf", "rb"); /* 打开数据库文件只读 */
    fseek(fp, 8, SEEK_SET); /* 文件位置指针定位于存储描述部分长度的字节 */
    j = fgetc(fp) + 256 * fgetc(fp); /* 得到描述部分长度 */
    while(ftell(fp) < j) /* 文件指针定位于记录内容处 */
        fgetc(fp);
```

```

while(fread(spo,sizeof(orig),1,fp)!=NULL) /* 读一条记录 */
{
    strncpy(name__a,spo->name,8);
    name__a[8]=0;
    strncpy(class__a,spo->class,4);
    class__a[4]=0;
    strncpy(subclas__a,spo->subclas,1);
    subclas__a[1]=0;
    strncpy(math__a,spo->math,4);
    math__a[4]=0;
    strncpy(phys__a,spo->phys,4);
    phys__a[4]=0;
    strncpy(chem__a,spo->chem,4);
    chem__a[4]=0;
    strncpy(engl__a,spo->engl,4);
    engl__a[4]=0;
    strncpy(biol__a,spo->biol,4);
    biol__a[4]=0;
    printf("%s  %s  %s  %s  %s  %s  %s  %s",
        name__a,class__a,
        subclas__a,math__a,phys__a,chem__a,engl__a,biol__a);
    f=atof(math__a)+atof(phys__a)+atof(chem__a)+
        atof(engl__a)+atof(biol__a);
    printf("      %4.1f\n",f/5.0);
}
fclose(fp);
}

```

本程序运行后屏幕上显示:

姓名	年级	班级	数学	物理	化学	英语	生物
张三	高二	1	90.5	80.0	75.5	80.0	83.0
李四	高二	1	89.0	90.0	70.0	90.0	65.0

在本例中,对每个结构成员都定义了一个比其长度多1的数组,如 name\_\_a[9]比 name[8]多一个元素,将每个结构成员中的字符串拷贝到对应的数组中之后,给数组的最后一个元素赋空。这是因为当数据库中的数据为定义的字段长度时,由于各字段间无空格,同时 Turbo C 对数组变量也不作边界检查,这就可能造成读取的一个字段数据中包含了下一字段。采取上述方法后,就可以避免这种错误。

另外,由于数据库记录之间有一个空格,因此,在结构变量中也同样定义了一个没有用的结构成员 begin(这里未考虑这个位可存放数据库记录的删除标记" \* ")。



根据以上所述,在编写读取 FOXBASE 数据库的 Turbo C 程序时,必须先进入 FOXBASE 环境查看数据库的结构,才可定义一个与之对应的结构变量。但是如果直接读取数据库结构描述部分的相应字节,得到字段个数和每个字段长度,再用定义指针和动态分配内存的办法,就可以编写一个通用的按记录方式读取.DBF 的 Turbo C 程序。然而进入 FOXBASE 环境查看数据库结构并不费力,所以上面方法完全通用而且简单。

以上介绍的只是 Turbo C 读取 FOXBASE 数据库中的记录内容,同样也可以用 Turbo C 对数据库进行写入、删除或修改记录的操作,但进行这些操作时要改写数据库描述部分中的有关内容及数据库结束标志 1AH 的位置,使之与实际记录相匹配。更新数据库文件时,最好也修改库头的日期。有兴趣的读者可以根据本节讲述的数据库结构,用 Turbo C 文件的随机写函数自行编写程序。但这样做无疑相当麻烦,而且也不能发挥 FOXBASE 管理数据库的优点,如能在 Turbo C 中需要时调用 FOXBASE 对数据库进行操作,操作完毕又退回到 Turbo C 中继续执行就容易多了。在 Turbo C 中提供这样一个函数 `system()`,它能调用其它可执行程序,执行完毕又退回到 Turbo C 中继续运行。因此只要在需调用 FOXBASE 的地方加一条语句:

```
system(foxbase filename);
```

就能完成所需功能。其中: `filename` 为 FOXBASE 的命令文件。

上述方法在实际使用时存在一个严重问题,就是 Turbo C 程序不能太大,否则将不调用 FOXBASE。原因很简单,因为 Turbo C 程序中用 `system()` 函数调用其它执行文件时,本身并不被覆盖,调用完毕还要返回继续运行,而 FOXBASE 命令却又必须在 FOXBASE 环境中运行,环境本身就占据很大内存,造成用 `system()` 函数调用时内存不够用。可以将 FOXBASE 改为占据内存较少的 dBASE,即写成:

```
system(dbase filename);
```

但仍不能从根本上解决问题。本书介绍一种巧妙使用 DOS 批处理文件的方法就可克服这一难题,由于其中涉及了菜单的技术,将放在第十二章中叙述。

## 10.2 FOXBASE 给 Turbo C 传递参数

用 FOXBASE 的 RUN 命令可以实现在 FOXBASE 环境中运行其它语言编写的可执行程序。同样 RUN 命令也可运行 Turbo C 编写的执行文件。对于 RUN 命令的使用,有关 FOXBASE 参考书已作了详细说明,这里不作介绍。在执行 RUN 命令时,一般并不给调用程序传递任何参数。可以使用上节讲述的方法,直接用 Turbo C 读取数据库中的数据来实现参数传递。但传递参数较少时,这种方法无疑显得太麻烦。在第六章 6.2.2 节中,曾讲述了 Turbo C 提供这样一种功能,可以将经过编译、链接后的执行文件看成一个 DOS 命令,而该程序运行时所需的参数可从命令行直接获得,这一功能是通过在 Turbo C 程序中给 `main()` 函数设立两个命令行变元 `argc` 和 `argv[]` 来实现的,即 `main(argc, *argv[])`。利用 Turbo C 的这一特点,可在 FOXBASE 环境的 RUN 命令行中写入要传递的参数,就能直接给 Turbo C 传递参数。

注:

● Turbo C 中命令行参数是按字符串来接收的,对命令行参数的解释由具体程序完成,故在 FOXBASE 中所有要传递的参数类型都应转换成字符串才能代入 RUN 命令中

传递。

- 用这种方法传递参数时，参数数目受到命令行长度的自然限制，传递的参数不可能很多。

- 由于 FOXBASE 本身占据较大内存，而且一般又多在汉字操作系统下运行，这就限制了用 Turbo C 编写的程序不能太大，如将 UCDOCS 显示字库放在扩展内存或改用代码较少的 dBASE，可对此有一些改观。

下面的例程将 FOXBASE 中四个数据传递给 Turbo C 程序，并根据所得到的数据用 Turbo C 绘制一个立体直方图。按任意键又可退到 FOXBASE 环境下。

FOXBASE 命令文件 EXAM10\_\_2.PRG 如下：

```
set talk off
set menu off
n1 = 250
y1 = str(n1)
n2 = 200
y2 = str(n2)
n3 = 150
y3 = str(n3)
n4 = 70
y4 = str(n4)
run exam10-2 &y1 &y2 &y3 &y4
run exam10-2 &y4 &y3 &y2 &y1
```

Turbo C 的程序为 EXAM10\_\_2.C 如下：

```
#include <stdio.h>
#include <graphics.h>
main(int argc, char *argv[]) /* 定义含命令行参数主函数 */
{
    int gdriver = DETECT, gmode;
    int y1, y2, y3, y4, y5;
    if(argc != 5) /* 判断是否为 5 个命令行参数(包括文件名) */
    {
        clrscr(); /* 不是显示提示信息并退出 */
        puts("Number of paramemters is wrong in FOXBASE calling");
        getch();
        exit(1);
    }
    registerbgidriver(EGAVGA__driver);
    initgraph(&gdriver, &gmode, "c:\\tc"); /* 图形屏幕初始化 */
    setbkcolor(BLUE);
    setcolor(LIGHTRED);
    setfillstyle(8, 10);
```

```

y1 = 300-atoi(argv[1]);          /* 得到第一个参数并将其转换 */
bar3d(160,y1,205,300,20,1);        /* 画立体直方图 */
setcolor(YELLOW);
setfillstyle(10,12);
y2 = 300-atoi(argv[2]);          /* 得到第二个参数并将其转换 */
bar3d(205,y2,245,300,20,1);        /* 画立体直方图 */
setcolor(GREEN);
setfillstyle(3,13);
y3 = 300-atoi(argv[3]);          /* 得到第三个参数并将其转换 */
bar3d(245,y3,285,300,20,1);        /* 画立体直方图 */
setcolor(WHITE);
setfillstyle(11,10);
y4 = 300-atoi(argv[4]);          /* 得到第四个参数并将其转换 */
bar3d(285,y4,325,300,20,1);        /* 画立体直方图 */
getch( );
closegraph( );
exit(0);
}

```

在 FOXBASE 命令文件中，定义了四个数据。一般数据多为数字型，所以程序中也用数字型变量，再由 str 函数将其转化成字符中，以宏替换形式写入 RUN 命令中。而在 Turbo C 中又将得到的字符型数据转化成数值，再根据数值大小绘制一个立体型的直方图。

运行上面程序时，首先将 EXAM10-2.C 编译成 EXE 文件装在 FOXBASE 所在的目录下，再进入 FOXBASE 环境中运行 EXAM10-2.PRG 的命令文件即可。

## 第十一章 Turbo C 的高级打印技术

一般的中文操作系统对各种打印机都配置了驱动程序,用户只要根据其打印机类型,运行相应的驱动程序就可以打印出汉字,有时也能够改变打印字体和大小等,但这只能用汉字操作系统本身所带的排版打印程序(如UCDOS 2.0中的PRI.EXE)或中文文字处理软件WORDSTAR等进行打印输出。有时用户使用计算机语言编写的应用程序同样有许多输出结果需要打印,并且希望能按一些特殊要求,如按不同字体、字距、行距及大小等进行输出。FOXBASE和BASIC等语言可在程序中使用一些控制打印的语句就能实现。但对于Turbo C这方面介绍的较少。本章针对这个问题,主要讲述在Turbo C中如何放大打印汉字、打印机拷贝高分辨率屏幕图形(EGA、VGA)以及打印机驱动程序的编制等技术,这些技术在一般的书籍中几乎看不到。

需要事先说明的是,本章所述内容以使用最多的M1724或M2024打印机为对象。然而,掌握了这种原理和技术,编写适用于其它打印机的程序也就很容易了。

### 11.1 利用打印机驱动程序放大打印汉字

第八章中在讲述biosprint()函数时曾对打印机打印西文、中文的方法作了介绍。但是一些有特殊要求的中文打印,如放大打印汉字还不能实现。那么怎样来放大打印汉字呢?熟悉FOXBASE(或dBASE)的读者知道,在FOXBASE中,用命令?CHR(27)+'T'加上一个从A到D(或更多)的不同字母就可以控制打印机输出汉字的大小。由此得到启发,如果将上述控制命令改成Turbo C的语句,是否能够实现改变汉字大小的打印呢?下面请看一个例子。

例 11-1:

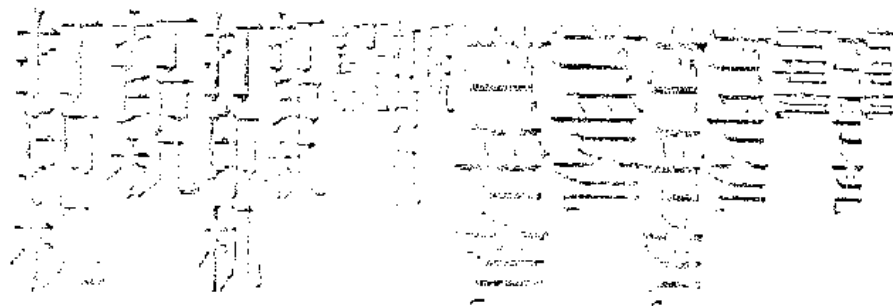
```
#include <stdio.h>
#define ESC 27      /* 定义一个符号常数 ESC */
main( )
{
    fprintf(stdprn,"%c%c%c", ESC, 'T', 'A'); /* 控制打印机字型 */
    fprintf(stdprn,"%s\n", "打印机");      /* 输出汉字串 */
    fprintf(stdprn,"%c%c%c", ESC, 'T', 'B');
    fprintf(stdprn,"%s\n", "打印机");
    fprintf(stdprn,"%c%c%c", ESC, 'T', 'C');
    fprintf(stdprn,"%s\n", "打印机");
    fprintf(stdprn,"%c%c%c", ESC, 'T', 'D');
    fprintf(stdprn,"%s\n", "打印机");
    fprintf(stdprn,"%c%c%c", ESC, 'T', 'E');
    fprintf(stdprn,"%s\n", "打印机");
    fprintf(stdprn,"%c%c%c", ESC, 'T', 'F');
```

