

# 目 录

## 第一章 Turbo C

- 1.1 C语言与 Turbo C ..... (1)
- 1.2 可执行程序的产生过程 ..... (2)
- 1.3 Turbo C 2.0 特点与配置要求 ..... (2)
- 1.4 Turbo C 2.0 的内容 ..... (3)
- 1.5 Turbo C 2.0 的安装 ..... (5)
- 1.6 综合开发环境编译程序 TC ..... (6)

## 第二章 Turbo C 基本数据

- 2.1 基本数据类型 ..... (8)
- 2.2 修饰符 ..... (8)
- 2.3 常量 ..... (9)
- 2.4 变量及其初始化 ..... (9)
- 2.5 函数 ..... (10)
- 2.6 表达式 ..... (10)
- 2.7 程序、文件、函数三者的关系 ..... (11)
- 2.8 按格式输入函数 scanf() ..... (12)
- 2.9 函数调用形式 ..... (13)
- 2.10 数据的存储类别 ..... (15)

## 第三章 命令行 Turbo C 和实用程序

- 3.1 命令行 Turbo C ..... (18)
- 3.2 预处理程序 CPP ..... (22)
- 3.3 程序管理工具 MAKE ..... (23)
- 3.4 Touch ..... (27)
- 3.5 Grep ..... (27)

## 第四章 C 运算符

- 4.1 算术、逻辑运算符 ..... (28)
- 4.2 递增、递减、赋值运算符 ..... (30)
- 4.3 求字节数、条件运算、逗号 ..... (31)
- 4.4 指针的概念及其运算符 ..... (32)
- 4.5 类型显式转换运算符 ..... (33)
- 4.6 运算符优先级 ..... (35)

## 第五章 程序控制结构

- 5.1 分支结构 ..... (37)
- 5.2 循环结构 ..... (41)
- 5.3 中断结构 ..... (44)
- 5.4 递归结构 ..... (46)

## 第六章 宏指令的用法

6.1 #define 与 #undef .....	(50)
6.2 #include .....	(52)
6.3 条件编译.....	(54)
6.4 #line .....	(56)
6.5 #pragma .....	(57)
6.6 #error .....	(58)
6.7 内部宏名.....	(58)

## 第七章 结构数据

7.1 列举.....	(60)
7.2 数组.....	(61)
7.3 结构.....	(67)
7.4 字段结构.....	(69)
7.5 共同体.....	(71)

## 第八章 指针与数组

8.1 一维数组的指针.....	(74)
8.2 数组的动态分配.....	(74)
8.3 指针和字符串.....	(75)
8.4 指针的运算性质.....	(76)
8.5 指针与多维数组.....	(81)
8.6 指针数组.....	(82)
8.7 指向指针的指针.....	(83)
8.8 数组指针.....	(85)

## 第九章 指针与函数、结构

9.1 函数指针.....	(87)
9.2 返回指针的函数.....	(89)
9.3 指针参数与函数参数值传递方法.....	(90)
9.4 指针与结构.....	(92)

## 第十章 文件管理

10.1 文件的概念 .....	(96)
10.2 标准 I/O .....	(97)
10.3 缓冲型文件的输入输出.....	(100)
10.4 非缓冲型文件的输入输出.....	(104)
10.5 文件的随机存取.....	(107)

## 第十一章 字符屏幕管理

11.1 PC 机显示适配器及其模式 .....	(110)
11.2 用于窗口的 I/O 函数 .....	(111)
11.3 屏幕操作函数.....	(111)
11.4 字符属性控制.....	(113)
11.5 字符屏显状态.....	(114)

11.6	directvideo 变量	(115)
<b>第十二章 绘图</b>		
12.1	Turbo C 的绘图系统	(116)
12.2	调色板和颜色的设置	(118)
12.3	基本图形函数	(120)
12.4	填充模式函数	(122)
12.5	着色图形函数	(123)
12.6	图形模式下的汉字输出	(125)
12.7	图形屏幕管理函数	(126)
12.8	图形信息管理函数	(128)
<b>第十三章 C 与汇编语言的混合编程</b>		
13.1	C 调用汇编语言子程序	(133)
13.2	汇编语言程序调用 C 函数	(144)
13.3	C 程序中插入指令行	(146)
<b>第十四章 PC 机硬件资源管理</b>		
14.1	PC 硬件资源	(153)
14.2	使用指令直接访问硬件	(156)
14.3	利用库函数管理相应硬件	(161)
14.4	执行 BIOS 软中断访问相应硬件	(169)
14.5	调用 DOS 系统功能访问相应硬件	(183)
<b>第十五章 排序与检索算法及其实现</b>		
15.1	排序的基本方法	(188)
15.2	改进型排序方法	(191)
15.3	结构数据的排序	(193)
15.4	归并排序法	(197)
15.5	检索	(200)
<b>第十六章 数据结构及其实现</b>		
16.1	队列	(205)
16.2	堆栈	(211)
16.3	单链表	(214)
16.4	双链表	(221)
16.5	二叉树	(231)
<b>第十七章 C 与 FoxBASE 的接口程序设计</b>		
17.1	接口的概念	(241)
17.2	FoxBASE 的文件类型	(242)
17.3	FoxBASE 数据库文件结构	(243)
17.4	C 如何读取数据库文件的说明信息	(245)
17.5	记录的定位、录入与修改	(245)
17.6	利用由数据库文件建立的文本文件作为数据缓存区的 FoxBASE 与 C 的接口程序设计	(254)

17.7 利用由 FoxBASE 直接建立的文本文件作为数据缓存区的 FoxBASE 与 C 的接口程序设计 .....	(258)
--	-------

# 第一章 Turbo C

我们在为计算机编制程序时究竟使用哪种语言好呢?这要根据实际情况来确定,也就是根据资源条件(包括技术力量)和要实现的功能来确定,即因地制宜。

目前,C语言受到人们的普遍喜爱。因为它既是成功的系统描述语言,又是通用的程序设计语言。作为系统描述语言,用它成功地编写出 UNIX、FoxBASE 等。C语言的可读性和可移植性比起汇编语言来要好得多,而代码效率却只比汇编语言低 10~20%,因此,人们给 C 起了个美称,叫便携汇编语言。作为通用的程序设计语言,越来越多的人使用它,替代其它高级语言,编写信息系统软件。目前,C语言已成为世界发达国家软件开发的主流语言。

## 1.1 C语言与 Turbo C

C语言是 70 年代贝尔实验室(Bell Laboratories)为描述 UNIX 操作系统和 C 编译程序,而开发的一种系统描述语言。

1969 年,美国贝尔实验室的两个研究人员 Ken Thompson 和 Dennis M. Ritchie,开始开发 UNIX,用了不到两个人年的时间就研制成功了,当时的 UNIX 版本是用汇编语言编写的。

1970 年,K. Thompson 为了提高 UNIX 的可读性和可移植性,在 BCPL 语言的基础上,开发了一种新的语言,起名叫“B”,之所以叫这个名字,据说其中一个因素就是根据 BCPL 的字头。由于 B 语言存在着一些缺点,例如,它没有定义数据类型,这就无法支持多种数据类型。因此,该语言没有流行起来。

从 1971 年开始,D. M. Ritchie 用了一年左右的时间,在 B 的基础上开发了第一个 C 编译程序,1972 年投入使用,C语言就是这样诞生的。因为这个编译程序是在 B 语言的基础上开发的,无论是在英文字母序列中也好,还是在 BCPL 这个名字中也好,排在 B 后面的均为 C,故起名为 C 语言。

1973 年,K. Thompson 和 D. M. Ritchie 两人合作,用 C 语言把 UNIX 又写了一遍,这就为 UNIX 的移植和发展奠定了基础。

1975 年 Brian W. Kernighan 和 D. M. Ritchie 合写了名为《The C Programming Language》的著作,为 C 语言在全世界范围内的推广与普及提供了一本很好的教科书,被世人誉为标准版本。1981 年日本著名学者石田晴久把这本书译为日文。1982 年我国 UNIX、C 的著名专家孙玉方、孟庆昌先生把这本书编译为中文教材。从此,C语言便越来越受到软件工程人员的喜爱,他们放弃原来已熟悉的 FORTRAN、Pascal 等语言,而使用 C 语言开发软件。

C语言开始是附属于 UNIX,运行在 PDP-11 机上,但到 1978 年以后,就移植到各种微机上,能在各种操作系统下运行。这就出现了各种 C 编译系统。这些 C 编译系统的规格是不尽相同的,因此,用户编辑的源程序要与所使用的编译系统对应,即各种编译系统下所编的源程序不具有互换性。

为提高互换性,美国国家标准学会(ANSI)的信息系统委员会(X3 委员会)中的 J11 组对 C 语言进行了规范化,于 1987 年提出了美国国家标准 C(ANSI C)方案,得到了各国的承认。最近推出的

一些 C 编译系统几乎都符合 ANSI C 标准。

Turbo C 编译系统是 Borland International 公司的产品,它支持 C 标准版本,支持 ANSI C 方案,备有与硬件有关的特殊函数。该编译系统深受国内外软件人员的欢迎。它有漂亮的编辑程序;运行速度较快;系统不算很大且功能较强,又具有现代 C 的风格。

## 1.2 可执行程序的产生过程

从要解决的问题到可执行程序要经过以下 6 步:

- (1)把要解决的问题分为几个小问题,明确它们的联系及顺序;
- (2)说明每个小问题的详细内容,以及有关算法;
- (3)使用编辑程序,用 C 语言编写源程序;
- (4)使用 C 编译程序,编译由(3)得到的 C 源程序;
- (5)如果编译发生错误,则要修正源程序,反复进行(4)和(5),直到没有错误为止。
- (6)使用链接程序,把编译后的目标文件变为可执行文件。如果发生错误,则要变更源文件或编译和链接的环境,重新编译和链接,直到没有错误为止。

Turbo C 是在 MS-DOS 操作系统上使用的 C 编译系统。因此,若使用 Turbo C 就必须能熟练使用 MS-DOS,这就要掌握 MS-DOS 的基本知识。

在 Turbo C 中,编辑程序、编译程序、链接程序等全部都是独立的程序。然而,Turbo C 提供了统一调配这些程序的开发环境,使得 Turbo C 用起来很方便。可以说,Turbo C 是编辑、编译、链接、调试各功能的一体化、有机结合的产物。

## 1.3 Turbo C 2.0 特点与配置要求

Turbo C 2.0 是 Turbo C 的最新版本。

(1)Turbo C 2.0 的特点

该版本在 1.5 版的基础上增加了许多功能,具有如下特点:

- 编辑、编译、调试、运行一体化
- 综合调试程序具有单步执行、单步跟踪、断点设置、表达式监视和求值等功能。
- 支持 Turbo Debugger 独立调试程序
- 具有更快的编译、链接程序(快 20~30%)
- 具有更快的内存分配函数和串函数
- EMS(扩展内存规范)用作编辑缓冲区
- 能仿真 80x87,浮点运行速度更快
- 新增加了 Signal 和 Raise 函数
- 新增加的 \_emit\_ 函数允许用户在编译时向程序插入机器代码
- 高级图形库中增加了许多新函数,包括可安装的驱动程序和字体
- 支持命令行上的匹配符 \* 和 ? 等
- 可连接生成小模式的 COM 文件,运行速度高
- 支持 Long double 常数和变量
- 能自动进行快缩进和回退及优化填充
- MAKE 实用程序可自动进行依赖关系检查
- 新增加一些实用工具,如

THELP:用于在DOS下得到与Turbo C 2.0版本内容相关的帮助系统的内存驻留程序  
CINSTXFR.EXE:用来把Turbo C 1.5版综合开发环境的配置文件转换成2.0版的对应文件  
OBJXREF.EXE:用于按指定目录顺序搜索所有目标文件和库文件