```

fprintf(stdprn,"%s\n","打印机");
fprintf(stdprn,"%c%c%c", ESC,T,'G');
fprintf(stdprn,"%s\n","打印机");
fprintf(stdprn,"%c%c%c", ESC,T,'H');
fprintf(stdprn,"%s\n","打印机");
fprintf(stdprn,"%c%c%c", ESC,T,'I');
fprintf(stdprn,"%s\n","打印机");
fprintf(stdprn,"%c%c%c", ESC,T,'J');
fprintf(stdprn,"%s\n","打印机");
fprintf(stdprn,"%c%c%c", ESC,T,'K');
fprintf(stdprn,"%s\n","打印机");
fprintf(stdprn,"%c%c%c", ESC,T,'L');
fprintf(stdprn,"%s\n","打印机");
fprintf(stdprn,"%c%c%c", ESC,T,'M');
fprintf(stdprn,"%s\n","打印机");
fprintf(stdprn,"%c%c%c", ESC,T,'N');
fprintf(stdprn,"%s\n","打印机");
}

```

将这个程序在 UC DOS 2.0 汉字操作系统下运行, 并对 UC DOS 关于打印配置中的各项内容选择 M1724 或 M2024、中文模式、单向打印、水平和垂直放大 1 倍、宋体字。此时输出结果如下:



由该例结果可以看出, Turbo C 同样支持这种打印功能。例 11-1 中最关键的语句就是 fprintf() 函数。如对于 A 型字用下面语句:

```
fprintf(stdprn, "%c%c%c", ESC,T,'A');
```

函数中 stdprn 代表打印机。ESC 是一个宏名, 在程序头已被定义为 27 (Esc 键的 ASCII 码)。M1724 打印机使用的控制命令是 ESC 系列 (许多打印机都使用 ESC 系列的控制命令)。上述语句也可不定义宏名直接用 Esc 的 ASCII 码值。此时变成:

```
fprintf(stdprn, "%c%c%c", 27,T,'A');
```

使用这种打印方法时, 只要在程序中按要求的字型加入一条 fprintf() 函数, 其后的打印输出字型就能满足要求了。

注:

- 也可以不在 UC DOS 汉字操作系统中打印, 只要运行一次 M1724 或 M2024 打印

机驱动程序,也可以在西文操作系统下打印出同样效果的汉字。

● 对于自带汉字库的打印机(如 LQ1600K、NEC6300/6200 等),不必运行打印机驱动程序也可以按上述方法输出相同的结果,只是字型少了几种。

## 11.2 VGA 高分辨率(640×480)屏幕图形的打印机输出

使用 DOS 的 GRAPHICS 命令后,可以用打印机硬拷贝低分辨率(如 640×200,320×200)的屏幕图形,但对于 EGA、VGA 高分辨率屏幕图形就无能为力了。本节讲述用 Turbo C 编写一个子函数,使用时只要在应用程序中调用该函数就可以将屏幕上 640×480 高分辨率的图形从打印机输出,掌握了这种方法,用户可以根据需要编写驱动程序,如对屏幕图形旋转 90 度硬拷贝、对 640×350 的 EGA 分辨率以及其它任何分辨率的屏幕图形进行硬拷贝,因为在原理上完全相同。

### 11.2.1 M1724 打印机的控制命令

每种打印机都有一些控制命令,打印机厂商已将这些命令以手册形式随打印机一起出售给用户。M1724 打印机使用的是 ESC 系列控制命令。下面讲述几个程序中用到的控制命令。

#### 1. 点图像打印输出的控制命令 ESC 4 和 ESC G

这两条命令的格式为:

ESC 4 n1 n2

ESC G n1 n2

转换成十进制 ASCII 的形式为: 27 52 n1 n2

27 71 n1 n2

为了打印屏幕图形(或汉字)必须将打印机设置成点图像打印模式,此时打印机的 24 个针分别对应三个字节共 24 位,沿垂直方向排列,如图 11-1 所示(打印机的设置为出厂设置)。

由图 11-1 可以看出为什么 24×24 点阵打印字模要按列的顺序来存储。其实原因很简单,按列存储汉字时,每三个连续的字节正好打印出一列,打印 24 列也就打印出一个汉字。

以上两条命令均为打印机点图像打印设置命令,其中 ESC 4 n1 n2 是将打印机设置为双向打印,而 ESC G n1 n2 则将打印机设置为单向打印。为了打印出高质量的图像,最好设置为单向打印,但此时速度比双向打印慢了近一倍。

指令中的 n1 和 n2 给出打印图像的幅面宽度,水平方向的打印点数按  $n1 \times 256 + n2$  计算,在打印机左、右边界缺省时,  $n1 \times 256 + n2 < 2176$ 。

#### 2. 行间距设置命令 ESC J

命令的格式为: ESC J n

对应的十进制 ASCII 码形式: 27 74 n

该命令设置行与行之间的间距为  $(n/120)''$ , n 的范围为  $1 < n < 255$ 。当  $n=0$  时,打印机取缺省行间距  $(1/6)''$ 。

#### 3. 行进给命令 ESC VT

命令的格式为: ESC VT n

对应的十进制ASCII码形式: 27 11 n

该命令按预先设置的行间距进给 n 行, n 的范围为  $1 \leq n \leq 255$ 。

由于在本章的程序中仅用到上述 3 条命令, 所以只介绍这三条命令。当然还有许多打印机控制命令, 需要时可查阅相应的打印机说明书。

ESC	4	n1	n2	D1	D2	D3	D4	• • • • •
27	52							

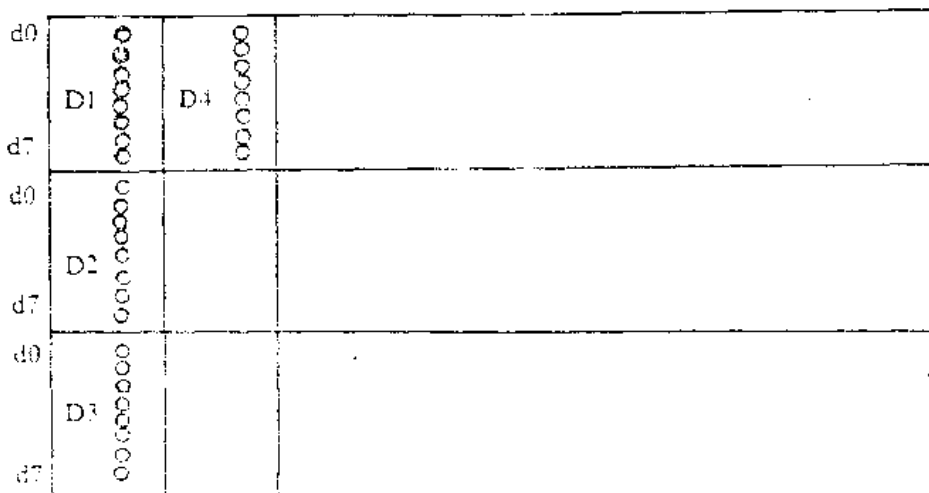


图 11-1 打印针与字节的对应关系

### 11.2.2 VGA 高分辨率屏幕图形的打印机输出程序

由于打印机的打印模式可以设置为点图像打印方式, 此时每向打印机送入 3 个字节, 打印机的 24 个针正好打印一列。Turbo C 在图形方式下, 提供了一个获得象素点颜色的函数 `getpixel()`, 因此只要判断屏幕上每个点的颜色码是否为 0, 就可知道该点的属性。当屏幕上有图像点时, 获得的颜色码不为 0 (作图色), 反之无图像点时获得的颜色码为 0 (背景色)。通过有关位操作函数, 可将屏幕上 Y 方向每 8 个象素点作为一个字节, 每 3 个字节时再判断下一个 X 的象素。X 从 0 到 639 为一行, 打印完一行后将 Y 增加 24, 再打印下一行。设置打印机行间距, 使换一行时正好走过 24 个打印点的距离, 以保证图像连续。由此可完成整个屏幕图像的打印机输出。

在 VGA 高分辨率图形模式时, 其分辨率为  $640 \times 480$ 。如果将垂直方向每 24 个点作为一列, 则正好为  $480 / 24 = 20$  行。但对于 EGA 高分辨率图形模式, 其分辨率为  $640 \times 350$ , 垂直方向只有 14 行并剩余 14 个象素点, 这只要在打印最后一行时作一些特殊处理, 同样可以打印, 当然如果将屏幕图像只画在垂直方向 336 个象元点之内, 则正好为 14 行, 从而可使程序简化。

为了获得高质量图形, 将打印机设置为单向打印, 以降低打印机的机械定位误差而使竖线较直, 但这势必造成打印速度变慢。

下例为 VGA 高分辨率图形的打印机输出程序。程序利用了第九章例 9-5 中的部分屏幕显示结果, 然后调用打印机输出图形子函数将整个屏幕显示结果从打印机输出。

**例 11-2:**

```
#include <stdio.h>
#include <graphics.h>
int handle;
main( )
{
    int gdriver,gmode,i,j,k;
    unsigned char *f="欢迎使用本书";          /* 定义汉字字符串 */
    unsigned char *s[ ]={"愿本书能给您有所帮助",
        "编者于一九九二年五月","实用编程技巧"};
    char c,*str;
    gdriver=DETECT;
    registerbgidriver(EGAVGA__driver);          /* 建立独立图形函数 */
    initgraph(&gdriver,&gmode,"c:\\tc");        /* 图形屏幕初始化 */
    setbkcolor(BLUE);
    cleardevice( );
    setcolor(12);
    rectangle(1,1,637,477);
    setcolor(14);
    setfillstyle(7,10);
    setlinestyle(0,0,3);
    rectangle(8,90,625,195);
    floodfill(200,100,14);
    setlinestyle(0,0,1);
    openhzk(c='k');                             /* 打开楷体字库 */
    puthz24(10,100,8,10,4,4,f);                /* 显示 24×24 点阵汉字 */
    close(handle);                             /* 关闭字库 */
    openhzk(c='s');
    puthz24(60,300,4,9,2,2,s[0]);
    puthz24(180,400,4,2,1,1,s[1]);
    close(handle);
    for(i=3;i<=5;i++)                          /* 将汉字重复显示 3 次 */
    {
        openhzk(c='k');
        puthz24(15-i,95+i,8,12,4,4,f);
        close(handle);
        openhzk(c='s');
        puthz24(60,300,4,10+i,2,2,s[0]);
        puthz24(180,400,4,6+i,1,1,s[1]);
    }
```



```

    close(handle);
    delay(500);                /* 延时 5 秒钟 */
}
prtscr( );                    /* 调用屏幕图形拷贝函数 */
getch( );
closegraph( );
}
int openhzk(char c)           /* 打开不同字库的函数 */
{
    if(c == 's')
        handle = open("c:\\\\clib",O_RDONLY|O_BINARY);
    if(c == 'f')
        handle = open("c:\\\\clibf",O_RDONLY|O_BINARY);
    if(c == 'k')
        handle = open("c:\\\\clibk",O_RDONLY|O_BINARY);
    if(c == 'h')
        handle = open("c:\\\\clibh",O_RDONLY|O_BINARY);
    if(c == 'o')
        handle = open("c:\\\\cclib.dat",O_RDONLY|O_BINARY);
    if(handle == -1)
    {
        cputs("Error on open cclib.dat");
        exit(1);
    }
}
int puthz16(int x,int y,int z,int color,char *p) /* 显示 16×16 点阵汉字 */
{
    unsigned int i, c1, c2, f=0;
    int i1, i2, i3, n;
    long l;
    char by[32];
    while((i = *p++) != 0)
    {
        if(i > 0xa1)                /* 判断是否为汉字 */
            if(f == 0)
            {
                c1 = (i-0xa1)&0x07f;    /* 取汉字区码 */
                f = 1;
            }
            else
            {
                c2 = (i-0xa1)&0x07f;    /* 取汉字位码 */

```

```

f=0;
n=c1*94+c2;
l=n*32L;
lseek(handle,l,SEEK_SET);
    /* 文件位置指针定位于该汉字字模首字节 */
read(handle,by,32);    /* 读汉字字模 */
for(i1=0;i1<16;i1++)
    for(i2=0;i2<2;i2++)
        for(i3=0;i3<8;i3++)
            if(getbit(by[i1*2+i2],7-i3))
                putpixel(x+i2*8+i3, y+i1, color); /* 逐点显示汉字 */
x=x+16+z;
}
}
return(x);
}

int puthz24(int x,int y,int z,int color,int m,int n,char *p)
{
    /* 显示 24×24 点阵汉字函数 */
    unsigned int i, c1, c2, f=0;
    int i1, i2, i3, i4, i5, rec;
    long l;
    char by[72];
    while((i=*p++)!=0)
    {
        if(i>0xa1)
            if(f==0)
            {
                c1=(i-0xa1)&0x07f;
                f=1;
            }
            else
            {
                c2=(i-0xa1)&0x07f;
                f=0;
                rec=c1*94+c2;
                l=rec*72L;
                lseek(handle,l,SEEK_SET);
                read(handle,by,72);
                for(i1=0;i1<24*m;i1=i1+m)
                    for(i4=0;i4<m;i4++)
                        for(i2=0;i2<2;i2++)
                            for(i3=0;i3<8;i3++)

```

```

        if(getbit(by[i1 / m * 3+i2],7-i3))
            for(i5 = 0;i5 < n;i5++)
                putpixel(x+i1+i4,y+i2 * 8 * n+i3 * n+i5,color);
        x = x+24 * m+z;
    }
}
return(x);
}
int getbit(unsigned char c, int n)
{
    return((c >> n)&1);
}
prtscr( )
{
    #include < alloc.h >
    #include < fcntl.h >
    int size,i;
    char ch;
    void * buffer;
    size = imagesize(200,200,370,260);
    buffer = malloc(size);
    getimage(200,200,370,260,buffer);
    setcolor(10);
    rectangle(200,200,370,260);
    rectangle(203,203,367,257);
    setfillstyle(1,9);
    floodfill(220,230,10);
    setcolor(12);
    outtextxy(230,220," < CR > ——Print");    /* 操作提示 */
    outtextxy(230,240," < ESC > ——Quit");
    while(ch != 27 && ch != 13)    /* 等待键盘输入字符直到按回车键或 Esc 键 */
        ch = getch( );
    putimage(200,200,buffer,COPY_PUT);
    free(buffer);
    if(ch == 13)
        readscr( );    /* 按回车键调屏幕图形打印函数 */
    return;
}
readscr( )
{
    #include < stdio.h >
    char c,ch;

```

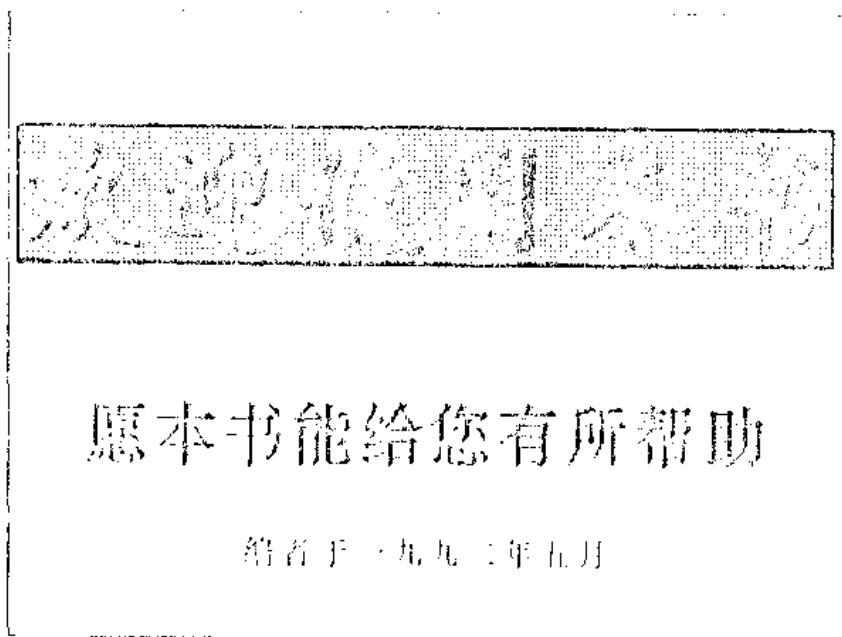
```

int x,y,i,j,k,cor;
unsigned char by[3][640];
for(i=0;i<20;i++)
{
    for(x=0;x<640;x++)
        for(j=0;j<=2;j++)
            by[j][x]=0x00;          /* 数组初始化 */
    for(x=0;x<640;x++)              /* 屏幕水平方向 640 个点 */
        for(j=0;j<=2;j++)           /* 垂直方向 3 个字节 */
            for(y=0;y<=7;y++)        /* 每个字节的 8 位 */
            {
                cor=getpixel(x,i*24+j*8+y); /* 得到屏幕上象素的颜色值 */
                if(cor!=0)                /* 不是背景色该位置！ */
                    k=1;
                else                       /* 是背景色该位置 0 */
                    k=0;
                by[j][x]=putbit(by[j][x],k,y); /* 将该位存入字节的对应位中 */
            }
    fprintf(stderr,"%c%c%c%c%c",27,71,2,128);
        /* 设置打印机为点图像打印 */
    for(x=0;x<640;x++)                /* 打印水平方向一行 */
        for(j=0;j<=2;j++)
            biosprint(0,by[j][x],0);
    fprintf(stderr,"%c%c%c%c",27,74,18); /* 设置打印机行间距 */
    fprintf(stderr,"%c%c%c%c",27,11,1); /* 打印机进给一行 */
}
}

int putbit(unsigned char c, int m, int n)
{
    if(n==7)
    {
        if(m==0)
            return(c&0xfe);
        return(c|0x01);
    }
    else
    {
        if(m==0)
            return((c&0xfe)<<1);
        return((c|0x01)<<1);
    }
}

```

运行结果是:



例 11-2 中的 `prtscr()` 子函数为屏幕图形打印机输出函数, 可以被其它 Turbo C 程序随意调用, 当然这个函数只能在屏幕为图形模式时被调用。在 `prtscr()` 函数中, 又调用了 `readscr()` 函数, 但对用户来说只有 `prtscr()` 函数是透明的。`prtscr()` 函数被调用时在屏幕上弹出一个小窗口, 其中提示要打印屏幕图形时按回车键, 不打印按 Esc 键, 使用户有选择的余地。这个函数完全可以作为独立模块被用户在其程序中的任何地方调用。当然如果作成常驻内存程序, 用屏幕拷贝热键中断调用则更加方便, 但本书不对此作介绍。

### 11.3 自编打印机驱动程序

本章第一节中讲述的汉字打印技术, 可以在运行了打印机驱动程序以后或带硬字库的打印机上输出汉字。即便是这样, 用户想在程序中变化输出汉字的行间距、字间距、大小、字体也比较困难或根本就不可能。本节讲述编写不需要运行打印机驱动程序就可打印出不同字体、字型汉字的 Turbo C 程序, 并且可以随意选择打印汉字的行距、字距, 程序使用灵活, 输出字形美观。

汉字打印的实质与屏幕图形打印机拷贝一样, 须将打印机设置成点图像打印方式, 然后再使用第九章的点阵字库的字模读取技术, 将读取的汉字字模打印出来。当然读取不同字体的字库, 也就改变了打印字体。把这种方法编写的程序直接嵌入到用户程序中, 应用自如, 并且打印字库不占内存。

根据以上介绍的原理, 编写一个打印  $24 \times 24$  点阵可选不同字体、即可定位打印机初始位置、又可随意改变打印机字间距、行间距的程序, 如下例所示。

例 11-3:

```
#include <stdio.h>
#include <graphics.h>
```

```

#include <fcntl.h>
int print(int,int,int,char *,char);
main( )
{
    clrscr( );
    print(0,1,0,"欢迎使用本书",'s');
    print(0,24,0,"欢迎使用本书",'s');
    print(0,30,12,"欢迎使用本书",'k');
    print(50,40,24,"欢迎使用本书",'h');
    print(200,24,48,"欢迎使用本书",'f');
    exit(0);
}

int print(int x,int y,int z,char * p, char c)
{
    unsigned int i,c1,c2,f=0;
    int handle,i1,i2,i3,rcc,test=0;
    long l;
    unsigned char by[72];
    if(c == 's')
        handle = open("c:\\\\clib",O_RDONLY|O_BINARY);
    if(c == 'h')
        handle = open("c:\\\\clibh",O_RDONLY|O_BINARY);
    if(c == 'k')
        handle = open("c:\\\\clibk",O_RDONLY|O_BINARY);
    if(c == 'f')
        handle = open("c:\\\\clibf",O_RDONLY|O_BINARY);
    fprintf(stderr,"%c%c%c%c",27,71,3,192);
        /* 设置打印机为点图像打印模式 */
    for(i1=0;i1<x;i1++)
        for(i2=0;i2<z;i2++)
            biosprint(0,0,0);          /* 定位初始打印位置 */
    while((i = *p++) != 0)
    {
        if(i > 0xa1)                    /* 判断是否为汉字 */
            if(f == 0)
            {
                c1 = (i-0xa1)&0x07f;    /* 取汉字区码 */
                f = 1;
            }
            else
            {
                c2 = (i-0xa1)&0x07f;    /* 取汉字位码 */

```

```

f=0,test++;
rec=c1*94+c2;
l=rec*72L;
lseek(handle,l,SEEK__SET);
read(handle,by,72); /* 读字模 */
for(i1=0;i1<24;i1++) /* 打印字模的24列 */
for(i2=0;i2<3;i2++) /* 打印每列的3个字节 */
biosprint(0,by[3*i1+i2],0); /* 打印机输出 */
for(i1=0;i1<z;i1++)
for(i2=0;i2<3;i2++)
biosprint(0,0,0); /* 字间距 */
if((test+1)*(24+z)+x>=960) /* 判断是否超过一行 */
{
for(i1=0;i1<960-test*(24+z)-x;i1++)
for(i2=0;i2<3;i2++)
biosprint(0,0,0);
fprintf(stdprn,"%c%c%c",27,74,0);
fprintf(stdprn,"%c%c%c",27,11,1);
test=0,x=0;
fprintf(stdprn,"%c%c%c%c",27,71,3,192);
}
}
}
for(i1=0;i1<960-test*(24+z)-x;i1++) /* 不足一行用0填补 */
for(i2=0;i2<3;i2++)
biosprint(0,0,0);
fprintf(stdprn,"%c%c%c",27,74,y); /* 设置行间距 */
fprintf(stdprn,"%c%c%c",27,11,1); /* 打印机进给一行 */
close(handle);
}

```

程序输出为:

欢迎使用本书

欢迎使用本书

欢迎使用本书

欢迎使用本书

程序中最关键的是函数 print( ), 调用格式为:

```
int print(int x, int y, int z, char *p, char c);
```

其中 x 为第一个汉字打印在行中的位置, x 以打印机在图象输出时水平方向相邻两点的距离, 即打印机的最高分辨率为单位。

y 为打印机的行间距。这个行间距只有在打印下面一行时才起作用。它的取值范围为  $0 < y < 255$ ，行间距为  $(y/120)''$ 。当  $y=0$  时，行间距为  $(1/6)''$ ，当  $y=1$  时只有  $(1/120)''$ ，此时肉眼几乎看不出打印机有进给，似乎在同一位置上打印，上例结果中的第一行就是  $y=1$  时的情况，相当于在原来位置重复打印了一次，输出汉字的笔划明显加深。

z 为相邻两个汉字的间距，单位与 x 相同。

p 为字符串指针，指向要打印的汉字字符串。注意这个字符串中只有中文信息才能输出到打印机。

c 用来选择打印汉字的字体。c='s' 打印宋体字，c='k' 打印楷体字，c='h' 打印黑体字，c='f' 打印仿宋体字。

程序中规定在  $x=0, z=0$  时，一行只能打印 40 个汉字，超过 40 个汉字将转到下一行开始并继续打印。但当 x 和 z 不为 0 时，也具有此功能，只不过一行不一定是 40 个汉字罢了。

打印机汉字库字模按列进行存储，每一列 3 个字节正好对应打印机的 24 根针头。如果打印一列后在水平方向下一个点再打印同样的一列，就相当于将打印的汉字水平放大了一倍，打印 m 次，相当于水平方向放大了 m 倍。对于垂直方向却有所不同。放大一倍时，相当于一个汉字打印了上半部后，进给一行(设置行间距使两行间不留空间)再打印下半部，即把  $24 \times 24$  点阵字库中每个字节的每一位变成相同的两位，字模垂直方向每 4 位生成一个新字节。字模第 1、4、7、10、... 字节中的低 4 位变成新字模第 1、4、7、10、... 字节。高 4 位变成新字模第 2、5、8、11、... 字节，字模第 2、5、8、10、... 字节中的低 4 位变成新字模的 3、6、9、12、... 字节。这时可用新字模的 72 个字节输出汉字的上半部分，然后读取  $24 \times 24$  点阵字模中的下半部汉字，按同样方法产生一个新字模，再输出汉字的下半部分。这样就实现了汉字垂直方向放大一倍的功能。请看下面的例程：

#### 例 11-4:

```
#include <stdio.h>
#include <graphics.h>
#include <fcntl.h>
int print(int,int,int,int,int,char *,char);
int getbit(unsigned char,int);
main( )
{
    clrscr( );
    print(0,1,0,1,1,"欢迎使用本书",'s');
    print(0,20,0,1,1,"欢迎使用本书",'s');
    print(0,30,12,1,2,"欢迎使用本书",'k');
    print(50,40,24,2,1,"欢迎使用本书",'h');
    print(200,24,48,2,2,"欢迎使用本书",'f');
    exit(0);
}
int print(int x,int y,int z,int m,int n,char *p, char c)
```