#### (2) Turbo C 2.0 的配置要求

Turbo C 2.0 有如下配置要求

- 适于 IBM PC 系列机,包括 XT、AT、PS/2 及其它兼容机
- 需要 2.0 或更高版本 DOS 支持
- 至少需要 48K RAM
- 80 列彩/单监视器
- 至少一个软盘驱动器,建议使用两个软盘或一硬盘带一软盘
- 能仿真 80x87 协处理器,若系统中有该芯片,将大大加快浮点运算速度

### 1.4 Turbo C 2.0 的内容

Turbo C 2.0 共有 6 张盘,各盘内容如下:

#### #1 盘 INSTALL/HELP(安装/帮助盘)

INSTALL.EXE 安装程序  
README.COM README 文件阅读程序  
TCHELP.TCH 帮助文件  
THELP.COM 读取 TCHelp.TCH 的驻留程序  
THELP.DOC THELP.COM 文件的文档  
README 有关 Turbo C 最新信息的文件

#### #2 盘 INTEGRATED DEVELOPMENT ENVIRONMENT(综合开发环境盘)

TC.EXE Turbo C 综合开发环境编译程序  
TCCONFIG.EXE 配置文件转换程序  
MAKE.EXE 程序管理工具  
GREP.COM 文件检索  
TOUCH.COM 时间记录器

#### #3 盘 COMMAND LINE/UTILITIES(命令行编译程序/实用工具盘)

TCC.EXE 命令行编译程序  
CPP.EXE 预处理程序  
TCINST.EXE TC.EXE 配置设置工具  
TLINK.EXE 链接程序  
HELPME!.DOC 一般问题和答案

#### #4 盘 LIBRARIES(库程序盘)

C0S.OBJ 小型模式启动程序  
C0T.OBJ 微型模式启动程序  
C0L.OBJ 大型模式启动程序  
MATHS.LIB 小型模式数学库  
MATHL.LIB 大型模式数学库  
CS.LIB 小型模式运行库

CL.LIB 大型模式运行库

EMU.LIB 8087 仿真库

GRAPHICS.LIB 图形库

FP87.LIB 8087 库

TLIB.EXE 库管理工具

#5 盘 HEADER FILES/LIBRARIES(标题文件和库程序)

????????? ·H 标题文件

<SYS> SYS\\*.H 标题文件目录

COC.OBJ 紧缩模式启动程序

COM.OBJ 中型模式启动程序

MATHC.LIB 紧缩模式数学库

MATHM.LIB 中型模式数学库

CC.LIB 紧缩模式运行库

CM.LIB 中型模式运行库

#6 盘 EXAMPLES/BGI/MICS(实例/BGI 图形库/MICS 文件盘)

UNPACK.COM 打开 .ARC 文件的工具

OBJXREF.COM 目标文件交叉引用工具

COH.OBJ 巨型模式启动代码

MATHH.LIB 巨型模式数学库

CH.LIB 巨型模式运行库

GETOPT.C 命令行选择分析器

HELLO.C 源程序例子

MATHERR.C 数学库例外情况处理源程序

SSIGNAL.C ssignal 和 gsignal 函数源程序

CINSTXFR.EXE 传送 1.5 版配置到 2.0 版的工具

INIT.OBJ 连接 Prolog 时的初始化代码

BGI.ARC BGI 驱动程序和字体

BGIOBJ.EXE 字体和驱动程序转换工具

ATT.BGI ATT400 图形卡驱动程序

CGA.BGI CGA 图形驱动程序

EGAVGA.BGI EGA 和 VGA 图形驱动程序

HERC.BGI Hercules 图形卡驱动程序

IBM8514.BGI IBM8514 图形卡驱动程序

PC3270.BGI PC3270 图形卡驱动程序

GOTH.CHR 哥特式字符集

LITT.CHR 小字符集

SANS.CHR sans serif 字符集

TRIP.CHR 立体字符集

BGIDEMO.C 图形演示程序

STARTUP.ARC 启动源程序的 ARC 文件和其它相关文件



RULES.ASI 和 Turbo C 接口的汇编 include 文件  
 C0.ASM 启动代码的汇编源程序  
 SETARGV.ASM 命令行分析的汇编源程序  
 SETENV.ASM 环境处理的汇编源程序  
 BUILD-C0.BAT 建立启动代码模块的批处理文件  
 MAIN.C 交互式 C 主文件  
 EMUVARS.ASI 仿真程序的汇编变量说明  
 WILDARGS.OBJ 匹配符参数扩充模块的目标码  
 EXAMPLES.ARC 各种例子  
 CPASDEMO.PAS Turbo Pascal 4.0 和 Turbo C 2.0 接口演示 Pascal 源程序  
 CPASDEMO.C Turbo Pascal 4.0 和 Turbo C 2.0 接口演示 C 源程序  
 CTOPAS.TC 在与 Turbo Pascal 4.0 程序链接时为产生正确格式的 Turbo C 模块所必须的 TC.EXE 配置文件  
 CBAR.C PBAR.PRO 文件中用到的函数例子  
 PBAR.PRO Turbo Prolog 和 Turbo C 接口演示 Prolog 程序  
 WORDCNT.C 源程序级调试演示例子  
 WORDCNT.DAT WORDCNT.C 中用到的数据文件

### 1.5 Turbo C 2.0 的安装

Turbo C 2.0 版有两种编译程序,即综合开发环境编译程序 TC 和命令行编译程序 TCC。其安装和启动按如下方法进行。

(1)安装前准备工作 先用 DOS 的 DISKCOPY 命令对六张盘作一备份,把源盘保存好,使用备份工作。

(2)安装方法有两种:

①使用 DOS 的 copy 命令 把必需的文件拷贝到工作软盘或硬盘的某一子目录下,如使用如下命令:

A>copy a:TC.EXE b:✓

A>copy a:TC.EXE c:\TC✓

可把 TC.EXE 文件分别拷贝到 B 盘上或 C 盘的 TC 目录下。

②利用安装程序 使用第一张盘中的 INSTALL.EXE 文件,也可安装 Turbo C 2.0 版。命令为:

A>INSTALL✓

INSTALL 有三种选择:

(1)Hard Drive 把 Turbo C 2.0 各部分安装到硬盘中各子目录下,并根据各子目录创建 Turbo C.CFG 文件。

(2)update from TC 1.5 把 Turbo C 1.5 版升级为 2.0 版,并传送当前 TC.EXE 中的配置选项。

(3)Floppy Drive 在软盘中建立 Turbo C 2.0,必须事先准备好三张格式化了了的空盘,每次运行 INSTALL 时,都让你安装一种存储模式的 Turbo C,如果想安装几种模式,就得有几套盘,每一套对应一种模式。

## 1.6 综合开发环境编译程序 TC

安装好 Turbo C 2.0 以后,在其目录下,打入 TC 或 TCC 即可分别启动 Turbo C 2.0 综合开发环境或命令行编译程序,然后,使用这两种方法之一就可以进行程序设计了。本讲先介绍 TC 的用法。

(1)TC TC 是 Turbo C 的综合开发环境编译程序。在该环境中,开发程序所必须的所有工具并非七零八落,为方便用户,而把它们综合为一体。其中除了编辑程序、编译程序、链接程序之外,还有 MAKE(重新编译、链接)、编译的状态设定、debug 等强有力的工具,构成一个功能很强,使用极为方便的系统。Borland 公司把该环境叫做 Integrated Development Environment(综合开发环境)。

### (2)TC 启动方法

装入 Turbo C 2.0 后,在 DOS 下,键入如下命令即可启动 TC。

A>TC✓

TC 启动后,出现主屏幕和版本信息,按任意键,版本信息消失,按 Shift-F10,版本信息重现。主屏幕由四部分组成,即主菜单、编辑窗口、信息窗口和功能键提示,如图 1.1 所示。

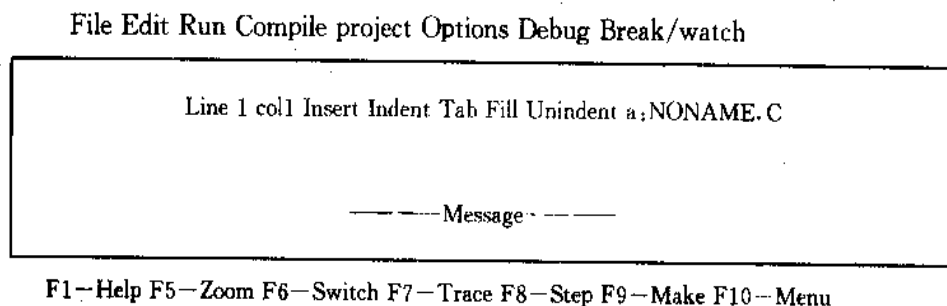


图 1.1 主屏幕

### (3)TC 命令行开关

TC 有如下命令行开关

/c——加载配置文件 其用法如:

TC /cmyconfig

该命令使 Turbo C 在当前目录下查找名为 myconfig 的配置文件。注意/c 与文件名之间无空格。

/b——重新编译 project 里的所有文件,把编译信息打印在输出设备上,返回 DOS。其用法如:

TC /cmyconfig.tc /b

或

TC /b

/m——只编译、链接过时文件

/d——在检测到合适的硬件时,使用双监视器。

### (4)TC 热键

TC 中可使用的热键如表 1.1 所示。

表 1-1 TC 热键

键	功能
F1	激活帮助窗口,提供有关当前位置的信息
F2	编辑的文件存盘
F3	加载文件(出现输入框)
F4	程序运行到光标所在行
F5	放大、缩小活动窗口
F6	开关活动窗口
F7	在调试模式下运行程序,跟踪到函数内部
F8	在调试模式下运行程序,跳过函数调用
F9	执行 Make
Ctrl-F1	调用有关函数的上下文帮助
Ctrl-F3	显示调用栈
Ctrl-F4	计算表达式
Ctrl-F7	增加监视表达式
Ctrl-F8	断点开关
Ctrl-F9	运行程序
Alt-F1	显示上次访问的帮助
Alt-F3	选择文件加载
Alt-F6	开关活动窗口里的内容
Alt-F7	定位上一错误
Alt-F8	定位下一错误
Alt-F9	把文件编译为 OBJ 文件
Alt-B	转到 Break/Watch 菜单
Alt-C	转到 Compile 菜单
Alt-D	转到 Debug 菜单
Alt-E	转到 Edit 菜单
Alt-F	转到 File 菜单
Alt-O	转到 Option 菜单
Alt-P	转到 Project 菜单
Alt-R	转到 Run 菜单
Alt-X	退出 TC,返回到 DOS

## 第二章 Turbo C 基本数据

### 2.1 基本数据类型

Turbo C 使用的基本数据类型有 5 种,如表 2.1 所示。

表 2.1 Turbo C 基本数据类型

数据类型	长度	数值范围
int(整型)	16 位	-32768~32767
float(单精度浮点型)	32 位	3.4E-38~3.4E+38
double(双精度浮点型)	64 位	1.7E-308~1.7E+308
char(字符型)	8 位	-128~127
void(无值型)	0 位	无值

### 2.2 修饰符

数据类型除 void 外,可依用途作如下修饰说明:

signed 有符号

unsigned 无符号

long 长整型

short 短整型

这四个修饰符还可按如下组合使用:

signed long

signed short

unsigned long

unsigned short

与 char、int、double 组合可得如下八种类型:

signed short int

unsigned short int

signed long int

unsigned long int

signed double

unsigned double

signed char

unsigned char

无符号字符和有符号字符的取值范围如表 2.2 所示。

表 2.2 字符数据的分类

类型	取值范围
char	-128~127
unsigned char	0~255
signed char	-128~127

### 2.3 常量

常量是指其值固定不变的量,有如下五种:

(1)字符型常量 用单引号引起来的单个字符,如'a',' '等。

(2)整型常数 可用 10、8 和 16 进制表示,用 8 和 16 进制表示时,在常数前要有前导词 0 和 0x(或 0X),如:

123...10 进制数

0123...8 进制数

0x123...16 进制数

长整型常数的后面要加上 l 或 L,如 123L。

(3)浮点型常量 它有 2 种表示方法,即:

①小数表示法 如 1.23;

②科学表示法 如 1.23e-3,e 也可用 E,代表  $1.23 \times 10^{-3}$ 。

(4)字符串常量 用双引号引起来的字符序列,如"abc"。在存储器内部为字符序列 abc\0。 \0 (Null)为结尾符。可见,字符与字符串是不同的。

(5)控制字符常量 ASCII 码值在 0x00~0x1f 之间的字符,即控制用字符,需采用反斜杠与特定字符组合表示,才能输入。用这种方法所表示的字符亦叫转义字符。Turbo C 的转义字符如表 2.3 所示。

表 2.3 转义字符表

符号	ASCII 码	功能
\0	0x00	Null
\a	0x07	响铃
\b	0x08	退格
\t	0x09	水平制表
\f	0x0c	走纸
\n	0x0a	回车换行
\v	0x0b	垂直制表
\r	0x0d	回车
\\	0x5c	反斜杠
\'	0x27	单引号
\"	0x22	双引号
\?	0x3f	问号
\DDD	0DDD	DDD 为 3 位 8 进制数
\xHHH	0xHHH	HHH 为 3 位 16 进制数

### 2.4 变量及其初始化

所谓变量是其值可变化的量,在程序中使用变量时必须首先说明其数据类型,其语句格式如下:

## 数据类型 变量表;

注意:C 语句末尾必须有一个分号(;),例如:

```
char a,b;
```

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

为使用这些变量,还必须先给出其值的大小,这叫变量初始化。例如:

```
int i=10,j=20,k;
```

```
k=i+j;
```

变量名的组成规则如下:

①以字母或下划线符( )开始

②能使用的字符有 a~z,A~Z,0~9, \_ 和 \$。

③有效长度为 32 个字符。

④大小写是有区别的,C 语言是以小写字母为主,书写的小写化是 C 程序的一大特点。

⑤不能使用 Turbo C 的关键字。

## 2.5 函数

结构的模块化,是 C 程序的又一大特点。函数是程序的基本模块,相当于子程序,定义形式如下:

函数类型说明 函数(参数表)

参数类型说明

```
{  
    说明语句  
    执行语句  
}
```

(1)函数类型 用前面所介绍的关于基本数据类型的关键字来说明,指出函数值的数据类型。

(2)函数名 标识符后紧跟一对圆括号构成。

(3)函数体 函数体的界线符是一对花括号,其内容大致为两大类:说明语句和执行语句。

函数的定义中,必不可少的部分是:

函数名()

```
{  
}
```

这是最小 C 程序。由于它的函数体没有实在内容,故什么也不执行。程序员常用它在程序中事先占据一个位置,好将来扩充一个有实际功能的函数。

## 2.6 表达式

(1)表达式的组成 表达式是用运算符把操作数连接起来所构成的式子。操作数可以是常量、变量和函数。

(2)表达式的解 指表达式中的操作数按运算符优先级进行运算,最终得到的表达式的值。

(3)表达式的副作用 在表达式中使用赋值运算符和递增(++)、递减(--)运算符时,将把值赋给某个变量,这就叫表达式的副作用(side effects)。

(4)表达式语句 表达式的后面加上一个分号(; )就构成一个表达式语句。C 程序主要是由表

达式语句构成的,语句的表达式化是 C 程序的又一大特点。

(5)简单语句和复合语句 C 语言中的分号是语句的结尾符。含一个分号的语句叫简单语句,如:

K=i+j;

多个简单语句借助一对花括号可组成复合语句,如:

```
{ i=10;  
  j=20;  
  K=i+j;  
}
```

## 2.7 程序、文件、函数三者的关系

C 程序的编译单位是文件。一个文件由一个或若干函数组成。一个可执行文件必须有一个名为 main() 的函数,叫主函数。不管该函数放在什么地方,程序总是从它开始执行,到它执行完毕,整个程序也就执行完毕。其它函数由 main() 或别的函数调用执行。

下面介绍一个简单的程序,以使读者了解 C 程序的风格,尽快掌握 C 程序的设计方法。

### [练习 2.1]

```
A>type li1.c  
main()  
{ int i=10,j=20,k;  
  k=i+j;  
  printf("k=%d\n",k);  
}  
A>li1  
K=30
```

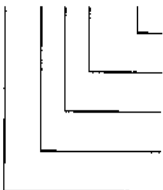
(1)该程序只含一个文件,文件名叫 li1.c;注意,C 源文件的扩展名必须是 c。该文件只含有一个函数,这就是主函数 main()。

(2)2 行为说明语句,说明 i、j、k 为整型变量;同时对 i 和 j 进行了初始化。

(3)3 行是计算 i+j 的值的执行语句。该语句就是一个表达式语句。符号“=”是赋值运算符,表示把 i+j 的值赋给变量 k,因此,该语句也叫赋值语句。

(4)4 行是 C 语言输出方式——调用输出函数。C 语言的输入输出功能是靠调用 C 编译系统的库函数实现的。输入输出的函数化是 C 程序的又一大特点。函数 printf() 是按格式输出函数,其功能是按照给定格式显示(或打印)表达式的值,其用法如图 2.1 所示。

```
printf(转换控制字符串,自变量);  
printf("k=%d\n", k);
```



- 自变量,其内容为要显示的值
- 回车换行
- 转换控制字符,表示按 10 进制显示所对应的 k 的值
- 转换控制开始符,指定要显示的值的位置
- 原样显示的部分

图 2.1 库函数 printf() 的用法

printf() 函数的转换控制字符如表 2.4 所示。

表 2.4 printf() 转换控制字符

转换控制	所指定的打印格式
%d	按十进制输出
%o	按无符号 8 进制输出
%x	按无符号 16 进制输出
%u	按无符号 10 进制输出
%c	按字符输出
%s	按字符串输出
%e	以指数形式输出
%f	以小数形式输出
%g	采用 %e 和 %f 较短的输出
%p	打印地址

在 % 与转换控制字符之间还可加上宽度说明。无宽度说明时,若输出字符,则按字符的实际个数输出;若输出浮点数则按 6 位小数处理。宽度说明的格式如图 2.2 所示。

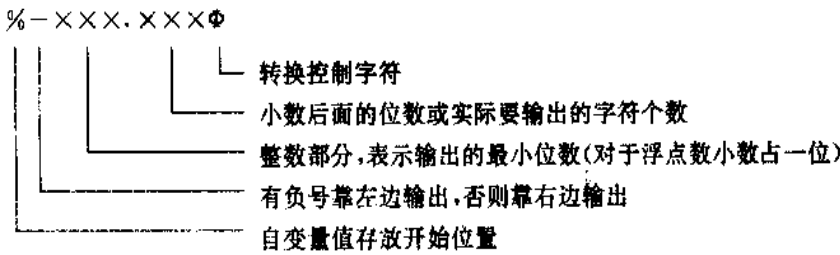


图 2.2

2.8 按格式输入函数 scanf()

(1) 功能和用法

scanf() 也是一个系统库函数,其功能是从键盘输入数据,用法是  
scanf(转换控制字符串,变量 1 指针,变量 2 指针,……);

[练习 2.2]

```
A>type li2.c
main()
{ int i,j;
  scanf("%d,%d",&i,&j);
  printf("input: %d,%d\n",i,j);
}
A>li2
246,48
input: 246,48
```

该练习中 scanf() 各参数的名称和功能如图 2.3 所示。



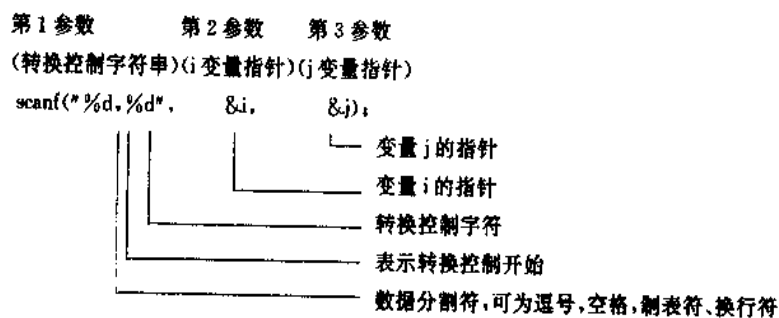


图 2.3 库函数 scanf( ) 的用法

## (2) 转换控制字符

- d——把输入看作是 10 进制整型数;
- f——把输入看作是浮点数;
- e——同 f;
- o——把输入看作是 8 进制数;
- x——把输入看作是 16 进制数;
- c——把输入看作是一个字符;
- s——把输入看作是字符串;
- h——把输入看作是短整型数;
- i——把输入看作是 10 进制整型数;
- p——把输入看作是指针;
- n——把输入视为整型数, 其值为读入字符个数。

## (3) scanf( ) 函数的一些具体用法

- ① 转换控制字符前的数字, 用于指定数据位数;
- ② 转换控制字符前面的 L, 用来表示读双精度浮点数或长整型数据;
- ③ 转换控制开始符 % 后面的 \* 用来禁止赋值;
- ④ 如若转换控制字符串中有非转换控制字符, 则输入数据时要在相应的部分输入与此相同的字符;

## 2.9 函数调用形式

(1) C 程序结构 C 程序总的看, 是由函数定义组成的。注意, 函数不能嵌套定义, 但能嵌套调用。

(2) 函数的调用关系 分为内部调用和外部调用。内部调用指一个文件内函数间的调用。外部调用指一个文件的函数调用另一个文件的函数。

(3) 函数调用形式 最基本的是如下三种形式:

① 函数语句调用 把被调用函数的函数名直接写出, 并以实参替换形参, 圆括号后加以分号。被调用函数将作为一个独立的语句出现, 如:

### 〔练习 2.3〕

A>type li3.c

```
main()
{ printf("I'm in main.\n");
  aia();
}

aia()
{ printf("Now I'm in aia.\n");}
```

A>li3

I'm in main.

Now I'm in aia.

②函数表达式调用 这种调用的特点是被调用函数出现在调用函数的表达式中,被调用函数要有返回值。在调用函数中,要说明被调用的数据类型,若其数据为整型,则无须进行类型说明。

### 〔练习 2.4〕

A>type li4.c

```
main()
{ float i=1.5;
  double square();
  i=square(i);
  printf("%f\n",i)
}
```

double square(x)

```
double x;
{ return(x * x);}
```

A>li4

2.250000

③函数参数调用 被调用函数以函数参数形式出现,其调用条件和注意事项同函数表达式调用。

### 〔练习 2.5〕

A>type li5.c

int i=2,j=5;

main()

```
{printf("%d,%d\n",sum(),sub());}
```

sum()

```
{ int s;
  s=i+j;
  return(s);
}
```

sub()

```
{ int e;
  e=i-j;
  return(e);
}
```

A>li5

7,-3

## 2.10 数据的存储类别

在C语言中,变量除了有类型之分,还有四种存储类别,即:自动存储变量、静态存储变量、外部存储变量和寄存器存储变量。这些存储类别用来说明诸如变量的作用域、变量的生存期等性质。变量的作用域是指能访问该变量的场所。变量的生存期是指变量值所能保留的时间。

(1)自动存储变量 是指其值在某一范围内得以保留,这一范围以外则消失的变量。

①性质 自动存储变量有如下性质:

- 作用域的局部性 其作用域为变量定义所在的模块(一对花括号围起来的部分)内或函数内。

- 生存期的暂时性 其生存期为该变量定义所在函数或模块的执行周期,即一旦进入该函数或模块,C就自动地为该变量建立存储区,而一旦退出该函数或模块,C就自动地收回此存储区。

- 未初始化的变量其值不确定,是无意义的。

②定义 定义自动存储变量用关键字 auto,如:

```
auto int i,j;
```

### 〔练习 2.6〕

A>type li6.c

```
main( )
```

```
{ int i=100,k=80;
```

```
printf("i=%d\n",i);
```

```
{ int i=200;
```

```
printf("i=%d k=%d\n",i,k++);
```

```
}
```

```
printf("i=%d k=%d\n",i,k);
```

```
}
```