```

unsigned int i,c1,c2,f=0;
int handle,i1,i2,i3,i4,rec,test;
long l;
unsigned char by[72],by1[72],*p1;
if(c == 's')
    handle=open("c:\\\\clib",O_RDONLY|O_BINARY);
if(c == 'h')
    handle=open("c:\\\\clibh",O_RDONLY|O_BINARY);
if(c == 'k')
    handle=open("c:\\\\clibk",O_RDONLY|O_BINARY);
if(c == 'f')
    handle=open("c:\\\\clibf",O_RDONLY|O_BINARY);
fprintf(stderr,"%c%c%c%c%c",27,71,3,192);
for(i1=0;i1<x;i1++)          /* 定位初始打印位置 */
    for(i2=0;i2<3;i2++)
        biosprint(0,0,0);
switch(n)
{
    case 1:                  /* 本选择为垂直方向不放大的打印 */
        test=0;
        while((i = *p++)!=0)
        {
            if(i>0xa1)      /* 判断是否为汉字 */
            if(f==0)
            {
                c1=(i-0xa1)&0x07f; /* 取汉字区码 */
                f=1;
            }
            else
            {
                c2=(i-0xa1)&0x07f; /* 取汉字位码 */
                f=0,test++;
                rec=c1*94+c2;
                l=rec*72L;
                lseek(handle,l,SEEK_SET);
                read(handle,by,72);          /* 读汉字字模 */
                for(i1=0;i1<24;i1++)        /* 打印汉字 */
                    for(i4=0;i4<m;i4++)
                        for(i2=0;i2<3;i2++)
                            biosprint(0,by[3*i1+i2],0);
                for(i1=0;i1<z;i1++)          /* 空字间距 */

```

```

        for(i2 = 0; i2 < 3; i2++)
            biosprint(0,0,0);
    }
}
for(i1 = 0; i1 < 960 - test * (24 + z) - x; i1++)
    for(i2 = 0; i2 < 3; i2++)
        biosprint(0,0,0);
fprintf(stdprn, "%c%c%c%c", 27, 74, y);    /* 设置行距 */
fprintf(stdprn, "%c%c%c%c", 27, 11, 1);    /* 进给一行 */
break;
case 2:                                /* 本选择为垂直方向放大一倍的打印 */
    p1 = p, test = 0;
    while((i = *p++) != 0)    /* 打印一行汉字的上半部分 */
    {
        if(i > 0xa1)
            if(f == 0)
            {
                c1 = (i - 0xa1) & 0x07f;
                f = 1;
            }
        else
        {
            c2 = (i - 0xa1) & 0x07f;
            f = 0, test++;
            rec = c1 * 94 + c2;
            l = rec * 72L;
            lseek(handle, l, SEEK_SET);
            read(handle, by, 72);
            for(i1 = 0; i1 < 24; i1++)
            {
                for(i2 = 0; i2 < 8; i2++)
                    if(getbit(by[3 * i1], 7 - i2))
                        by1[i1 * 3 + i2 / 4] = (by1[i1 * 3 + i2 / 4] < < 2) | 3;
                else
                    by1[i1 * 3 + i2 / 4] = (by1[i1 * 3 + i2 / 4] < < 2) & 0xfc;
                for(i2 = 0; i2 < 4; i2++)
                    if(getbit(by[3 * i1 + 1], 7 - i2))
                        by1[i1 * 3 + 2] = (by1[i1 * 3 + 2] < < 2) | 3;
                    else
                        by1[i1 * 3 + 2] = (by1[i1 * 3 + 2] < < 2) & 0xfc;
            }
            for(i1 = 0; i1 < 24; i1++)

```

```

        for(i4 = 0; i4 < m; i4++)
            for(i2 = 0; i2 < 3; i2++)
                biosprint(0, buf[3 * i1 + i2], 0);
    for(i1 = 0; i1 < z; i1++)
        for(i2 = 0; i2 < 3; i2++)
            biosprint(0, 0, 0);
    if((test + 1) * (24 * m + z) + x >= 960)
    {
        for(i1 = 0; i1 < 960 - test * (24 * m + z) - x; i1++)
            for(i2 = 0; i2 < 3; i2++)
                biosprint(0, 0, 0);
        fprintf(stderr, "%c%c%c%c", 27, 74, 0);
        fprintf(stderr, "%c%c%c%c", 27, 11, 1);
        test = 0, x = 0;
        fprintf(stderr, "%c%c%c%c", 27, 71, 3, 192);
    }
}

for(i1 = 0; i1 < 960 - test * (24 * m + z) - x; i1++)
    for(i2 = 0; i2 < 3; i2++)
        biosprint(0, 0, 0);
fprintf(stderr, "%c%c%c%c", 27, 74, 18);
fprintf(stderr, "%c%c%c%c", 27, 11, 1);
fprintf(stderr, "%c%c%c%c", 27, 71, 3, 192);
for(i1 = 0; i1 < x; i1++)
    for(i2 = 0; i2 < 3; i2++)
        biosprint(0, 0, 0);
test = 0;
while((i = * p1++) != 0)    /* 打印一行汉字的下半部分 */
{
    if(i > 0xa1)
        if(f == 0)
        {
            c1 = (i - 0xa1) & 0x07f;
            f = 1;
        }
    else
    {
        c2 = (i - 0xa1) & 0x07f;
        f = 0, test++;
        rec = c1 * 94 + c2;
        l = rec * 72L;
    }
}

```

```

lseck(handle,1,SEEK_SET);
read(handle,by,72);
for(i1=0;i1<24;i1++)
{
    for(i2=4;i2<8;i2++)
        if(getbit(by[3*i1+1],7-i2))
            by1[i1*3]=(by1[i1*3]<<2)|3;
        else
            by1[i1*3]=(by1[i1*3]<<2)&0xfc;
    for(i2=0;i2<8;i2++)
        if(getbit(by[3*i1+2],7-i2))
            by1[i1*3+i2/4+1]=(by1[i1*3+i2/4+1]<<2)|3;
        else
            by1[i1*3+i2/4+1]=(by1[i1*3+i2/4+1]<<2)&0xfc;
    }
    for(i1=0;i1<24;i1++)
        for(i4=0;i4<m;i4++)
            for(i2=0;i2<3;i2++)
                biosprint(0,by1[3*i1+i2],0);
    for(i1=0;i1<z;i1++)
        for(i2=0;i2<3;i2++)
            biosprint(0,0,0);
}
}
for(i1=0;i1<960-test*(24*m+z)-x;i1++)
    for(i2=0;i2<3;i2++)
        biosprint(0,0,0);
fprintf(stdprn,"%c%c%c%c",27,74,y);
fprintf(stdprn,"%c%c%c%c",27,11,1);
break;
default:
    break;
}
close(handle);
}
int getbit(unsigned char c,int n)
{
    return((c>>n)&1);
}

```

程序输出:

欢迎使用本书

欢迎使用本书

欢迎使用本书

欢迎使用本书

该程序中最关键的同样是函数 `print()`，调用格式为：

```
int print(int x, int y, int z, int m, int n, char *p, char c);
```

其中：`m` 为水平放大倍数，最小为 1，最大几乎可以不受限制。

`n` 为垂直放大倍数。本例只能取 1 或 2，即最大只能放大 2 倍，但程序中使用了一个开关语句。读者可自己设计放大 3 倍、4 倍、... 的程序，加到本例中开关语句的不同选择部分。其原理与放大 2 倍完全相同，只是比较麻烦而已。

其它参数的含义均与例 11-3 相同。

学习了汉字的这种打印原理，相当于从根本上了解了打印机的汉字输出技术，读者可以自行设计手写体汉字库，再从打印机输出，也可以用类似的方法调用高点阵字库，如  $40 \times 40$ 、 $48 \times 48$  点阵字库等进行打印输出，也可以旋转 90 度打印汉字。因此完全能够自行设计一个功能齐全的打印机驱动程序了。

## 第十二章 菜单设计技术

菜单设计在用户编写的程序中占有相当一部分比重。设计一个高质量的菜单,不仅能使用户美观悦目,更重要的是能够使操作者使用方便,并避免因误操作带来的严重后果。本章主要介绍用 Turbo C 编写各种菜单的技术,这些菜单包括西文操作系统下文本方式时下拉式菜单的设计和西文操作系统下用读取汉字库字模的方法进行中文窗口式、下拉式菜单的设计。最后再介绍用窗口式菜单生成批处理文件的办法将 FOXBASE 和 Turbo C 执行文件联合使用的技巧。本节所述程序,只须根据具体情况改变菜单的内容,就可以直接供读者使用。

### 12.1 西文下拉式菜单的设计

下拉式菜单是一种典型的窗口菜单。所谓窗口是指屏幕上划出的一个矩形区域,它可以从屏幕上消失,也可以重新显示在屏幕上,窗口之间也允许覆盖。下拉式菜单则是自上而下向屏幕弹出一个个窗口菜单供操作者选择条目或输入内容。Turbo C 集成开发环境就是一个下拉式菜单的例子。下拉式菜单中一般有一个主菜单,其中包括几个选择项,主菜单的每一项又各衍生出下一级菜单,这样逐级地以一个个窗口的形式弹出在屏幕上,一旦操作完毕又可从屏幕上消失,恢复原来的屏幕状态。下拉式菜单整体感强,使操作者一目了然、使用方便。

设计下拉式菜单的关键就是在下一级的菜单窗口弹出之前,把要被该窗口掩盖的屏幕区域保存起来,然后产生这一级菜单窗口,并可使用光标键选择菜单中各项,回车键来确认。如果某选择项还有下一级菜单,则按同样的方法去产生再下一级菜单窗口。另外,在选择或输入结束之后,还要能用 Esc 键向上逐级退回到主菜单,也就是将先前保存的各帧屏幕又向上逐级释放出来。而 Turbo C 在文本方式时提供了函数 `gettext()` 用来存放屏幕规定区域的内容,当需要时又可用 `puttext()` 函数释放出来,再加上关于键盘操作管理的函数 `bioskey()`,就完成了下拉式菜单的设计要求。

下面通过一个例子说明西文文本方式时下拉式菜单的设计方法。

#### 例 12-1:

```
#include <conio.h>
main( )
{
    int i,key0,key,key1,y,test;
    char * m[] = {"File","Edit","Run","Compile","Project",
                  "Options","Debug","Break / watch"}; /* 定义主菜单内容 */
    char * f[] = {" Load      F3 ",
                  " Pick   Alt-F3 ",
                  " New          " }; /* 定义 File 子菜单内容 */
```

```

        " Save      F2 " ,
        " Write to  " ,
        " Directory " ,
        " Change dir " ,
        " Os shell  " ,
        " Quit Alt-X "};

char buf[16 * 10 * 2],buf1[16 * 2];          /* 定义保存屏幕区域的数组变量 */
textbackground(BLUE);                        /* 设置文本屏幕背景色 */
clrscr( );                                  /* 屏幕着色 */
window(1,1,80,1);                           /* 定义一个文本窗口 */
textbackground(WHITE);                       /* 设置窗口背景色 */
textcolor(BLACK);

clrscr( );
window(1,1,80,2);
for(i=0;i<8;i++)
    cprintf(" %s",m[i]);                    /* 显示主菜单内容 */
while(1)
{
    key=0;
    while(bioskey(1) == 0);                  /* 等待键盘输入 */
    key = bioskey(0);                         /* 取键盘输入码 */
    key = key & 0xff70; key >> 8;              /* 只取扩充键码 */
    if(key == 45) exit(0);                    /* 如果是按 Alt+X 则退出 */
    if(key == 33)                             /* 如果是按 Alt+F 则显示子菜单 */
    {
        textbackground(BLACK);
        textcolor(WHITE);
        gotoxy(4,1);
        cprintf("%s",m[0]);
        gettext(4,2,19,11,buf);             /* 保存窗口区域的原有内容 */
        window(4,2,19,11);
        textbackground(WHITE);
        textcolor(BLACK);
        clrscr( );
        window(4,2,19,12);
        gotoxy(1,1);                          /* 作一个单线矩形边框 */
        putch(0xda);
        for(i=2;i<16;i++) putch(0xc4);
        putch(0xbf);
        for(i=2;i<10;i++)
        {

```

```

        gotoxy(1,i);putch(0xb3);
        gotoxy(16,i);putch(0xb3);
    }
    gotoxy(1,10);
    putch(0xc0);
    for(i=2;i<16;i++)putch(0xc4);
        putch(0xd9);
    for(i=2;i<10;i++)                /* 显示 File 子菜单内容 */
    {
        gotoxy(2,i);
        cprintf("%s",f[i-1]);
    }
    gettext(2,2,18,3,buf1);          /* 保存产生色条前的原来信息 */
    textbackground(BLACK);             /* 产生一个色条 */
    textcolor(WHITE);
    gotoxy(2,2);
    cprintf("%s",f[0]);
    y=2;
    key0=0;
    while(key0!=27&&key1!=45&&key0!=13) /* 输入为Alt+X、回车或Esc键时退出循
                                        环 */
    {
        while(bioskey()==0);          /* 等待键盘输入 */
        key0=key1=bioskey(0);          /* 取键盘输入码 */
        key0=key0&0xff;                /* 只取 ASCII 码 */
        key1=key1&0xff?0:key1>>8;      /* 只取扩充码 */
        if(key1==72||key1==80)          /* 如果为上、下箭头键 */
        {
            puttext(2,y,18,y+1,buf1);  /* 恢复原来信息 */
            if(key1==72)y=y==2?9:y-1;   /* 上箭头处理 */
            if(key1==80)y=y==9?2:y+1;   /* 下箭头处理 */
            gettext(2,y,18,y+1,buf1); /* 保存新色条产生前这一位置屏幕内容 */
            textbackground(BLACK);       /* 产生新色条 */
            textcolor(WHITE);
            gotoxy(2,y);
            cprintf("%s",f[y-1]);
        }
        test=y-1;
    }
    if(key1==45)exit(0);                /* Alt+X 则退出 */
    if(key0==13)                        /* 回车按所选子菜单项进行处理 */

```