A>li6

i=100

i=200 k=80

i=100 k=81

(2)静态存储变量

①定义 可按如下形式加以定义及初始化:

```
static 数据类型 变量名=初值;
```

②种类 有局部和全局两种静态存储变量:

- 静态局部存储变量 仅能在所定义的模块内使用。与 auto 变量不同,它可保留原值不变,如函数:

```
count_up()
```

```
{ static int number=0;
```

```
number+=25;
```

```
return(number);
```

```
}
```

其返回值为 25,以后每调用一次,其值就增加 25。这是由于在编译时就给静态存储变量 number 分配了存储空间,而以后该变量就永久地存在下去的缘故。

但变量 number 不能在它所在的函数外进行访问,说明以这种方式定义的静态存储变量其作用域是局部性的。因此这种变量叫做内部静态存储变量。

• 静态全局存储变量 如果把变量定义在模块外部,则所有函数都可以使用这个变量,例如:

```
static int number;
count_up()
{ number += 25;
  return(number);
}
reset()
{ number = 0;
}
```

这样定义的存储变量叫做外部静态存储变量。

③性质 静态存储变量有如下性质:

• 作用域 内部静态存储变量的作用域为函数或模块内;外部静态存储变量的作用域为整个程序。

• 生存期的永久性。

• 未初始化的静态存储变量的值为 0。

下面说明静态存储变量和自动存储变量的区别。

#### (练习 2.7)

A>type li7.c

```
main()
{ count();
  count();
  printf("\n");
  add();
  add();
}
count()
{ static int num0=0;
  num0 += 25;
  printf("%d\t",num0);
}
add()
{ auto int num1=0;
  num1 += 25;
  printf("%d\t",num1);
}
```

A>li7

25 50

25 25

### (3) 外部存储变量

①用途 用于把大程序分割为若干程序单元(文件)而开发的场合。说明外部存储变量使用关键字 `extern`。例如把一个程序分为两个单元来编译:

```
A>type file. c
int i,j;
float x; /* 定义外部存储变量 i,j 和 x */
main()
{
    :
func1(); /* 函数调用 */
func2(); /* 函数调用 */
}
```

```
A>type file2. c
extern int i,j;
extern float x; /* 对外存储变量的说明 */
func1() /* 函数定义 */
{
    :
}
func2() /* 函数定义 */
{
    :
}
```

这说明,若在文件 `file2` 中使用外部存储变量,则须在该文件中进行外部变量说明。

②性质 外部存储变量有如下性质:

- 作用域的全局性 外部存储变量是全局变量。
- 生存期的永久性。
- 未初始化的外部存储变量的值为 0。

### (4) 寄存器存储变量

用关键字 `register` 定义,如:

```
register int e ;
```

寄存器变量值存储在 CPU 中。设定这种变量是为了提高操作速度。它们常用于循环控制。Turbo C 允许同时定义两个寄存器存储变量,使用寄存器 `SI` 和 `DI`。如果所定义的寄存器存储变量多于两个,则多余的按自动存储变量对待。

## 第三章 命令行 Turbo C 和实用程序

本章介绍用 DOS 命令行可执行的命令行 Turbo C 和实用程序 CPP、MAKE、TOUCH、GREP 的用法。

### 3.1 命令行 Turbo C

命令行 Turbo C 包括如下程序：TCC——C 编译程序；TLINK——链接程序；TLIB——库管理程序。

命令行 Turbo C 还具有直接插入汇编指令的功能和汇编源程序输出功能（TC 没有这些功能）。

#### (1) TCC——命令行编译程序

TCC 比 TC 有更多的任选项，如表 3.1 所示。其用法是：

TCC 任选项 要编译的文件名

表 3.1 TCC 任选项

-I	88/86/286 指令集	-gN	N 次警告后停止
-A	禁止非 ANSI 扩展	-iN	最大标识符长度 N
-B	通过汇编程序编译	-jN	N 次错误后停止
-C	允许注释嵌套	-k	标准堆栈结构
-Dxxx	定义宏	-lX	传 X 选项到链接程序
-Exxx	选择汇编程序名 *	-mc	紧缩模式
-G	按时间优化	-mh	巨型模式
-Ixxx	包含文件目录	-ml	大型模式
-K	缺省字符无符号型	-mm	中型模式
-L	库文件目录	-ms	小型模式 *
-O	优化转移指令	-mt	微型模式
-S	产生汇编输出	-nxxx	输出文件目录
-M	产生链接映射	-oxxx	目标文件名
-N	堆栈溢出检查	-p	Pascal 调用
-Uxxx	未定义宏	-r	寄存器变量 *
-Z	寄存器优化	-u	外部名下打下线符 *
-a	产生字对齐	-v	源级调试 *
-c	只编译，不链接	-w	允许所有警告
-d	合并重复字符串	-wxxx	允许 XXX 警告
-exxx	执行文件名	-w-xxx	不允许 XXX 警告
-f	浮点仿真 *	-y	产生行号信息
-f87	8087 浮点运算	-zxxx	设置段名

几点说明：

- ①打 \* 者为缺省项;  
 ②打 · 者为 2.0 版新增任选项;  
 ③-×-表示任选项×之否定;  
 ④-WXXX 任选项如表 3.2 所示。

表 3.2 -WXXX 任选项

(不符合 ANSI 标准)	
-wdup	重定义的标识符不一致
-wret	同时使用了 return 与 return of value
-wstr	标识符不是结构部分
-wstu	标识符是未定义的结构
-wsus	可疑的指针变换
-wvui	void 函数不能返回值
-wzat	长度为零的结构
(常见错误)	
-wauz	赋予标识符一个未用过的值
-wdef	标识符已定义过
-weff	无效代码
-wpar	标识符为从未用过的参数
-wpia	象是不正确的赋值
-wrch	不能实现的编码
-wrvi	函数应返回值
(不常见错误)	
-wamb	二义性运算符(要加括号)
-wamp	函数或数组有多余的 & 运算符
-wnod	函数未定义
-wpro	无函数原型
-wstv	用结构传值
-wuse	标识符未使用
(移植性错误)	
-wapt	无互换性指针赋值
-wcln	常数太长
-wept	无互换性指针比较
-wdgn	比较的常数超出范围
-wrpt	返回值转换中无互换性
-wsig	类型变换中丢失位数
-wucp	signed 与 unsigned char 型指针相混合

### (练习 3.1) DOS 下使用 TCC 的实例

```
C>tcc -IA:\include -IA:\lib -etest li.c
```

```
Turbo C Version 2.0 Copyright(c).....
```

```
li.c;
```

```
Turbo Link Version 2.0 Copyright (c).....
```

```
Available memory 408688
```

```
C>type li.c
```

```
main()
```

```
{printf("abcd.");}
```

## (2)TLINK——链接程序

TLINK 是不受操作系统控制的。它把目标模块、库模块等链接成可以执行的文件。用法如下：

Link 目标文件,执行文件,映象文件,库文件

其任选项如表 3.3 所示。

表 3.3 TLINK 任选项

/c	在公共的和外部的符号中大小写有意义
/d	发现库中符号重复的警告
/i	初始化全部段
/l	插入源文件行号
/m	映象文件含公共符号
/n	无缺省库
/s	详尽的段映象
/x	无映象文件

关于 TLINK 的用法,请注意以下几点:

①被链接文件的扩展名可缺省。

②执行文件和映象文件可缺省,但其后的逗号不能缺省;执行文件缺省表示与目标文件同名;映象文件缺省表示与执行文件同名。

③任选项可放在命令行任何位置,如:不想产生映象文件,且把源程序行号放入可执行文件时,可用如下命令:

```
tlink /x/l c0s myfile,.,cs
```

④多个目标文件用空格隔开;第一个目标文件必须是与编译该程序时所用存储模式一致的初始化模块名。初始化模块名与相应存储模式类型见表 3.4。

⑤库表用空格分开。标准库和数学库也须与编译程序所用存储模式一致,其对应关系见表 3.4。

如果程序使用浮点数,须在命令行写入 EMU.Lib(无 8087/80287,使用浮点仿真库),或 FP87.Lib(有 8087/80287,使用 80x87 浮点库)。

表 3.4 存储模式及其相应的文件

存储模式	初始化模块	标准库	数学库
微型	C0T.obj	CT.lib	MATHt.lib
小型	C0S.obj	CS.lib	MATHs.lib
紧凑型	C0C.obj	CC.lib	MATHc.lib
中型	C0M.obj	CM.lib	MATHm.lib
大型	C0L.obj	CL.lib	MATHl.lib
巨型	C0H.obj	CH.lib	MATHh.lib

例如:使用小型存储模式,浮点仿真,链接名为 li 的目标文件,其 TLIN 命令行为:

```
tlink c0s li,.,emu cs maths
```

## (3)TLIB——库管理程序

①功能 为建立目标形式的函数库提供方便。它有增加模块到库、从库中消除模块和从库中抽出一个.obj 文件三种功能。其命令的一般形式是:



TLIB 库名(OP) 模块名(OP) 模块名...

(所指定的库) (操作符)(操作模块)

②注意事项:

- 库扩展名为 .Lib, 操作模块扩展名为 .obj.
- 允许使用的操作符如表 3.5 所示。

表 3.5 TLIB 的操作符

操作符	功能
+	增加模块到指定库
-	从指定库中消去模块
*	从指定库中抽取 .obj 文件
-+ 或 +-	消去同时又增加(即重新拷贝)
-* 或 *-	抽取同时又消去

- TLIB 只能对整个模块进行操作

下面通过实例综合练习 Tcc、Tlib、Tlink 三个程序的用法。练习 3.2 中给出的序号为操作顺序。

〔练习 3.2〕

(1) 编辑求  $x^y$  的文件 sqr.c

```
C>type sqr.c
long sqr(x,y)
int x,y;
{ int i;
  long j=1;
  for(i=0;i<y;i++)
    j=(long)j*x;
  return(j);}
```

(2) 用 Tcc 编译该文件

```
C>tcc -c sqr.c
Turbo C Version 2.0 Copyright(c)...
```

sqr.c:

Available memory 419722

(3) 用 dir 查看编译结果

```
C>dir sqr.*
SQR  OBJ    260   1-01-80   12:05a
SQR  C      104   1-01-80   12:33a

2 File(s) 13703168 bytes free
```

(4) 用 Tlib 把 sqr.lib 加入到 maths 库

```
C>tlib c:\turbo\lib\maths+sqr
TLIB Version 2.0 Copyright(c)...
```

(5) 编辑调用函数 sqr() 的文件 test.c

```
C>type test.c
#include <stdio.h>
#include "sqr.c"
```

```
main()
{ int x,y;
  printf("Please Input:\n");
  scanf("%d,%d\n",&x,&y);
  printf("%d^%d=%ld\n",x,y,sqr(x,y));
}
```

#### (6)编译文件 test. c

```
C>tcc -c test. c
```

Turbo C Version 2.0 Copyright(c)...

test. c;

Available memory 85560

#### (7)查看编译结果

```
C>dir test. *
```

```
TEST  C      162    1-01-80    12:14a
TEST  OBJ     459    1-01-80    12:17a
```

2 File(s) 13711360 bytes free

#### (8)用 Tlink 对 test. obj 文件进行链接

```
C>tlink c:\turbo\lib\c0s test,test, c:\turbo\lib\emu c;\turbo\lib\maths c;\turbo\lib\cs
```

Turbo Link Version 2.0 Copyright(c)...

#### (9)查看链接结果

```
C>dir test. *
```

```
TEST  C      162    1-01-80    12:14a
TEST  OBJ     459    1-01-80    12:17a
TEST  MAP     514    1-01-80    12:30a
TEST  EXE    9544    1-01-80    12:30a
```

4 File(s) 13709312 bytes free

#### (10)执行文件

```
C>test
```

Please Input:

2,4

2,4

2^4=16

对于源文件 test. c,若使用如下任选项编译,则可直接得到执行文件 test. exe

```
C>tcc -Ic:\turbo\include -Ic:\turbo\lib -ea;test a;test. c
```

Turbo C Version 2.0 Copyright(c)...

a;test. c;

Turbo Link Version 2.0 Copyright(c)...

Available memory 404038

```
C>dir a;test. *
```

```
TEST  OBJ     562    1-01-80    1:08a
TEST  C      162    1-01-80    12:14a
TEST  EXE    9544    1-01-80    12:04a
```

3 File(s) 71680 bytes free

## 3.2 预处理程序 CPP

CPP 也是不受操作系统控制的应用程序,其功能是对源程序预处理。它最初由编译程序启动。使用 TC 时,我们完全意识不到它。其用法如下:

CPP 文件名

源文件经 CPP 预处理后,其中的宏指令均被预处理而变为扩展名为.i 的文件,如练习 3.3。

### 〔练习 3.3〕

(1)编辑含有宏指令的程序

```
C>type exp3_3.c
#define MAX(x,y) (x>y)? x:y
main()
{   int a=5, b=3;
    printf("MAX=%d\n",MAX(a,b));
}
```

(2)用 CPP 预处理

```
C>CPP exp3_3.c
CPP version 2.0 copyright(c) ...
exp3_3.c
Available memory 525430
```

(3)用 type 命令查看预处理后的文件

```
C>type exp3_3.i
exp3_3.c 1:
exp3_3.c 2:main()
exp3_3.c 3:{   int a=5,b=3;
exp3_3.c 4:    printf("MAX=%d\n", (a>b)? a:b);
exp3_3.c 5;}
exp3_3.c 6:
```

CPP 任选项如表 3.6 所示。

表 3.6 CPP 任选项

-A	禁止非 ANSI 扩展	-ml	大型模式
-C	允许注释嵌套	-mm	中型模式
-Dxxx	定义宏	-ms	小型模式
-Ixxx	Include 文件目录	-mt	微型模式
-P	包含源文件信息	--nxxx	输出文件目录
-Uxxx	未定义宏	--oxxx	目标文件名
-gN	N 次警告后停止	-p	Pascal 调用
-iN	最大标识符长度	-w	允许所有警告
-jN	N 次错误后停止	-wxxx	允许×××警告
-mc	紧凑模式	-w-xxx	不允许×××警告
-mh	巨型模式		

## 3.3 程序管理工具 MAKE

MAKE 也是不受操作系统控制的软件工具。

(1)功能 自动重新编译修改过的文件,避免对所有文件重新编译而造成时间浪费。

(2)MAKE 的启动 启动 MAKE 的命令行是:

A>MAKE

该命令将编译所需的模块,建立可执行文件。当不指定其他文件且 MAKE 存在时,就执行其内容。要想使用其它文件(如 MYMAKE),就要用任选项-f,如:

A>MAKE -fMYMAKE

(3)任选项 MAKE 命令行的任选项如表 3.7 所示。

表 3.7 MAKE 任选项

任选项	含义
-Didentifier	定义标识符 identifier
-Diden=string	把标识符定义为字符串(不能含空格与制表符)
-Idirectory	在 directory 目录内检索包含
-identifier	解除前面所定义的标识符
-s	无该选项,MAKE 显示全部命令;否则不显示
-n	显示命令但不执行,用于观察 make 文件
-ffilename	把 filename 作为 MAKE 文件使用。
?, -h	显示帮助信息。

#### 〔练习 3.4〕

C>make -s -fsample.mak

MAKE Version 2.0 Copyright(c)...

Available memory 247615 bytes

(4)MAKE 文件 由目标文件、源文件和命令表组成。目标文件由它所依存的源文件经编译产生。MAKE 文件的一般形式如下:

目标文件 1:源文件 1

命令序列

目标文件 2:源文件 2

命令序列

⋮

说明如下:

①目标文件须从最左列写起,其后紧跟冒号。

②命令形式为:若干空格〔词头〕命令

目标文件的词头有三个,即@、-num 和-。

@:表示执行命令时,不显示该命令。

-num:若指定数值 num,则当退出状态大于 num 时,MAKE 停止执行;若未指定,则当退出值不为零时,MAKE 停止执行,且删除目前的目标文件。

-:不检验退出状态,因此,继续执行 MAKE。

命令能够使用 MS-DOS 的全部命令,但改道命令和滚边命令例外。命令序列中的命令、任选项、文件名等用空格或制表符隔开。

③注释以#开关,可跟在源文件表或命令序列之后;若注释单独占一行,就必须从最左列开始。

④目标文件定义与下一个定义至少间隔一空行。

(5)MAKE 文件组成要素 由注释、法则、隐含法则、宏定义、指令等五个基本要素组成。

①注释 是为便于阅读 MAKE 文件而设置的以 # 为开头的说明,并不被 MAKE 执行。

②法则 用来描述文件间的依存关系与命令,其形式如下:

目标文件 [目标文件...];[源文件 源文件...]

[命令]

[命令]

例如:c.obj;c.c d.h

tcc -c -mm -f c.c

其中 c.obj 为目标文件,c.c 为源文件,d.h 为源文件,tcc -c -mm -f c.c 为命令。

法则执行过程如下:

- 若目标文件不存在则生成目标文件。

- 若目标文件存在,则检查其生成日期与时间,并与对应的每个源文件的生成日期与时间比较。若目标文件较新,则不执行命令;而当源文件较新时,则执行命令,制作相应的新目标文件。

③隐含法则 请看下面这些法则:

a.obj;a.c

tcc -c -mm -f a.c

c.obj;c.c

tcc -c -mm -f c.c

它们的形式完全相同,目标文件名与对应的源文件也分别相同,可以用如下一条隐含法则来表示。

-c.obj;

tcc -c -mm -f \$<

这条隐含法则表示,所有 .obj 文件皆依存于对应的 .c 文件,且通过 tcc 命令实现由 .c 到 .obj 的转变。这种隐含法则也适用于不用法则指定命令的情况。

④宏定义 MAKE 文件用宏定义的形式是:

宏名=宏展开字符

宏名用不含空格的英文字母、数字序列指定。宏展开字符可用英文字母、数字、空格等构成,末尾是回车换行符。所定义的宏名可按 \$(宏名)形式用在 MAKE 文件中。MAKE 将把 \$(宏名)置换为宏展开字符。如果宏名未定义,将置换为 Null 字符。例如下面的法则,第一行为宏定义。第 3 行为该宏定义的使用,MAKE 将把 \$(MCR)置换为 S。

MCR=s

a.obj;a.c

tcc -c -m \$(MCR)-f a.c

要想改变存储模式,只要改变宏定义即可。

⑤控制指令 MAKE 的控制指令(directives)有:

- 包含指令(! include) 该指令的功能是把 MAKE 文件所指定的文件包含进来,其形式为:

! include "文件名"

有了这条指令就可把不同程序共用的部分放入一个专门文件,这样不管哪个 MAKE 文件都能使用。

- 条件指令(! if,! elif,! else,! endif)

(用法 1) ! if 表达式 (若表达式成立,则执行语句 1、语句 2...)

```

        语句 1
        语句 2
    ! endif
〔用法 2〕 ! if      表达式    (若表达式成立,则执行语句 1;否则执行语句 2)
        语句 1
        ! else
        语句 2
    ! endif
〔用法 3〕 ! if      表达式 1  (表达式 1 为真,则执行语句 1;)
        语句 1
        ! elif      表达式 2  (表达式 1 为假,表达式 2 为真,则执行语句 2;)
        语句 2
    ! endif      (若表达式 1、表达式 2 皆假,则什么也不执行)

```

语句 1、语句 2 包括宏定义、法则、隐含法则、include 指令等。! if 和 ! elif 的表达式是 32 位带符号的整型表达式,由操作数、运算符构成。操作数可用 10、8、16 进制表示。可用运算符如下:

单项运算符有 -、~、!;

双项运算符有 +、-、\*、/、%、>>、<<、&、|、^、&&、||、>、<、>=、<=、==、!=;

三项运算符有?:。

• 错误指令(! error) 功能是使 MAKE 停止执行,显示文本。一般与条件指令组合使用,如:

```

! if ! $d(MCR)
! error MACRO not defined
! endif

```

• 解除指令(! undef) 解除以宏名定义的宏,用法:

```
! undef 宏名
```

⑥ 包含宏定义 MAKE 的包含宏定义用法有:

• \$d(宏名) 宏名定义过,\$d(宏名)返回 1;否则返回 0。该宏可与条件指令组合使用,如:

```

! if ! $d(MCR)
MCR=s
! endif

```

表示若 MCR 未定义,则把 MCR 定义为 s。

• \$\*(基文件名宏) 它可以展开为不带扩展名的目标名。如:

```

OBJ=a.obj b.obj c.obj
target.exe: $(OBJ)
tcc c0 $(MCR) a b c, $*, $*, \emu
math $(MCR) c $(MCR)

```

其中,\$\* 用不带扩展名的 target.exe 替代即可。该例中的 \$(OBJ) 用宏所定义的 OBJ 的目标文件列表替代即可。此外,该宏也适用于隐含规则,如:

```

.c.obj
tcc $*

```

• \$<(全文件名宏) 展开成含扩展名的目标名,例如

```
target.obj;target.c
```

```
copy $<A:\usrobj
```

```
tcc -c $ *
```

可展开成:target.obj;target.c

```
copy target.obj A:\usrobj
```

```
tcc -c target
```

- \$:(文件路径名宏) 可展开成路径名。当文件名为 A;a\b\c.c 时,可展开成 A;a\b\。
- \$. (带扩展名的文件名宏) 可展开成带扩展名的文件名。如用 \$., 可把 \a\b\c.c 展开成 c.c。
- \$&.(文件名宏) 该宏可展开成不带扩展名的文件名。例如用 \$& 可把 A;\a\b\c.c 展开成 c。

### 3.4 Touch

用于修改一个或多个文件的时间和日期为当前时间和日期,使其比依赖于它们的文件更新。用法如下:

Touch 文件名表

文件名可用通配符 \* 或?。对于执行 Touch 的文件名,执行 MAKE 时,是要重新构造的。

### 3.5 Grep

Grep 是功能很强的查找实用程序,用法如下:

Grep 任选项 查找对象 指定文件

Grep 任选项如表 3.8 所示。

表 3.8 Grep 的任选项

-c	只输出匹配行数目	-o	UNIX 输出格式
-d	查找子目录	-r	规则表达式查找
-i	忽略大小写区别	-u	更新缺省项
-l	列出匹配文件	-v	打印不匹配的行
-n	行前有标号	-w	字查找
		-z	输出查找的文件名

#### 〔练习 3.5〕

```
C>dir *.c
```

```
LI4 C 102 1-01-80 12:03a
```

```
LI5 C 46 1-01-80 12:29a
```

```
A C 38 1-01-80 12:35a
```

```
3 File(s) 13687072 bytes free
```

```
C>grep printf *.c
```

```
File LI4.C;
```

```
printf("Tianjin Shida Computer");
```

```
printf("Turbo C 2.0 exercise");
```

```
File A.C;
```

```
printf("Now is a.c");
```

## 第四章 C 运算符

运算符较多,是C语言的一个显著特点。那么,C运算符究竟有多少,又如何使用,请看本章。

### 4.1 算术、逻辑运算符

(1)算术运算符 标准算术运算符见表 4.1。

表 4.1 算术运算符

运算符	操 作	项 数	优先级
-	负	单项	1
*	乘	双项	2
/	除	双项	2
%	余	双项	2
+	加	双项	3
-	减	双项	3

#### (练习 4.1)

A>type exp4\_1.c

main()

{ int=2,j=3;

printf("add=%d\n",i+j);

printf("mod=%d\n",(i+j)%3);

}

A>exp4\_1

add=5

mod=2

该程序在 $(i+j)\%3$ 中使用了运算符 $()$ ,它是优先级别最高的运算符,常用它改变表达式的运算顺序。

(2)关系运算符 是用来判断两个值大小的运算符,共有六种,如表 4.2 所示。

表 4.2 关系运算符

运算符	操 作	项 数	优先级
>	大于	双项	1
>=	大于或等于	双项	1
<	小于	双项	1
<=	小于或等于	双项	1
==	等于	双项	2
!=	不等于	双项	2