```

    {
        switch(y)
        {
            case 1:
                break;
            case 2:
                break;
            case 9:
                exit(0);
            default:
                break;
        }
    }
else
    /* Esc 键则返回主菜单 */
    {
        window(1,1,80,2);
        puttext(4,2,19,11,buf); /* 释放子菜单窗口占据的屏幕原来内容 */
        textbackground(WHITE);
        textcolor(BLACK);
        gotoxy(4,1);
        cprintf("%s",m[0]);
    }
}
}
}

```

程序模仿 Turbo C 集成开发环境的菜单，运行时在屏幕顶上一行显示主菜单的内容，若按 Alt+F 则进入 File 子菜单，然后可用光标键移动色条选择操作，回车键确认，也可用 Esc 键退回到主菜单，可用 Alt+X 热键退出系统，本例程序只作了个别几个操作，仅用来说明菜单的设计方法而已。

此例程设计的下拉式菜单主要通过以下几个环节来实现：

1. 采取用不同背景色和字符色重写的办法产生色条。
2. 在产生色条之前，必须用 `gettext()` 函数存取屏幕原来的内容，移动色条之后，又用 `puttext()` 函数将原来屏幕内容释放。
3. 下一级菜单弹出之前也要用 `gettext()` 函数保存屏幕原有的内容，退出下级菜单时，只要将存入内存中的原来屏幕内容用 `puttext()` 释放即可。
4. 使用了关于键盘操作的函数 `bioskey()`，该函数读取按键的 ASCII 码(普通键)或扩充码(特殊键)，根据得到的键码判断是否按了设定的键而执行相关的操作。
5. 使用“?”号表达式完成循环移动色条，当色条处于菜单最顶上时，若再按向上的箭头键，则色条将移到最底下，同样当色条处于菜单最底下时，若再按向下的箭头键，则色

条将移到最顶上。

## 12.2 中文窗口式菜单的设计

在汉字操作系统下，当屏幕为文本方式时，`gettextl()`、`puttextl()`和关于窗口的输出函数都不能使用，只能使用 `printf()`、`puts()`、`scanf()`、`gets()`等标准的输入输出函数，仅使用这些函数要设计出用光标键移动色条的窗口式菜单相当困难，即使设计出来也是不很美观的。

本章要讲述的是在西文操作系统中，用第九章所讲的读点阵字模在图形方式下显示汉字的方法来设计窗口式菜单的技术。当然程序也可以在中文操作系统下运行，其效果与西文操作系统下运行时完全相同。

### 例 12-2:

```
#include <conio.h>
#include <stdio.h>
#include <stdlib.h>
#include <alloc.h>
#include <graphics.h>
#include <fcntl.h>
int menu(void);
int getbit(unsigned char, int);
int puthz16(int, int, int, int, char *);
int puthz24(int, int, int, int, char *);
int openhz(void);
int handle;
main( )
{
    int gdriver,gmode,chose;
    gdriver = DETECT;
    registerbgidriver(EGAVGA__driver);
    initgraph(&gdriver,&gmode,"c:\\tc");    /* 初始化图形屏幕 */
    while(1)
    {
        chose = menu( );                    /* 进入菜单选择操作 */
        switch(chose)                        /* 根据不同选择分别处理 */
        {
            case 1:
                /*      system("dbase lr");    */
                break;
            case 2:
                /*      system("dbase ix");    */
                break;
```

```

        case 3:
            /*      system("dbase xg");      */
            break;

        case 4:
            /*      system("dbase dy");      */
            break;

        case 5:
            cleardevice( );
            /*      draw( );      */
            break;

        default:
            closegraph( );
            exit(0);
    }
}

int menu( )
{
    int x,y,key1,key2,size,test,i;
    void * buffer;
    char c;
    unsigned char * f[] = {"== <<","系统菜单",">> =="};
    unsigned char * m[] = {"数据录入","数据查询","数据修改",
        "统计计算","报表打印","退出系统"}; /* 定义菜单内容 */
    setbkcolor(BLUE);
    cleardevice( );
    openhzk(c='s'); /* 打开宋体字库 */
    x = puthz24(135,80,0,14,f[0]); /* 显示汉字 */
    x = puthz24(x+8,80,8,12,f[1]);
    puthz24(x,80,0,14,f[2]);
    close(handle);
    setcolor(12);
    setlinestyle(0,0,3);
    rectangle(200,130,440,340); /* 画一矩形框 */
    setfillstyle(1,3);
    floodfill(250,200,12); /* 填充矩形框 */
    openhzk(c='o'); /* 打开 16×16 点阵字库 */
    for(i=0;i<6;i++) /* 显示菜单内容 */
        puthz16(265,i*30+150, 16, 1, m[i]);
    close(handle);
    size = imagesize(230,148,410,168);

```

```

buffer = malloc(size);
getimage(230,148,410,168,buffer);
putimage(230,148,buffer,NOT__PUT);
y = 148;
while(key1 != 13)                                /* 按回车键退出循环 */
{
    while(bioskey(1) == 0);                       /* 等待键盘输入 */
    key1 = key2 = bioskey(0);                     /* 取键盘输入码 */
    key1 = key1 & 0xff;                            /* 只取 ASCII 码 */
    key2 = key2 & 0xff; key2 > 8;                  /* 只取扩充码 */
    if(key2 == 72 || key2 == 80)
    {
        putimage(230,y,buffer,COPY__PUT);        /* 恢复原有信息 */
        if(key2 == 72) y = y == 148 ? 298 : y - 30; /* 上箭头处理 */
        if(key2 == 80) y = y == 298 ? 148 : y + 30; /* 下箭头处理 */
        getimage(230,y,410,y+20,buffer);         /* 保存新位置色条大小的内容 */
        putimage(230,y,buffer,NOT__PUT);         /* 反像释放产生色条 */
    }
}
test = (y - 148) / 30 + 1;
free(buffer);                                    /* 释放内存 */
return(test);
}

int openhzk(char c)                              /* 打开不同字库 */
{
    if(c == 's')
        handle = open("c:\\clib",O_RDONLY|O_BINARY);
    if(c == 'o')
        handle = open("c:\\cclib.dat",O_RDONLY|O_BINARY);
    if(handle == -1)
    {
        cputs("Error on open cclib.dat");
        exit(1);
    }
}

int puthz16(int x, int y, int z, int color, char *p)
{
    /* 显示 16×16 点阵汉字的子函数 */
    unsigned int i, c1, c2, f = 0;
    int i1, i2, i3, rec;
    long l;
    char by[32];

```

```

while((i = * p++) != 0)
{
    if(i > 0xa1)
    if(f == 0)
    {
        c1 = (i - 0xa1) & 0x07f;
        f = 1;
    }
    else
    {
        c2 = (i - 0xa1) & 0x07f;
        f = 0;
        rec = c1 * 94 + c2;
        l = rec * 32L;
        lseek(handle, l, SEEK_SET);
        read(handle, by, 32);
        for(i1 = 0; i1 < 16; i1++)
            for(i2 = 0; i2 < 2; i2++)
                for(i3 = 0; i3 < 8; i3++)
                    if(getbit(by[i1 * 2 + i2], 7 - i3))
                        putpixel(x + i2 * 8 + i3, y + i1, color);
                        x = x + 16 + z;
    }
}
return(x);
}

int puthz24(int x, int y, int z, int color, char * p)
{
    /* 显示 24 × 24 点阵汉字子函数 */
    unsigned int i, c1, c2, c3, f = 0;
    int i1, i2, i3, rec;
    long l;
    char by[72];
    while((i = * p++) != 0)
    {
        if(i > 0xa1)
        if(f == 0)
        {
            c1 = (i - 0xa1) & 0x07f;
            f = 1;
        }
        else

```

```

    {
        c2 = (i-0xa1)&0x07f;
        f = 0;
        rec = c1 * 94 + c2;
        l = rec * 72L;
        lseek(handle, l, SEEK_SET);
        read(handle, by, 72);
        for(i1 = 0; i1 < 24; i1++)
            for(i2 = 0; i2 <= 2; i2++)
                for(i3 = 0; i3 < 8; i3++)
                    if(getbit(by[i1 * 3 + i2], 7 - i3))
                        putpixel(x + i1, y + i2 * 8 + i3, color);
        x = x + 24 + z;
    }
}

return(x);
}

int getbit(unsigned char c, int n)
{
    return((c >> n) & 1);
}

```

本例运行时在屏幕中央有一个菜单窗口，可利用光标键移动其上的色条选择功能。若按回车键，则立即执行所选功能。程序中将“系统菜单”四个汉字设计成  $24 \times 24$  点阵红色显示，窗口背景用青色，边框用红色，色条中的字用黄色，窗口中其它字用蓝色。基本上可以说得到了最佳的效果。读者只要根据具体情况把菜单内容、窗口大小和汉字显示位置作适当修改，就可以直接使用。

程序中使用的读汉字字模技术可参考第九章，这里不再解释。除此之外，比较关键的部分是光标键控制移动色条和 `menu()` 函数的返回值。光标键控制移动色条又包括色条的产生与消失、色条的移动。色条的产生与消失采用图形屏幕的操作函数 `getimage()` 和 `putimage()`，先用 `getimage()` 函数存储色条大小的屏幕内容，设置 `putimage()` 函数中的参数值使存储在内存中的图像反像释放在同一位置，则可产生出色条，当按光标键要将此色条移走时，只要将存储的图像按复制的方式释放，色条自然也就消失。

对于按垂直方向移动的色条，设定一个色条移动的距离，即相邻两个色条位置的距离，作为  $y$  坐标的增减量。上例中此值为 30。当按回车键时 `menu()` 函数返回一个从上到下色条所在位置的序号，这个序号为 1 时对应最上面一个选择项。只要在调用函数中设置一个 `switch` 开关语句，根据返回的序号就可转去执行相应的功能。本例程中规定了只能用光标键和回车键对菜单进行操作，除此之外，按其它键均不起作用。这样设计的菜单既简单又实用。

### 12.3 FOXBASE 和 Turbo C 程序交替使用时的菜单设计

第十章曾留下一个问题, 就是当较大的 Turbo C 程序调用 FOXBASE 的命令文件时, 由于内存不够用而无法实现, 但这种情况经常存在。这里结合菜单的技术, 介绍一种较实用的巧用 DOS 批处理命令解决这一问题的方法。当然这种方法也可用于 BASIC 语言或其它执行程序。

假定用户编写了一个数据库管理系统, 其命令文件为 MGS.PRG, 还用 Turbo C 编写了一个读取数据库中数据进行模拟决策的系统 DCS.EXE。现在可以编写一个主菜单分别调用这两个系统。菜单的程序如下所示。

#### 例 12-3:

```
#include <conio.h>
#include <stdio.h>
#include <graphics.h>
#include <fcntl.h>
int menu(void);
int getbit(unsigned char, int);
int puthz(int,int,int,int,char *);
int openhz(void);
int handle;
main( )
{
    int gdriver=DETECT,gmode,chose;
    FILE *fp;
    registerbgidriver(EGAVGA__driver);
    initgraph(&gdriver,&gmode,"c:\\tc");
    while(1)
    {
        chose = menu( );
        fp = fopen("chose.bat","w");    /* 创建一个批处理文件只写 */
        switch(chose)
        {
            case 1:
                fputs("echo      Please wait!\n",fp); /* 向文件写入内容 */
                fputs("cd fox\n",fp);                /* 进入 fox 子目录 */
                fputs("foxbase mgs\n",fp);            /* 运行 FOXBASE 数据文件 */
                fputs("cd..\n",fp);                  /* 退出子目录 */
                fputs("menu",fp);                    /* 运行 menu 菜单程序 */
                break;
            case 2:
                fputs("dcs\n",fp);                    /* 运行 DCS.EXE 程序 */
        }
    }
}
```

```

        fputs("menu",fp);                /* 运行 menu 菜单程序 */
        break;
    default:
        break;
    }
    fclose(fp);                          /* 关闭文件 */
    closegraph( );
    exit(0);
}
}
int menu( )
{
    int x,y,key1=0,key2,size,test,i;
    void * buffer;
    unsigned char * fl={"<<","系统菜单",">>"};
    unsigned char * m[ ]={"数据管理子系统","模拟决策子系统","退出系统"};
    setbkcolor(BLUE);
    cleardevice( );
    openhzk( );
    x=puthz(206,100,0,15,fl[0]);
    x=puthz(262,100,8,12,fl[1]);
    puthz(x,100,0,15,fl[2]);
    setcolor(12);
    setlinestyle(0,0,3);
    rectangle(190,154,450,354);
    setfillstyle(1,3);
    floodfill(230,200,12);
    for(i=0;i<2;i++)
        puthz(225,i*50+190, 4, 1, m[i]);
    puthz(255,290,12,1,m[2]);
    close(handle);
    size=imagesize(220,188,422,218);
    buffer=malloc(size);
    getimage(220,188,422,218,buffer);
    putimage(220,188,buffer,NOT_PUT);
    y=188;
    while(key1!=13)                      /* 按回车键结束循环 */
    {
        while(bioskey(1)==0);           /* 等待键盘输入 */
        key1=key2=bioskey(0);           /* 取键盘输入码 */
        key1=key1&0xff;                 /* 只取 ASCII 码 */
    }
}

```