这些运算符的用法是：

操作数 1      关系运算符      操作数 2

该式成立时,运算结果为 1(代表真);否则为 0(代表假)。注意,这里的 1 或 0 不是布尔量,是整型常数 1 或 0。C 语言里没有定义布尔量。

#### [练习 4.2]

```
C>type exp4_2.c
main()
{ int i=5,j=3,a=3 ,b=5;
  printf("< <= > >= == != \n")
  printf("%d %d %d %d %d %d\n",i<j,i<=j,i>j,i>=j,i==j,i!=j);
  printf("%d %d %d %d %d %d",
    5<3,5<=3,5>3,5>=3,5==3,5!=3);
  printf("%d %d %d %d %d %d\n",
    a<b,a<=b,a>b,a>=b,a==b,a!=b);
}
C>exp4_2
< <= > >= == !=
0 0 1 1 0 1
0 0 1 1 0 1
1 1 0 0 0 1
```

(3)逻辑运算符 基本的逻辑运算符有逻辑与(AND)、逻辑或(OR)和逻辑非(NOT)。在 C 语言中其运算符号如表 4.3 所示。

表 4.3 逻辑运算符

运算符	操 作	项 数	优先级
&&	逻辑与	双项	2
	逻辑或	双项	3
!	逻辑非	单项	1

C 语言无布尔量,故逻辑运算结果也用整型数 1 代表真,整型数 0 代表假,而对于操作数来说,若其值不是 0,为真;否则为假。

#### [练习 4.3]

```
A>type exp4_3.c
main()
{ printf("%d\n",2&&1);
  printf("%d\n",2||3);
  printf("%d\n",! 2);
}
A>exp4_3
1
1
0
```

(4)位逻辑运算符 有六种,如表 4.4 所示。

表 4.4 位逻辑运算符

运算符	名 称	操 作	项 数	优先级
&	位积	按位与	双项	3
	位和	按位或	双项	5
^	位差	按位异或	双项	4
~	位非	按位取反	双项	1
<<	左位移	按右值位数左移左值	双项	2
>>	右位移	按右值位数右移左值	双项	2

## 〔练习 4.4〕

C&gt;type exp4-4.c

main()

{ int x=7,y=5;

printf("&amp; | ^ ~ &lt;&lt; &gt;&gt;\n");

printf("%x %x %x %x %x %x",x&amp;y,x|y,x^y,~x,x&lt;&lt;y,x&gt;&gt;y);

}

c&gt;exp4-4

&amp; | ^ ~ ( )

5 7 2 fff8 e0 0

## 4.2 递增、递减、赋值运算符

这三种运算符用在表达式中,使其具有副作用。

(1)递增与递减运算符 这两种运算符是C语言特有的,它们都是单项运算符,用法如下:

$$x++ \quad \text{等价于} \quad x=x+1$$

$$x-- \quad \text{等价于} \quad x=x-1$$

这两种运算符可放在操作数的前面,叫前置运算;也可放在操作数之后,叫后置运算。

对于前置运算,例如:

$$x=10; y=++x;$$

运算结果为:

$$x=11; y=11。$$

对于后置运算,例如

$$x=10; y=x++;$$

运算结果为:

$$x=11; y=10。$$

## 〔练习 4.5〕

C&gt;type exp4-5.c

main()

{ int i=10,j;

printf("i\tj\n");

j=i;

printf("----\t----\n");

```
printf("%d\t%d\n",i++,++j);
j=i++;
printf("%d\t%d\n",i++,j++);
}
```

C>exp4\_5

```
i      j
-----
10     11
12     11
```

(2)赋值运算符 我们已经知道,C 语言的赋值运算符是"=",其用法是:

变量=表达式;

此外,在 C 语言中,假定 a 为变量,e 为表达式,op 为运算符,则表达式:

a=a op e;

可用如下表达式替代。

a op=e;

其中 op=叫做复合赋值运算符。op 可以是+、-、\*、/、%、&、|、^、<<、>>,共 10 个。这样,赋值运算符又扩展了 10 个。

#### [练习 4.6]

C>type exp4\_6.c

```
main()
{ int a=100,b=15;
  a+=b;
  a-=b;
  a*=b;
  a/=b;
  a%=b;
  a<<=4;
  printf("shift left %d\n",a);
  a>>=4;
  a&=b;
  a|=b;
  a^=0xffff;
  printf("exclusive or %d\n",a);
}
```

C>exp4\_6

```
shift left 160
exclusive or -16
```

### 4.3 求字节数、条件运算、逗号

(1)sizeof 运算符 其功能是计算变量或类型的大小,结果为字节数,其用法如练习 4.7 所示。

#### [练习 4.7]

C>type exp4\_7.c

```
main()
```

```

{ int a;
  int b[100];
  printf("1:size of int variable=%d\n",sizeof(a));
  printf("2:size of int=%d\n",sizeof(int));
  printf("3:size of array of int=%d\n",sizeof(b));
  printf("4:size of float=%d\n",sizeof(float));
  printf("5:size of double=%d\n",sizeof(double));
}

```

C>exp4\_7

```

1:size of int variable=2
2:size of int=2
3:size of array of int=200
4:size of float=4
5:size of double=8

```

(2)条件运算符 C语言特有,也是C语言中唯一的三项运算符,功能相当于一段程序,用法如下:

表达式1? 表达式2:表达式3

其含义是:

若表达式1为真,则整个表达式的解就是表达式2的解,否则,为表达式3的解。

[练习4.8]

C>type exp4\_8.c

```

main()
{ int a=5,b=3;
  printf("(a>b)? a,b=%d\n",(a>b)? a:b);
}

```

C>exp4\_8

(a>b)? a,b=5

(3)逗号运算符 该运算符用在括号中有多个表达式的场合。其功能是:从左到右求解各个表达式,而整个表达式的解为最后所求解的表达式的值。例如:下面逗号表达式的解为x的值,即等于4。

x=3,x=4;

逗号运算符有效地用于 while 语句和 for 语句中。

[练习4.9]

C>type exp4\_9.c

```

main()
{ int x,y;
  printf("%d\n",(x=3,y=4));
}

```

C>exp4\_9

4

## 4.4 指针的概念及其运算符

(1)指针的概念 指针也是一种变量,是用来指定另一个变量(允许是结构变量或另一个指针

变量)所在场所的变量,其内容为所指定变量的地址,指针变量的定义格式如下所示:

数据类型 \* 指针变量名;

例如:

char \* p; p 为指向字符型数据的指针。

int \* var; var 为指向整型数据的指针。指针变量 p 和 var 的内容及其功能如图 4.1 所示。

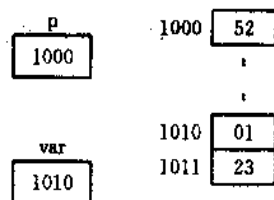


图 4.1 指针的概念

(2)指针有关的运算符 与指针有关的运算符有 & 与 \* 两种。& 用来求变量的地址,叫取地址运算符。\* 用来访问指针所指变量,叫取内容运算符。例如:

int i=10; 定义变量 i 且赋初值 10;

int \* p; 定义 p 为指针变量;

p=&i; 给 p 赋上变量 i 的地址,这时,就可以说指针 p 指向了变量 i。

\* p=21; 给 p 所指的变量内赋上数据 21。

说明上述分析正确性的程序,见练习 4.10

〔练习 4.10〕

```
C>type exp4_10.c
```

```
main()
```

```
{ int i=10,j;
  int * p;
  p=&i,j=* p;* p=21;
  printf("address of p j i * &i\n");
  printf("%p %d %d %d\n",p,j,i,* &i);
}
```

```
C>exp4_10
```

```
address of p j i * &i
```

```
ffda 10 21 21
```

#### 4.5 类型显式转换运算符

C 语言中的数据类型转换分为隐式(自动进行的)和显式(强制性的)两种。

(1)隐式转换 C 编译程序对表达式中出现的不同数据类型,按如下规则自动进行类型转换。

①表达式中的所有 float 类型数据,在任何操作之前均被转换为 double 类型。只有在数组和结构中 float 类型才真正按 float 型处理。

②在算术表达式里,所有 short 和 char 型数据都转换为 int 型。

③参加运算的两个操作数类型不同时,要按图 4.2 所示把优先级低的类型转换为优先级高的

类型。

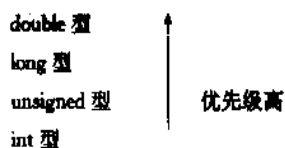


图 4.2 数据类型的优先级别

④在赋值表达式中,要把右边表达式的类型转换成左边表达式的类型,如练习 4.11 所示。

〔练习 4.11〕

C>type exp4-11.c

```
main()
{
    int i,j;
    char c;
    float f;
    c=50;
    printf("c=%c\t c=%d\n",c,c);
    i=2.6,f=2;
    printf("i=%d\t f=%f\n",i,f);
    i='a';j='a'+1;
    printf("i=%d\t j=%d\n",i,j);
    printf("c=%d\t c=%c\n",c-'0',c);
}
```

C>exp4-11

```
c=2      c=50
i=2      f=2.000000
i=97     j=98
c=2      c=2
```

(2)显式转换 一个表达式 exp 可用显式转换表达式

(type)exp

将其值转换为圆括号中 type 所指定的数据类型,如:

```
int i=10;
double f;
f=sqrt((double)i);
```

其中圆括号及其内的类型名即类型显式转换运算符。

读进 double 型数据,进而显示其值及截断后的值的程序,如练习 4.12 所示。

〔练习 4.12〕

C>type exp4-12c

```
main ()
{
    double f;
    scanf("%lf",&f);
    printf("the float value%10.2f\n",f);
    printf("truncated is%d\n",(int)f);
}
```

```
C>exp4_12
12.2345678
the float value    12.23
truncated is 12
```

(3)溢出问题 在进行运算时,时常会遇到结果产生溢出现象,如练习 4.13 所示。

#### 〔练习 4.13〕

```
C>type exp4_13.c
main()
{ int x,y,z;
  x=400;y=200;
  z=x*y;
  printf("z=%d\n",z);
}
C>exp4_13
z=14464
```

在这个练习里,x、y、z、被定义为整型变量,x\*y的结果明显超出int型的数据范围。即使把变量z定义为long型,终因x和y的相乘是在整型范围内进行的,是不会有正确结果的。解决办法只有用类型显式转换,把x、y中的一个数转换为long型,这时,根据隐式转换规则,则另一个数也就自动转换为long型的了;然后再把long型的结果赋给事先已定义为long型的变量z,如练习 4.14 所示。

#### 〔练习 4.14〕

```
C>type exp4_14.c
main()
{ int x,y;
  long z;
  x=400;y=200;
  z=(long)x*y;
  printf("%ld",z);
}
C>exp4_14
80000
```

### 4.6 运算符优先级

在C语言中,各运算符是有其运算的优先级别的。运算符的优先级决定了表达式的求解顺序。C语言运算符的优先级如表 4.5 所示。

表 4.5 C 语言运算符优先级

运 算 符	分 类	结 合 性
() [] - > .		从左至右
! ~ ++ -- (type) * & sizeof	单项运算符	从左至右
* / % + - << >> <<= >>= == != & ^   && 	双项运算符	从左至右
? :	条件运算符	从右至左
= += -= *= /= %= >>-- <<-- &= ^=  =	赋值运算符	从右至左
,	逗号运算符	从左至右

从该表我们可以看出：

(1) C 语言有 44 个运算符。

(2) 这些运算符分 15 个优先级；优先级自上向下变低；其中()运算符优先级最高；逗号运算符优先级最低；赋值运算符的优先级也比较低。

(3) C 语言除了有优先级外，还有结合性问题。所谓结合性是指当表达式中同时出现多个优先级相同的运算符时的求解顺序。如果运算顺序是从左到右，叫左结合性；从右到左，叫右结合性。表中最右边一项，说明了各级运算符的结合性。

(4) C 语言中有 5 个运算符的结合性是可交换的。它们是 \*、+、&、|、^。

(5) 此外，注意 Turbo C 表达式中允许随意增加空格和使用多余的圆括号。

#### 〔练习 4.15〕

A>type exp4\_15.c

main()

```
{ printf("%d\n", 5>1&&! (1<5)||2<=3);
  printf("%d,%d\n", 5+3, 3+5);
  printf("%d,%d\n", 5|3, 3|5);
}
```

A>EXP4\_15

1

8,8

7,7



## 第五章 程序控制结构

本章介绍C程序控制结构。在C程序中,控制程序流程要使用控制结构。控制结构是由控制语句构成的。控制结构大致分为三种,即:

- 条件分支结构
- 循环控制结构
- 中断与转移结构

使用这三种控制结构,就能够编制出结构化程序。

本章的最后介绍递归函数,这是一种特殊的程序结构形式。

### 5.1 分支结构

(1)if 结构 其结构形式如下所示:

if(条件表达式) 语句

例如,求a,b两个数中大者的程序,就可以使用if结构编写,如练习5.1所示。

#### 〔练习 5.1〕

```
A>type exp5_1.c
main()
{ int a,b,x;
  a=5,b=3;
  x=b;
  if(a>b)x=a;
  printf("max=%d\n",x);
}
A>exp5_1
max=5
```

(2)if~else 结构 其结构形式如下:

if(条件表达式) 语句1  
else 语句2

#### 〔练习 5.2〕

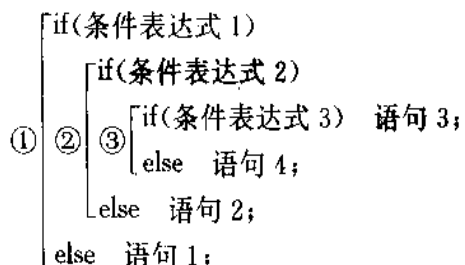
```
A>type exp5_2.c
main()
{ int a=5,b=3,x;
  if(a>b)x=a;
  else x=b;
  printf("max=%d\n",x);
}
```

```

}
A>exp5_2
max=5

```

(3) 嵌套 if~else 结构 if~else 结构中还可以放进 if 结构或 if~else 结构, 这叫做嵌套结构。其结构形式如下:



在这种嵌套结构中, 为使嵌套层次关系清晰, 我们提倡程序员把程序写成如上所示的齿状模样。

### [练习 5.3]

```

A>type exp5_3.c
main()
{ int number=55;
  /* nested if statement 1 */
  if(number>0){
    if(number%2==1)
      printf("%d positive odd\n", number);
  }
  else
    printf("%d negative\n", number);
  /* nested if statement 2 */
  number=-number;
  if(number>0){
    if(number%2==1)
      printf("%d positive odd\n", number);
    else
      printf("%d positive even\n", number);
  }
  else{
    if(number%2==0)
      printf("%d negative even\n", number);
    else
      printf("%d negative odd\n", number);
  }
}
A>exp5_3
55 positive odd
-55 negative odd

```

(4)多分支 if~else 结构 其结构形式如下:

```
if(条件表达式 1)语句 1
else if (条件表达式 2) 语句 2
else if (条件表达式 3) 语句 3
:
else if (条件表达式 n) 语句 n
else 语句 n+1
```

在这种结构形式中,程序只执行满足条件的语句。如果哪个条件都不成立,就执行语句 n+1。

#### [练习 5.4]

A>type exp5-4.c

```
main()
{ unsigned char c;
  printf("enter char:");
  scanf("%c",&c);
  if(c>0x00 && c<0x20)
    printf("%x,control code",c);
  else if(c>=33&& c<47)
    printf("%x[%c],special char(group 1)\n",c,c);
  else if(c>=48&& c<=57)
    printf("%x[%c],digits\n",c,c);
  else if(c>=58&& c<=64)
    printf("%x[%c],special char(group 2)\n",c,c);
  else if(c>=65&& c<=90)
    printf("%x[%c],uppercase letter\n",c,c);
  else if(c>=97&& c<=122)
    printf("%x[%c],lowercase letter\n",c,c);
  else if(c>=91&& c<=96)
    printf("%x[%c],special char (group3)\n",c,c);
  else if(c>=123 && c<=126)
    printf("%x[%c],special char(group 4)\n",c,c);
  else
    printf("%x[%c],non ascii char\n",c,c);
}
```

A>exp5-4

enter char:1

31[1],digits

A>exp5-4

enter char: \$

24[ \$ ],special char(group 1)

A>exp5-4

enter char:8.

26[&],special char(group1)

(5)switch 结构 switch 语句可实现多分支结构,其结构形式如下:

```
switch(变量)
{ case 常量表达式 1:
    语句 1
    break;
  case 常量表达式 2:
    语句 2
    break;
    :
  case 常量表达式 n:
    语句 n
    break;
  default:
    语句 n+1
}
```

当 switch 括号中的变量值与某 case 中常量表达式的值相同时,就执行该 case 中的语句。若该语句后跟有 break 语句,则跳出 switch 结构;若无 break 语句,则顺序执行下一个 case 中的语句。若变量的值与任何 case 的常量表达式的值都不相同,则执行 default 中的语句。使用 switch 结构需要注意以下几点:

①switch(变量)中的变量只限字符型(char)、整型(int)和枚举型(enum)

②switch 语句也可嵌套使用。

③若干个 case 可公用一个语句,如练习 5.5 所示。

〔练习 5.5〕

A>type exp5\_5.c

main()

```
{ int i;
  printf("enter i=");
  scanf("%d",&i);
  switch(i)
  { case 1:
      printf("case 1\n");
      break;
    case 2:
      printf("case 2\n");
      break;
    case 3:
    case 4:
    case 5:
      printf("case 3-5\n");
      break;
```

```

        default:
            printf("default");
    }
}
A>exp5-5
enter i=1
case 1
A>exp5-5
enter i=9
default

```

## 5.2 循环结构

用下面的循环控制语句,便可实现循环结构。

- for
- while
- do-while

(1)for 结构 for 结构的形式如下所示:

for(表达式 1;表达式 2;表达式 3) 语句

其中,表达式 1 一般用来初始化循环控制变量;表达式 2 为条件表达式,以此控制循环次数;表达式 3 用来修改循环控制变量。例如:

```

int i;
for(i=0;i<100;i++)
printf("%d",i);

```

和

```

int i;
for(i=100;i>0;i--)
printf("%d",i);

```

此外,可以使用两个循环控制变量,如:

```

int i,j;
for(i=0,j=100;i<j;i++,j--)语句

```

**〔练习 5.6〕**

```

A>type exp5-6.c
main()
{ int i,j=0;
  /* for statement 1 */
  for(i=0;i<100;i++)j+=i;
  printf("summation=%d\n",j);
}
A>exp5-6
summation=4950

```

(2)无限循环的 for 结构 for 语句的三个表达式中任何一个都可以缺省。当表达式 2 缺省时，C 编译系统则认为表达式 2 恒为真，故 for 结构就变成无限循环结构，其特例如下所示：

for(;;)语句

为使无限循环的 for 语句在某条件下终止，可加入 break 语句，如练习 5.7 所示。

#### 〔练习 5.7〕

A>type exp5-7.c

```
main()
{ int i,j;
  i=0;j=1;
  for(;;)
  { int k;
    k=j,j=i+j,i=k;
    if(k>100)break;
    printf("%d ",i);
  }
}
```

A>exp5-7

1 1 2 3 5 8 13 21 34 55 89

(3)while 结构 其结构形式如下所示：

while(条件表达式) 语句

功能是：只要条件表达式成立，就执行条件表达式后所跟的语句；该语句可以是空语句、简单语句或复合语句。例如：

```
int len=10;
while(len>0)
{printf(" * ");len--;}

```

只要 len 为正，该程序就打印 \* 号。

while 的条件表达式中可引进字符输入库函数，如：

while((ch=getchar())!= '\n');

其中 getchar() 为字符输入库函数。注意：该 while 语句的执行语句是空语句。这样，只要从键盘上输入的字符不是 \n，输入操作就可以连续进行下去。

#### 〔练习 5.8〕

A>type exp5-8.c

```
#include <stdio.h>
main()
{ char ch;
  int len=10;
  char s[80];
  printf("while statement test 1\n");
  while(len>0)
  { printf(" * ");
    len--;
  }
}
```

```

}
printf("\nwhile statement test 2\n");
while(scanf("%d",&len) != '\n')
    printf("len=%d\n",len);
}

```

A>exp5-8

```

while statement test 1
* * * * *
while statement test 2
2 3 4 5 (仿效返回)
: len=2
: len=3
: len=4
: len=5
^ C

```

执行结果中注有仿效返回的行,是为使操作者及时了解输入情况而设的操作系统功能。紧跟在仿效返回下面的,才真正是程序执行结果。

(4)无限循环 while 结构 其结构形式为:

```
while(1){.....}
```

只要圆括号中的值非 0,就是无限循环 while 结构。

#### [练习 5.9]

A>type exp5-9.c

```

main()
{ int len;
  printf("while statement test\n");
  len=10;
  while(1)
  { printf("%d",len--);
    if(len<=0)
      break;
  }
}

```

A>exp5-9

```

while statement test
10 9 8 7 6 5 4 3 2 1

```

(5)do-while 结构 其一般结构形式是:

```
do{语句}
while(条件表达式);
```

若条件表达式成立,则执行语句。与 while 结构不同的是,不管条件成立与否,首先要执行一次 do 后面的语句,如练习 5.10 所示。

#### [练习 5.10]

A>type exp5-10.c

```
#include <stdio.h>
main()
{ char ch;
  int len;
  printf("do-while statement test 1\n");
  len=10;
  do{ printf(" * ");
    } while((len--)>=0);
  printf("\ndo-while statement test 3\n");
  len=50;
  do{printf("%d",len--5);
    if(len<=0)break;
    } while(1);
}
A>exp5-10
do-while statement test 1
* * * * *
do-while statement test 3
45 40 35 30 25 20 15 10 5 0
```

### 5.3 中断结构

(1)中断循环结构 这种结构是由中断语句实现的,这里介绍两个用于C程序各种循环结构中的中断语句:

break;——跳出循环语句

continue;——继续判别循环条件语句

①break语句 该语句除了用在switch结构中,还常用于各种循环控制结构中,其功能是从其所在的循环结构中跳出,如练习5.11所示。

#### 〔练习5.11〕

```
C>type exp5-11.c
main()
{ int i,j,k;
  for(j=0;j<10;j++)
  { k=i+j;
    if(k>15) break;
  }
  printf("i      j      k\n");
  printf("%d  %d  %d\n",i,j,k);
}
C>exp5-11
i      j      k
0      0      0
0      1      1
0      2      2
0      3      3
0      4      4
0      5      5
0      6      6
0      7      7
0      8      8
0      9      9
```

②continue语句 用于各种循环结构中,常作if的执行语句,其功能是继续进行循环条件的判别,如练习5.12所示:



### [练习 5.12]

C>type a:exp5-12.c

```
main()
{
    int i,j;
    /* continue test 1 */
    for(j=4;j<10;j++)
    {
        for(i=0;i<50;i++)
        {
            if(i%j) continue;
            printf("%d",i);
        }
        printf("\n");
    }
    /* continue test 2 */
    i=0;
    while(++i<=9)
    {
        if(i==5) continue;
        printf("%d",i);
    }
}
```

C>exp5-12

```
0 4 8 12 16 20 24 28 32 36 40 44 48
0 5 10 15 20 25 30 35 40 45
0 6 12 18 24 30 36 42 48
0 7 14 21 28 35 42 49
0 8 16 24 32 40 48
0 9 18 27 36 45
1 2 3 4 6 7 8 9
```