```

key2=key2&0xff70:key2>>8;          /* 只取扩充码 */
if(key2==72||key2==80)              /* 如果为上下箭头键 */
{
    putimage(220,y,buffer,COPY_PUT); /* 恢复屏幕原来信息 */
    if(key2==72)y=y==188?288:y-50;   /* 上箭头键处理 */
    if(key2==80)y=y==288?188:y+50;   /* 下箭头键处理 */
    getimage(220,y,422,y+30,buffer); /* 保存屏幕内容 */
    putimage(220,y,buffer,NOT_PUT);  /* 反像释放产生色条 */
}
}
test=(y-188)/50+1;
free(buffer);
return(test);
}
int openhzk( )                      /* 打开不同字库的函数 */

```

```

        rec = c1 * 94 + c2;
        l = rec * 72L;
        lseek(handle, l, SEEK__SET);
        read(handle, by, 72);
        for(i1 = 0; i1 < 24; i1++)
            for(i2 = 0; i2 <= 2; i2++)
                for(i3 = 0; i3 < 8; i3++)
                    if(getbit(by[i1 * 3 + i2], 7 - i3))
                        putpixel(x + i1, y + i2 * 8 + i3, color);
        x = x + 24 + z;
    }
}
return(x);
}
int getbit(unsigned char c, int n)
{
    return((c >> n) & 1);
}

```

本例中大部分与例 12-2 相同，不同的只是在选择某一功能时，由程序自动向批处理文件 CHOSE.BAT 中写入批命令。设例 12-3 菜单文件的名字为 MENU.EXE，建立一个 MGDCS.BAT 的批处理文件，内容如下：

```

ECHO OFF
MENU
CHOSE
ECHO ON

```

运行整个系统时，只要键入 MGDCS，将先调用菜单 MENU 供用户选择功能，在选择数据库管理系统时，假定 FOXBASE 环境及命令文件 MGS.PRГ 均存放在当前目录的子目录 FOX 下，则如下内容：

```

Please wait
cd fox
foxbase mgs
cd..
menu

```

被存入 CHOSE.BAT 文件中，接着退出菜单并按 CHOSE.BAT 文件中的内容去执行 FOXBASE 的命令文件 MGS.PRГ，运行完毕又调用 MENU 菜单，当选择模拟决策系统时，若假定 DCS.EXE 就放在当前目录下，则将如下内容：

```

dcs
menu

```

存入 CHOSE.BAT 文件中,接着退出菜单并按 CHOSE.BAT 文件的内容去执行用 Turbo C 编写的程序 DCS.EXE,运行完毕又调用 MENU 菜单。若选择的是退出系统,则 CHOSE.BAT 的文件中不写入任何内容也就不作任何操作,从而可退出整个程序。

通过以上方法,彻底消除了运行 FOXBASE 时内存中有 Turbo C 程序存在而导致内存不够用的问题。但美中不足的是:在进入 FOXBASE 后,首先将在屏幕上显示一下 FOXBASE 的文件头,然后才执行命令文件。但可以用 DEBUG 将其去掉,从而在操作者来看,整个系统似乎用一种语言编写,增强了整体效果。还有一点需要提醒一下,在设计 FOXBASE 命令文件结束时,应使用 QUIT 命令退出(对于 BASIC 则必须使用 SYSTEM 语句)。

#### 12.4 中文下拉式菜单的设计

中文下拉式菜单的设计仍然采用西文操作系统图形屏幕模式下,读字模显示汉字的方法。主菜单一般设计成水平方向移动色条进行选择,而每个选择项又可向下弹出一个菜单窗口。注意在弹出之前应将屏幕这一位置的原来内容先保存在堆栈中,仍然采用 getimage( )函数存储。当返回时再逐级向上用 putimage( )函数释放。

下面是一个中文下拉式菜单的一部分程序,可供读者参考。

##### 例 12-4:

```
#include <stdio.h>
#include <stdlib.h>
#include <alloc.h>
#include <graphics.h>
#include <fcntl.h>
int handle,test1,test2,test,test4,test5;
main( )
{
    int gdriver=DETECT,gmode,chose0;
    registerbgidriver(EGAVGA__driver);
    initgraph(&gdriver, &gmode, "c:\\tc");
    setbkcolor(LIGHTBLUE);
    cleardevice( );
    mainmenu( );
    closegraph( );
    exit(0);
}
int mainmenu( )
{
    unsigned char *f[]={ "数据录入", "数据查询", "数据修改",
        "统计分析", "报表打印", "退出系统" }; /* 定义主菜单内容 */
    int i,x,key1,key2,test,size;
    char c;
```

```

void * buffer;
setcolor(12);
rectangle(1,0,637,33);
setfillstyle(1,3);
floodfill(200,20,12);
setcolor(LIGHTRED);
setlinestyle(0,0,3);
rectangle(1,0,637,479);
openhz(c='s');
x=-3;
for(i=0;i<=5;i++)          /* 屏幕顶部显示主菜单内容 */
    x=puthz(10+x,6,0,1,1,1,f[i]);
close(handle);
setcolor(12);
line(1,33,637,33);
line(107,0,107,33);
line(213,0,213,33);
line(319,0,319,33);
line(425,0,425,33);
line(531,0,531,33);
size = imagesize(3,2,105,31);
buffer = malloc(size);
getimage(3,2,105,31,buffer);
putimage(3,2,buffer,NOT_PUT);
x=3;
while(1)
{
    while(key1!=13)          /* 回车终止循环 */
    {
        while(bioskey(1)==0); /* 等待键盘输入 */
        key1=key2=bioskey(0); /* 取键盘输入码 */
        key1=key1&0xff;       /* 只取 ASCII 码 */
        key2=key2&0xff?0:key2>>8; /* 只取扩充码 */
        if(key2==75||key2==77) /* 左右箭头键处理 */
        {
            putimage(x,2,buffer,COPY_PUT); /* 恢复原来信息 */
            if(key2==77)x=x>=533?3:x+106; /* 右箭头键处理 */
            if(key2==75)x=x<=3?533:x-106; /* 左箭头键处理 */
            getimage(x,2,x+102,31,buffer); /* 保存色条大小的屏幕 */
            putimage(x,2,buffer,NOT_PUT); /* 产生色条 */
        }
    }
}

```

```

test = (x-3) / 106 + 1;
switch(test)          /* 选择不同项时分别处理 */
{
    case 1:
        menu11( );
        break;
    case 2:
        menu12( );
        break;
    case 3:
        menu13( );
        break;
    case 4:
        menu14( );
        break;
    case 5:
        menu15( );
        break;
    case 6:
        closegraph( );
        exit(0);
}
key1 = 0;
}
free(buffer);
}
int menu11( )          /* 主菜单第一个选择项的子菜单 */
{
    unsigned char *f[] = {"单个录入","连续录入"};
    int i, y, key1, key2, size0, size1;
    void * buf0, * buf1;
    char c;
    size0 = imagesize(44,35,156,145);
    buf0 = malloc(size0);
    getimage(44,35,156,145,buf0);
    setcolor(12);
    rectangle(45,35,155,144);
    setfillstyle(1,12);
    floodfill(50,50,12);
    setcolor(11);
    setlinestyle(SOLID__LINE,0,1);
    rectangle(50,40,150,140);

```

```

openhz(c = 'o');
for(i = 0; i < 2; i++)
    puthz16(62, i * 40 + 62, 4, 15, f[i]);
close(handle);
size1 = imagesize(60, 60, 140, 80);
buf1 = malloc(size1);
getimage(60, 60, 140, 80, buf1);
putimage(60, 60, buf1, NOT_PUT);
y = 60, key1 = 0;
while(key1 != 13 && key1 != 27)          /* 回车或 Esc 键退出循环 */
{
    while(bioskey() == 0);                /* 等待键盘输入 */
    key1 = key2 = bioskey(0);              /* 取键盘输入码 */
    key1 = key1 & 0xff;                     /* 只取 ASCII 码 */
    key2 = key2 & 0xff70; key2 >> 8;        /* 只取扩充码 */
    if(key2 == 72 || key2 == 80)           /* 上下箭头键处理 */
    {
        putimage(60, y, buf1, COPY_PUT);  /* 恢复原有信息 */
        if(key2 == 72) y = y == 60 ? 100 : y - 40; /* 上箭头处理 */
        if(key2 == 80) y = y == 100 ? 60 : y + 40; /* 下箭头处理 */
        getimage(60, y, 140, y + 20, buf1); /* 保存色条处的屏幕内容 */
        putimage(60, y, buf1, NOT_PUT);    /* 反象释放生成色条 */
    }
}
free(buf1);
if(key1 == 13) test1 = (y - 60) / 40 + 1;
if(key1 == 27) test1 = 0;
putimage(44, 35, buf0, COPY_PUT);
free(buf0);
}

int menu12( )
{
}

int menu13( )
{
}

int menu14( )
{
}

menu15( )
{
}

```

```

int openhz(char c)          /* 打开不同字库的函数 */
{
    if(c == 's')
        handle = open("c:\\\\c:\\c\\clib",O_RDWR|O_BINARY);
    if(c == 'f')
        handle = open("c:\\\\c:\\c\\clibf",O_RDWR|O_BINARY);
    if(c == 'k')
        handle = open("c:\\\\c:\\c\\clibk",O_RDWR|O_BINARY);
    if(c == 'h')
        handle = open("c:\\\\c:\\c\\clibh",O_RDWR|O_BINARY);
    if(c == 'o')
        handle = open("c:\\\\c:\\c\\clib.dat",O_RDWR|O_BINARY);
    if(handle == -1)
    {
        printf("Error on open cclib.dat");
        getch( );
        closegraph( );
        exit(1);
    }
}

int puthz16(int x,int y,int z,int color,char *p) /* 显示 16×16 点阵汉字的子函数 */
{
    unsigned int i, c1, c2, f=0;
    int rec, i1, i2, i3;
    long l;
    char by[32];
    while((i = *p++) != 0)
    {
        if(i > 0xa1)
            if(f == 0)
            {
                c1 = (i-0xa1)&0x07f;
                f = 1;
            }
        else
        {
            c2 = (i-0xa1)&0x07f;
            f = 0;
            rec = c1 * 94 + c2;
            l = rec * 32L;
            lseek(handle,l,SEEK_SET);
            read(handle, by, 32);
        }
    }
}

```

```

        for(i1=0;i1<16;i1++)
            for(i2=0;i2<2;i2++)
                for(i3=0;i3<8;i3++)
                    if(getbit(by[i1*2+i2],7-i3))
                        putpixel(x+i2*8+i3, y+i1, color);
        x=x+z+16;
    }
}
return(x);
}

int puthz(int x,int y,int z,int color,int m,int n,char *p) /* 显示 24×24 点阵汉字的子函数
* /
{
    unsigned int i, c1, c2, f=0;
    int i1,i2,i3,i4,i5,rec;
    long l;
    char by[72];
    while((i=*p++)!=0)
    {
        if(i>0xa1)
            if(f==0)
            {
                c1=(i-0xa1)&0x07f;
                f=1;
            }
        else
        {
            c2=(i-0xa1)&0x07f;
            f=0;
            rec=c1*94+c2;
            l=rec*72L;
            lseek(handle,l,SEEK_SET);
            read(handle, by, 72);
            for(i1=0;i1<24*m;i1=i1+m)
                for(i4=0;i4<m;i4++)
                    for(i2=0;i2<=2;i2++)
                        for(i3=0;i3<8;i3++)
                            if(getbit(by[i1/m*3+i2],7-i3))
                                for(i5=0;i5<n;i5++)
                                    putpixel(x+i1+i4, y+i2*8*n+i3*n+i5, color);
            x=x+24*m+z;
        }
    }
}

```



```

    }
    return(x);
}
int getbit(unsigned char c, int n)
{
    return((c >> n)&1);
}

```

本程序运行时，可用左右光标键移动色条选择主菜单里的内容，回车则进入该选择项下一级的子菜单(本程序只编制了主菜单的第一项)，进入下一级子菜单后可用上下光标键移动色条选择子菜单的项目，回车则认可。若按 Esc 键则退回到主菜单。这个菜单程序设计得比较巧妙而且美观。主菜单用 24×24 点阵汉字，子菜单用 16×16 点阵汉字。读者可以按同样的方法向其中填入内容来完成设计，从而可作成完整实用的下拉式菜单。

## 第十三章 与汇编语言的接口技术

Turbo C 尽管具有强大的功能和高效的集成开发环境,但在某些特殊场合,如需访问计算机系统的硬件资源或操作系统的功能,还有要求操作速度特别快的时候,用汇编语言实现比用 C 语言实现更能满足要求。本章主要讲述有关 Turbo C 与汇编语言的接口技术,其中包括 Turbo C 调用汇编子程序和 Turbo C 行间嵌入汇编语句等内容。

### 13.1 Turbo C 调用汇编子程序

汇编语言本质上是机器语言的符号表示,它直接对硬件设备进行操作,是一个执行速度快、功能强的语言系统,从这两方面来说,汇编语言算得上首屈一指。但是,汇编语言编程复杂,对硬件依赖性强是它的致命弱点。所以如果使用 Turbo C 作为主要编程语言,再加上一些汇编子模块完成个别特殊功能,可以说是最佳配合了。一般来说,在高级语言中使用汇编语言主要有以下几个原因:

1. 为了提高程序某些关键部分的执行速度,可以用汇编语言把关键代码段的时钟周期数减到最少;
2. 完成一些高级语言所难以实现的功能,如高分辨率绘图;
3. 引用已经开发的汇编语言模块。

由于不同的微处理器各有一套不同的汇编指令,本书仅就通用的 8086/8088/80286 微处理器和 MS-DOS 操作系统,介绍汇编语言与 Turbo C 的接口。另外,要实现这种接口,除了需要 Turbo C 系统外,还应拥有 Microsoft 公司出版的 MASM4.0 以上版本的宏汇编编译程序,用来编译汇编语言程序。

#### 13.1.1 Turbo C 与汇编语言的接口方法

当 Turbo C 调用汇编语言程序时,汇编程序指令序列应当具有一定的顺序,这种顺序可描述为:

正文段描述  
段模式  
组描述  
进栈  
程序体  
退栈  
正文段结束

如果一组汇编程序不符合上面的顺序,则 Turbo C 将不能对其进行调用,下面通过一个实例来说明 Turbo C 调用汇编语言的方法。

设有一个 Turbo C 程序从键盘上获得两个数,并将其传递给汇编语言子程序

ASMT.C.ASM 完成这两个数的相乘并返回乘积，然后在 Turbo C 程序中将结果显示在屏幕上。

先建立一个 Turbo C 主程序 EXAM13\_1.C 如下：

**例 13-1:**

```
#include <stdio.h>
int asmte(int,int,long *);
main( )
{
    int i, j;
    long k;
    printf("Please input i,j = ?");
    scanf("%d,%d",&i,&j);
    asmte(i,j,&k);
    printf("the %d times %d is %ld\n",i,j,k);
    getch( );
}
```

其中 asmte( ) 是调用汇编语言子程序的函数，该函数含有三个参数，前两个为整型数，第三个为长整型数指针。可以通过三个参数的传递方法来弄清 Turbo C 与汇编语言的各种参数传递。

由上述 Turbo C 主程序调用的汇编语言 ASMT.C.ASM 的子程序如下：

```
                PUBLIC  __asmte
__TEXT  SEGMENT BYTE    PUBLIC 'CODE'
DGROUP  GROUP    __DATA, __BSS
__DATA  SEGMENT WORD    PUBLIC 'DATA'
__DATA  ENDS
__BSS   SEGMENT WORD    PUBLIC 'BSS'
__BSS   ENDS
        ASSUME CS:__TEXT, DS:DGROUP, SS:DGROUP
__asmte  proc      near
            push    bp
            mov     bp,sp
            push    si
            mov     ax,[bp+4]    /* 得到第一个参数i */
            mov     bx,[bp+6]    /* 得到第二个参数j */
            mul     bx          /* 两数相乘 */
            mov     si,[bp+8]    /* 得到参数k的地址 */
            mov     [si],ax      /* 送乘积的低位两个字节 */
            inc     si
            inc     si
            mov     [si],dx      /* 送乘积的高位两个字节 */
__asmte  endp
```

```

        pop     si
        pop     bp
        ret
__asmtc  endp
__TEXT  ENDS
END

```

其中有几个不同的段模式，汇编语言的段模式用于存放不同类型的数据或信息，在不同的 Turbo C 语言模式下，汇编语言的段模式是不同的。本书仅介绍常用的 Turbo C 小模式(small 模式)的情况。

\_\_TEXT 段是一个代码段，它以 \_\_TEXT SEGMENT BYTE PUBLIC 'CODE' 开始，以 \_\_TEXT ENDS 结束。

\_\_DATA 段用来存放所有初始化了的全程数据和静态数据，它以 \_\_DATA ENDS 结束。

\_\_BSS 段用来存放未初始化的静态数据，它以 \_\_BSS ENDS 结束。

组描述是指将各段放在同一个组中，这样允许通过用同一个段寄存器访问一个组里的各个段。如 GROUP \_\_DATA, \_\_BSS。需要指出的是，在汇编语言中用 ASSUME 伪指令时，DS、SS 都应指向 DGROUP。

进/退栈操作是为了保存有关寄存器的原始值，汇编语言中指令过程为：

```

        push bp
        mov bp,sp
        .
        .
        .
        mov sp,bp
        pop bp

```

另外，在栈操作时，如果汇编程序正文中需要用到某些寄存器(如 DI、SI)，则须将这些寄存器压入堆栈，待程序正文将要结束时，再弹出堆栈。

以上给出由 Turbo C 调用的汇编程序的结构，在调用中还有一个问题就是参数的传递。参数的传递包括两个方面，一个是从 Turbo C 程序中向汇编子模块传递参数，另一个是从汇编语言向 Turbo C 调用程序返回参数。

Turbo C 程序向汇编程序的参数传递是通过栈操作进行的，先传递的参数被最后压入堆栈，即传递参数按出现顺序的相反次序被压入堆栈。而且第一个参数总是位于堆栈的顶部，堆栈的地址变化是向下增长的。最后一个进入堆栈的参数总是在内存的低端。如上例中的调用，先入栈的是变量 k 的地址，再是 j，最后才是 i。Turbo C 不同类型的参数占用堆栈的字节数见表 13-1。

汇编语言程序运行的结果返回给 Turbo C 主程序时是通过 AX 和 DX 寄存器来完成的，许多情况只需 AX 就可将汇编程序的结果返回给 Turbo C 程序。汇编语言的返回值与 Turbo C 的各种数据类型的对应关系如表 13-2 所示。

表 13-1 数据类型占用堆栈的字节数

数据类型	字节数
char	2
unsigned char	2
short	2
unsigned short	2
int	2
unsigned int	2
long	4
unsigned long	4
float	4
double	8
near 指针	2(偏移量)
far 指针	4(段和偏移量)

表 13-2 Turbo C 数据类型与汇编语言返回值的对应关系

数据类型	返回值寄存器
char	AX
unsigned char	AX
int	AX
unsigned int	AX
long	高位字节在 DX 中, 低位字节在 AX 中
unsigned long	高位字节在 DX 中, 低位字节在 AX 中
float	高位字节在 DX 中, 低位字节在 AX 中
double	值的地址在 AX 中
struct & union	值的地址在 AX 中
near 指针	AX
far 指针	DX 中为段地址, AX 中为偏移量

### 13.1.2 自动产生汇编语言的框架程序

可以按照上节介绍的接口方法编制汇编子模块, 但 Turbo C 提供了一种自动产生汇编语言框架的办法使用更加方便。下面通过举例进行说明, 假设 Turbo C 主程序仍为前面的 EXAM13\_1.C, 现在的目的就是用 Turbo C 命令行编译程序 TCC.EXE 的设置开关 -S, 来产生一个汇编接口框架程序。一旦有了这个框架程序, 剩下的工作就是在框架程序中填上完成特定功能的汇编语句。

首先用 Turbo C 写一个与调用汇编程序的函数名相同的空的函数, 对例 13-1 应写成:

```
asmtc( )
{
}
```

给此函数取名 ASMTC.C(取名可以任意)予以存放, 然后使用命令:

```
tcc -S asmtc
```

进行编译, 编译后自动生成一个名为 ASMTC.ASM 的文件, 该文件的格式如下:

```
        ifndef ??version
?debug  macro
        endm
        endif
        ?debug S "asmtc.c"

__TEXT  segment      byte public 'CODE'
DGROUP  group  __DATA,__BSS
        assume cs:__TEXT,ds:DGROUP,ss:DGROUP

__TEXT  ends
__DATA  segment word public 'DATA'
d@      label  byte
d@w     label  word
__DATA  ends
__BSS   segment word public 'BSS'
b@      label  byte
b@w     label  word
        ?debug C E9797B9E180761736D74632E63

__BSS   ends
__TEXT  segment      byte public 'CODE'
;       ?debug L 1
__asmtc proc  near
@1:
;       ?debug L 3
        ret
__asmtc endp
__TEXT  ends
        ?debug C E9
__DATA  segment word public 'DATA'
s@      label  byte
__DATA  ends
__TEXT  segment      byte public 'CODE'
__TEXT  ends
        public __asmtc
        end
```

在这个框架式汇编语言程序中, 只要在"@1:"处插入实现该函数特定功能的汇编语句和有关进栈、退栈以及参数传递的语句即可。用这种方法建立的汇编接口程序一定能准确地与 Turbo C 主程序相接。

在该例的汇编语言框架中的"@1:"处, 插入完成乘法的汇编程序后形成了最终的汇编

# 子程序 ASMTC.ASM:

```

        ifndef ??version
?debug  macro
        cndm
        endif
        ?debug S "asmtc.c"
__TEXT  segment          byte public 'CODE'
DGROUP  group  __DATA,__BSS
        assume cs:__TEXT,ds:DGROUP,ss:DGROUP
__TEXT  ends
__DATA  segment word public 'DATA'
d@      label  byte
d@w     label  word
__DATA  ends
__BSS   segment word public 'BSS'
b@      label  byte
b@w     label  word
        ?debug C E9797B9E180761736D74632E63
__BSS   ends
__TEXT  segment          byte public 'CODE'
;       ?debug L 1
__asmtc proc  near
@1:
        push    bp
        mov     bp,sp
        push    si
        mov     ax,[bp+4]
        mov     bx,[bp+6]
        mul     bx
        mov     si,[bp+8]
        mov     [si],ax
        inc     si
        inc     si
        mov     [si],dx
        pop     si
        pop     bp
;       ?debug L 3
        ret
__asmtc  endp
__TEXT  ends
        ?debug C E9
__DATA  segment word public 'DATA'

```

```

s@      label byte
__DATA  ends
__TEXT  segment      byte public 'CODE'
__TEXT  ends
        public __asmtc
        end

```

### 13.1.3 接口程序的编译、链接和运行

编写了汇编语言程序和调用它的 Turbo C 主程序以后，接下来就是将它们编译、链接成可执行文件。首先用宏汇编程序 MASM.EXE 编译汇编程序，键入：

```

MASM ASMTC.ASM;

```

编译后将生成一个名为 ASMTC.OBJ 的目标文件。

再向 Turbo C 集成开发环境 Project 项的 Project name 中，写入一个 .PRJ 的项目文件，本例设为 EXAM13\_1.PRJ，其内容如下：

```

EXAM13_1.C

```

```

ASMTC.OBJ

```

最后用 F9 键对 Turbo C 主程序进行编译，编译后自动与汇编模块 ASMTC.OBJ 进行链接并生成一个 EXAM13\_1.EXE 的执行文件。

由于 Turbo C 对大小写字母是很敏感的，因此就必须使 Turbo C 主程序中汇编模块调用语句的函数名与汇编语言中的过程名大小写一致，否则将不能正确地链接。但如果在用 Turbo C 进行编译链接之前，将集成开发环境的 Options/Linker 中大小写敏感选择开关的 Case sensitive link 置成关闭状态，再进行编译链接，将不会再区分大小写字母。

下面详细叙述用 TCC 选择项 -S 产生汇编框架的办法来实现 Turbo C 调用汇编子程序的整个过程。仍以上面的例程名来说明：

一、编写一个 Turbo C 调用程序，如 EXAM13\_1.C。

二、用 TCC 的 -S 选择项产生一个供 Turbo C 主程序调用的汇编语言框架。按下述步骤来建立：

1. 按主程序中调用汇编子程序的函数名写一个 Turbo C 函数，但此函数中无任何内容。如 ASMTC.C。

2. 用命令 TCC -S ASMTC 就可以产生一个文件名为 ASMTC.ASM 的汇编程序框架。

三、向生成的汇编语言框架中“@1:”后插入汇编语言程序。程序应包含以下内容：

1. 开始时将 BP 寄存器的内容保护入栈，再将栈指针 SP 送入 BP 中。

2. 如果汇编语言程序中还使用了其它寄存器，也需保护入栈。

3. 接收由主程序传递的参数。

4. 接下来才是实现具体功能的汇编程序。

5. 将其它寄存器退栈，并使 BP 复原后返回。如有返回参数，返回前则必须根据参数的数据类型将 AX 或 AX、DX 的值返回给调用函数。



#### 四、编译和链接:

1. 用 MASM 宏汇编程序编译生成的汇编语言子程序, 并确认编译中无出错, 编译后生成一个 .OBJ 目标文件。上例产生的目标文件为 ASMTC.OBJ。

2. 在 Turbo C 中建立一个项目文件, 文件内容应包括要编译的 Turbo C 源文件名和汇编语言目标文件名。上例建立了一个 EXAM13\_1.PRJ 的项目文件。

3. 将 Turbo C 集成开发环境中 Options / Linker / Case-sensitive link 开关置为 off。

4. 用 F9 编译链接, 可生成一个执行文件, 上例生成的执行文件为 EXAM13\_1.EXE。

注:

● 用 TCC 命令的 -S 选择项生成汇编语言框架的方法既方便又准确, 因此是值得推荐的方法。

### 13.2 Turbo C 行间嵌入汇编语句

尽管用 Turbo C 调用汇编子模块的方法可实现与汇编语言的接口, 但当汇编语言较短时, 这种调用就显得比较繁琐。Turbo C 提供一种可以在汇编语句前加上关键字 asm 直接作为其程序一部分的方法, 即 Turbo C 允许行间嵌入汇编代码。

Turbo C 语言源程序使用行间嵌入汇编代码时必须满足以下约定:

1. 行间汇编语句前必须有 asm 关键字开头, 其后是汇编语句;

2. 可以自由地与正常的 Turbo C 语句混在一起使用。可以用 Turbo C 的分隔符“;”, 也可以用换行符分隔。

3. 不能象在汇编语言中那样使用分号作为注释分界符, 而必须用 Turbo C 语言的分界符“/\*”和“\*/”。

下面通过一个例子来说明行间嵌入汇编代码的方法。

一、编写行间嵌入汇编语句的 Turbo C 源程序, 取名为 EXAM13-2.C, 本例中用汇编语句实现乘 2 的操作。程序如下:

#### 例 13-2:

```
main( )
{
    int i, j;
    char *s;
    printf("Please input: i= ");
    scanf("%d",&i);
    asm mov ax, i;      /* 将输入的值送入 ax */
    asm mov cl, 2;      /* 将 2 送入 cl */
    asm mul cl;         /* ax 乘 cl, 乘积在 ax 中 */
    asm mov j, ax;      /* 将乘积送给 j */
    printf("The result is: %d x 2 = %d",i,j); /* 输出结果 */
    getch( );
}
```

}

二、将 Microsoft 公司的 MASM.EXE 宏汇编编译程序更名为 TASM.EXE, 放入 TC 子目录中。

三、用 Turbo C 的命令行编译程序 TCC.EXE 编译, 链接行间嵌入汇编语句的 Turbo C 源程序, 使用格式为:

```
tcc -B -Lxxx 文件名 库文件名
```

其中: xxx 为库文件所在目录的路径, 文件名为带行间嵌入汇编语句的 Turbo C 文件名, 库文件名为程序中使用的 Turbo C 函数所在的库文件(Turbo C 的标准库可省略)。

本例没有用到库文件, 所以可用下面的语句编译并链接:

```
tcc -B exam13_2
```

执行后生成一个名为 EXAM13\_2.EXE 的执行文件。

下面再来看一个例子。这个程序分别用 Turbo C 的图形初始化函数和汇编语言将屏幕设置成图形模式, 其中有两段汇编程序, 主函数中的汇编程序段将屏幕设置成有 256 种颜色的 320×200 分辨率图形模式, 并画出 256 条不同颜色的竖线, 子函数 fun1() 中的汇编语言段则是将屏幕设置成 VGA 高分辨率图形模式, 并在屏幕中央画出一个实方块。

#### 例 13-3:

```
#include <dos.h>
#include <graphics.h>
main( )
{
    int i = 256;
    int gdriver = DETECT, gmode;
    registerbgidriver(EGAVGA_driver);
    initgraph(&gdriver, &gmode, "c:\\tc"); /* 用 Turbo C 初始化图形屏幕 */
    bar3d(100, 200, 200, 300, 100, 1);
    printf("In Turbo C    VGA: 640 x 480, 16 color");
    sleep(3);
    asm          mov    bl, i;      /* BIOS 功能调用, 设置屏幕 */
    asm          mov    ah, 0;
    asm          mov    al, 13h;
    asm          int    10h;
    asm          mov    cx, 290;    /* 画出 256 条不同颜色的直线 */
loop:   asm          mov    dx, 199;
loop:   asm          mov    ah, 0ch;
        asm          mov    al, bl;
        asm          mov    bh, 0;
        asm          int    10h;
        asm          dec    dx;
        asm          cmp    dx, 0;
        asm          jnc    loop;
```

```

asm      dec  cx;
asm      dec  bl;
asm      cmp  bl, 0;
asm      jnc  lop;
printf("Assembly  VGA: 320x200, 256 color");
sleep(3);          /* 延时3秒钟 */
fun1( );
sleep(3);
closegraph( );
printf("The end");
sleep(1);
}
fun1( )
{
asm  mov  ah, 0;      /* 设置屏幕 */
asm  mov  al, 12h;
asm  int  10h;
asm  mov  dx, 240
lop1:  asm  mov  cx, 440      /* 按画线方法画实方块 */
lop2:  asm  mov  ah, 0ch
asm  mov  al, 4
asm  mov  bh, 0
asm  int  10h
asm  dec  cx
asm  cmp  cx, 200
asm  jnc  lop2
asm  dec  dx
asm  cmp  dx, 120
asm  jnc  lop1
printf("Assembly  VGA: 640 x 480, 16 color");
}

```

注:

● 对有标号的汇编语句, 关键字 `asm` 应放在标号后。

本例用到了 Turbo C 图形函数库, 所以必须用下面的语句编译和链接:

```
tcc -B -Lc:\tc\lib exam13_3 graphics.lib
```

编译后产生一个 EXAM13\_3.EXE 的可执行文件。

# 附录 ASCII 字符代码表

## 1. 屏幕显示输出字符

ASCII码 十进制值	ASCII码 十六进制值	代表字符	ASCII码 十进制值	ASCII码 十六进制值	代表字符	ASCII码 十进制值	ASCII码 十六进制值	代表字符	ASCII码 十进制值	ASCII码 十六进制值	代表字符
00	00	空	32	20	空格	64	40	Ⓐ	96	60	'
01	01	Ⓐ	33	21	!	65	41	A	97	61	a
02	02	●	34	22	"	66	42	B	98	62	b
03	03	♥	35	23	#	67	43	C	99	63	c
05	05	♦	36	24	\$	68	44	D	100	64	d
04	04	♣	37	25	%	69	45	E	101	65	e
06	06	♠	38	26	&	70	46	F	102	66	f
07	07	•	39	27	'	71	47	G	103	67	g
08	08	■	40	28	<	72	48	H	104	68	h
09	09	○	41	29	>	73	49	I	105	69	i
10	0A	■	42	2A	*	74	4A	J	106	6A	j
11	0B	Ⓐ	43	2B	+	75	4B	K	107	6B	k
12	0C	?	44	2C	,	76	4C	L	108	6C	l
13	0D	♪	45	2D	-	77	4D	M	109	6D	m
14	0E	♪	46	2E	.	78	4E	N	110	6E	n
15	0F	♪	47	2F	/	79	4F	O	111	6F	o
16	10	▼	48	30	0	80	50	P	112	70	p
17	11	▲	49	31	1	81	51	Q	113	71	q
18	12	↑	50	32	2	82	52	R	114	72	r
19	13	!!	51	33	3	83	53	S	115	73	s
20	14	!!	52	34	4	84	54	T	116	74	t
21	15	§	53	35	5	85	55	U	117	75	u
22	16	-	54	36	6	86	56	V	118	76	v
23	17	↓	55	37	7	87	57	W	119	77	w
24	18	↑	56	38	8	88	58	X	120	78	x
25	19	↑	57	39	9	89	59	Y	121	79	y
26	1A	→	58	3A	+	90	5A	Z	122	7A	z
27	1B	→	59	3B	;	91	5B	[	123	7B	{
28	1C	└	60	3C	<	92	5C	\	124	7C	
29	1D	↑	61	3D	=	93	5D	]	125	7D	}
30	1E	▲	62	3E	>	94	5E	^	126	7E	~
31	1F	▼	63	3F	?	95	5F	-	127	7F	△

续

ASCII码 十进制值	ASCII码 十六进制值	代表字符	ASCII码 十进制值	ASCII码 十六进制值	代表字符	ASCII码 十进制值	ASCII码 十六进制值	代表字符	ASCII码 十进制值	ASCII码 十六进制值	代表字符
128	80	Ç	160	A0	á	192	C0	Ł	224	E0	œ
129	81	ü	161	A1	í	193	C1	ł	225	E1	ð
130	82	ë	162	A2	ó	194	C2	ŧ	226	E2	ŕ
131	83	â	163	A3	ô	195	C3	ŧ	227	E3	π
132	84	ä	164	A4	ñ	196	C4	ı	228	E4	Σ
133	85	à	165	A5	ñ	197	C5	ı	229	E5	σ
134	86	â	166	A6	ä	198	C6	ı	230	E6	u
135	87	ç	167	A7	ö	199	C7	ı	231	E7	ŕ
136	88	ê	168	A8	ç	200	C8	ı	232	E8	ş
137	89	ë	169	A9	ı	201	C9	ı	233	E9	o
138	8A	è	170	AA	ı	202	CA	ı	234	EA	Ω
139	8B	ı	171	AB	1/2	203	CB	ı	235	EB	ø
140	8C	ı	172	AC	1/4	204	CC	ı	236	EC	8
141	8D	ı	173	AD	ı	205	CD	ı	237	ED	•
142	8E	Ä	174	AE	«	206	CE	ı	238	EE	ƒ
143	8F	Å	175	AF	»	207	CF	ı	239	EF	ƒ
144	90	É	176	B0	ı	208	D0	ı	240	FO	ƒ
145	91	ı	177	B1	ı	209	D1	ı	241	F1	ı
146	92	Æ	178	B2	ı	210	D2	ı	242	F2	ı
147	93	Ö	179	B3	ı	211	D3	ı	243	F3	ı
148	94	Ü	180	B4	ı	212	D4	ı	244	FA	ı
149	95	Ö	181	B5	ı	213	D5	ı	245	FB	ı
150	96	Ö	182	B6	ı	214	D6	ı	246	F6	ı
151	97	Ö	183	B7	ı	215	D7	ı	247	F7	ı
152	98	ÿ	184	B8	ı	216	D8	ı	248	F8	ı
153	99	Ö	185	B9	ı	217	D9	ı	249	F9	ı
154	9A	Ü	186	BA	ı	218	DA	ı	250	FA	ı
155	9B	ı	187	BB	ı	219	DB	ı	251	FB	ı
156	9C	ı	188	BC	ı	220	DC	ı	252	FC	ı
157	9D	ı	189	BD	ı	221	DD	ı	253	FD	ı
158	9E	ı	190	BE	ı	222	DE	ı	254	FE	ı
159	9F	ı	191	BF	ı	223	DF	ı	255	FF	ı

## 2. 键盘输入控制字符 (00 到 31)

ASCII 码	输入字符	控制符	作用	ASCII 码	输入字符	控制符	作用
00	Ctrl-Z	NUL		16	Ctrl-P	DLE	
01	Ctrl-A	SOH		17	Ctrl-Q	DC1	
02	Ctrl-B	STX		18	Ctrl-R	DC2	
03	Ctrl-C	ETX		19	Ctrl-S	DC3	
04	Ctrl-D	EOT		20	Ctrl-T	DC4	
05	Ctrl-E	ENO		21	Ctrl-U	NAK	
06	Ctrl-F	ACK		22	Ctrl-V	SYN	
07	Ctrl-G	BEL	喇叭发声	23	Ctrl-W	ETB	
08	BS	BS	回 空 格	24	Ctrl-X	CAN	
09	→I	HT	光标跳移	25	Ctrl-Y	EM	
10	Ctrl-J	LF	换 行	26	Ctrl-Z	SUB	
11	Ctrl-K	VT	光标左上角	27	ESC	ESC	
12	Ctrl-L	FF	换 页	28	Ctrl-\	FS	光标右移
13	↵	CR	回车换行	29	Ctrl-]	GS	光标左移
14	Ctrl-N	SO		30	Ctrl-6	RS	光标上移
15	Ctrl-O	SI		31	Ctrl-—	US	光标下移

## 3. 扩充 ASCII 码字符

扩充 ASCII 码字符由两个代码组成, 第一个代码是 00, 第二个代码表如下。

扩充 ASCII 码字符第二个代码表

第二代码	相 对 应 按 键	第二代码	相 对 应 按 键
15	←	82	Ins
16-25	Alt-Q,W,E,R,T,Y,U,I,O,P	83	Del
30-38	Alt-A,S,D,F,G,H,J,K,L,	84-93	F11—F20(Shift—F1 到 F10)
44-50	Alt-Z,X,C,V,B,N,M	94-103	F21—F30(ctrl—F1 到 F10)
59-68	功能键 F1 到 F10	104-113	F31—F40(Alt—F1 到 F10)
71	Home	114	Ctrl—Prtsc
72	↑	115	Ctrl—←(前 一个字)
73	PgUP	116	Ctrl—→(下 一个字)
75	←	117	Ctrl—End
77	→	118	Ctrl—PgDn
79	End	119	Ctrl—Home
80	↓	120-131	Alt—1,2,3,4,5,6,7,8,9,0,—,=
81	PgDn	132	Ctrl—PgUp