(2)中止函数执行结构 这种程序结构使用 return 语句实现。该语句有两个功能：一是它将立即中断含有它的那个函数的执行，返回到调用此函数的语句处往下执行；二是它可以用来回送一个数值。C 程序中大多数函数是采用 return 语句来终止运行的，原因是有时函数须返回一个数值；有时需要在函数中设置多个终止点以提高效率。这就是说，一个函数可以有多个返回语句。如练习 5.13 所示。

### [练习 5.13]

C>type exp5-13.c

```
main()
{
    char str1[]="abcdefghijk1";
    char str2[]="efg";
    printf("str2 in str1=%d\n",substr(*str1,*str2));
}

substr(s1,s2)
char *s1,*s2;
{
    register int t;
```

```

char *p2, *p;
for(t=0;s1[t];t++)
{ p=&s1[t];
  p2=s2;
  while(*p2&&*p2==*p)
  { p++;p2++;
  }
  if(! *p2)return(t);
}
return(-1);
}
C>exp5_13
str2 in str1=4

```

(3)无条件转移结构 使用 goto 语句可实现无条件转移,其结构形式如下:

goto 标号;

功能是:它能转去执行标有该标号的语句。所谓标号,同变量一样,也是用标识符来表示,只是其后面跟有冒号而已。用 goto 语句转去执行的语句前面要放有标号。

goto 语句可从多重循环的某层次转到任何层次;而 break 语句只能从里向外跳转一层。这是 goto 语句的优点。使用 goto 必须注意如下两点:

①goto 语句只能在 goto 所在函数内跳转,不能转到函数外。

②goto 语句不适于结构化程序,因此,奉劝大家尽量少使用,以免造成程序难读难懂。

这里,介绍一个使用 goto 结构从最里层 for 结构中跳出的程序,如练习 5.14 所示。

#### 〔练习 5.14〕

```

A>type exp5_14.c
main()
{ int i,j,k;
  for(i=0;i<10;i++)
  { printf("i=%d\n",i);
    for(j=0;j<10;j++)
    { printf("i=%d,j=%d\n",i,j);
      for(k=0;k<10;k++)
      { goto stop;
        printf("%d,%d,%d\n",i,j,k);
      }
    }
  }
  stop:printf("from stop jmp.");
}
A>exp5_14
i=0
i=0,j=10
from stop jmp.

```

## 5.4 递归结构

从2章中,我们知道,一个函数可以调用另一个函数。此外,函数还可以调用其本身。这样的函数,就叫递归函数,递归函数就是一个递归结构。阶乘的算法就可用递归结构来实现,如练习5.15所示。

### 〔练习 5.15〕

```
A>type exp5_15.c
main()
{ int i;
  long int fact(int);
  for(i=1;i<10;i++)
    printf("fact(%d)=%8ld\n",i,fact(i));
}
long int fact(i)
int i;
{ if(i==1)
  return((long)i);
  else
    return((long)i * fact(i-1));
}
```

```
A>exp5_15
fact(1)=      1
fact(2)=      2
fact(3)=      6
fact(4)=     24
fact(5)=    120
fact(6)=    720
fact(7)=   5040
fact(8)=  40320
fact(9)= 362880
```

这里,函数 fact()就是递归函数。

下面,再介绍2个递归结构的用法。

第1个是求两个整数最大公约数的程序,如练习5.16所示。

### 〔练习 5.16〕

```
A>type exp5_16.c
#include <stdio.h>
main()
{ int i,j,k;
  char s[80];
  for (k=0;k<4;k++)
  { i=atoi(gets(s));
    if(i<0)
      exit(0);
```

```

    j=atoi(gets(s));
    printf("hcf(%d, %d)=%d\n", i, j, hcf(i, j));
}
}
hcf(i, j)
int i, j;
{ int r;
  r=i%j;
  if(r==0)
    return(j);
  else
    return(hcf(j, r));
}

```

A>exp5\_16

9

21

hcf(9, 21)=3

8

2

hcf(8, 2)=2

40

16

hcf(40, 16)=8

5

2

hcf(5, 2)=1

这里用的是欧几里得算法,即对于正整数  $p$  和  $q$ , 设  $p$  除以  $q$  的余数为  $r$ , 就有:

若  $r$  为 0, 则最大公约数就是  $q$ ;

若  $r$  不为 0, 则用  $q$  和  $r$  分别置换  $p$  和  $q$ , 重复进行上述判断。

第 2 个例子是求费班纳赛(Fibonacci)级数的程序。该级数为:

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, ……

可见, 每个元素是由相邻的前两个元素的和组成的, 如:

$$\begin{aligned}
 f(4) &= f(3) + f(2) \\
 &= \{f(2) + f(1)\} + \{f(1) + f(0)\} \\
 &= \{f(1) + f(0) + f(1)\} + \{f(1) + f(0)\} \\
 &= \{1 + 0 + 1\} + \{1 + 0\} \\
 &= 3
 \end{aligned}$$

其一般形式为:

$$n=0, 1 \text{ 时, } f(n)=n;$$

$$n \geq 2 \text{ 时, } f(n)=f(n-1)+f(n-2)$$

实现该级数的程序如练习 5.17 所示

**〔练习 5.17〕**

A>type exp5\_17.c

```
main()
```

```
{ int i=6;
```

```
  for(i=0;i<10;i++)
```

```
    printf("fib(%d)=%d\n",i,fib(i));
```

```
}
```

```
fib(i)
```

```
int i;
```

```
{ if(i<=1)
```

```
    return(i);
```

```
  else
```

```
    return(fib(i-1)+fib(i-2));
```

```
}
```

A>exp5\_17

fib(0)=0

fib(1)=1

fib(2)=1

fib(3)=2

fib(4)=3

fib(5)=5

fib(6)=8

fib(7)=13

fib(8)=21

fib(9)=34

## 第六章 宏指令的用法

典型的 C 源程序的编译过程是:用预处理程序处理带 # 号的宏预处理指令(简称宏指令);进行语法分析,产生目标代码;进行代码优化处理。

本讲,我们介绍 C 宏指令的用法。

### 6.1 #define 与 #undef

(1)define 这条宏指令叫宏定义。它用得比较多,也非常有用。其使用格式如下:

#define 标识符 字符串

这里出现的标识符也叫宏名。#define 的功能是:用字符串置换宏名。如练习 6.1 所示。

#### [练习 6.1]

```
A>type exp6-1.c
#define message "This is a message"
main()
{printf(message);}
```

A>exp6-1

This is a message

宏名可以带参数,如练习 6.2、6.3。

#### [练习 6.2]

```
A>type exp6-2.c
#define dmax(x,y) (x>y)? x:y
main()
{ int a=3,b=2;
  printf("%d\n",dmax(a,b));
}
```

A>exp6-2

3

#### [练习 6.3]

```
A>type exp6-3.c
#define begin {
#define end }
#define DMAX(x,y) x>y? x:y
main()
begin
  double x=1000.0,y=200.0;
  long lx=11,ly=21;
  int i;
```

```

printf("d_max=%lf\n",DMAX(x,y));
printf("l_max=%ld\n",DMAX(lx,ly));
end
A>exp6-3
d_max=1000.000000
l_max=21

```

宏定义经预处理的结果叫宏展开,练习 6.3 的宏展开程序是:

```

main()
{ double x=1000.0,y=200.0;
  long lx=11,ly=21;
  int i;
  printf("d_max=%lf\n",x>y? x:y);
  printf("l_max=%ld\n",lx>ly? lx:ly);
}

```

宏展开程序再经过语法分析,产生目标代码,代码优化,之后,经过链接程序 link 把程序的目标码和集中存放诸如 printf()、scanf() 等函数目标码的库链接起来,才使程序变为可执行程序。

带参数的宏的调用只是宏置换,因此,执行速度比较快,而预处理展开后的程序要占存储空间,故带参数的宏要比函数需要较多的存储空间。

宏名还可以嵌套使用,如练习 6.4 所示。

#### 〔练习 6.4〕

```

A>type exp6-4.c
#define a 10
#define b 20
#define c 30
#define v a*b*c
main()
{ printf("staiseki%d\n",v); }
A>exp6-4
staiseki 6000

```

使用宏定义应注意以下几点:

- ① # 前不能有空格或标识符(有些系统无此限制)。
- ② 宏指令不是 C 语句,故其末端不能有分号(;),否则,分号被看作是字符串的一部分。
- ③ 使用宏名时不能用双引号引起来,否则不能置换。
- ④ 宏指令一般放在文件开头部分,且只在该文件内有效;出了这文件,该宏指令失去作用。

(2) #undef 取消宏定义,其用法是:

#undef 标识符

其中标识符是由宏定义指令定义过的宏名。预处理程序检测到取消宏定义后,便使对应的宏定义失效。

#### 〔练习 6.5〕

```

A>type exp6.5.c
#define PAI 3.14159

```

```

main()
{ float r,l,a,v;
  r=4.0;
  l=2.0*PAI*r;
  #undef PAI
  a=PAI*r*r;
  v=(3.0/4.0)*PAI*r*r*r;
  printf("R=%6.2f\n",r);
  printf("L=%f\n A=%f\n V=%f\n",l,a,v);
}

```

从宏展开可以看出,在取消宏定义指令 `#undef PAI` 后的宏名 `PAI` 不能展开。因为预处理程序遇到这条宏指令后,便使宏定义 `#define PAI 3.14159` 失效。

## 6.2 #include

这条宏指令叫宏包含,其用法有两种:

```

#include "文件名"
#include <文件名>

```

遇到这两条指令之一,预处理程序就把文件名所指定的文件引进到本指令所在的源文件中,即用文件名所指定的文件内容置换 `#include` 指令行。

这两种宏包含指令是有区别的。遇到前者,预处理程序首先检索当前的文件目录有无该文件,若没有检索到,再检索 C 编译系统中的目录。遇到后者,预处理程序只检索 C 编译系统的目录。宏包含的实例如下:

### 〔练习 6.6〕

```

A>type u.h
#define begin {
#define end }
#define DMAX(x,y) x>y? x:y
C>type a;exp6_6.c
#include "a;u.h"
main()
begin
  double x=1000.0,y=200.0;
  long lx=11,ly=21;
  int i;
  printf("d-max=%lf\n",DMAX(x,y));
  printf("l-max=%ld\n",DMAX(lx,ly));
end
A>exp6_6
d-max=1000.000000
l-max=21

```

我们可以事先编好具有各种功能的标题文件(扩展名为 `.h` 的文件)或其它文件,然后,再用 `#include` 把它们包含到当前文件里使用。



stdio.h 是用得最多的标题文件,内容大致包括:符号定义、标准输入设备的定义及常数、函数类型的定义等。在根目录下可用如下命令查看。

```
C>cd tc\include
```

```
C>type stdio.h
```

标准输入输出函数 `getchar()` 和 `putchar()` 就是在该文件里定义的,故凡是要使用这两个函数的文件,其开头部分必须有宏包含:

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

或 

```
#include "stdio.h"
```

`getchar()` 的功能是从操作系统确定的标准输入设备输入一个字符。其特点是:没有参数;类型为整型;返回值为字符的 ASCII 码。用法:

```
int c;
```

```
c=getchar();
```

或 

```
char c;
```

```
c=getchar();
```

`putchar()` 的功能是从操作系统所确定的标准输出设备上输出一个字符,其特点是:必须把输出字符作为参数给出;其类型为整数;没有返回值。

下面是用这两个函数输入输出的最简单的例子:

#### [练习 6.7]

```
A>type exp6-7.c
```

```
#include "stdio.h"
```

```
main()
```

```
{ int c;
```

```
  c=getchar();
```

```
  putchar(c);
```

```
}
```

```
A>exp6-7
```

```
3(仿效返回)
```

```
3(执行结果)
```

```
A>exp6-7
```

```
w
```

```
w
```

下面是使用这两个函数连续输入输出的程序:

#### [练习 6.8]

```
A>type exp6-8.c
```

```
#include "stdio.h"
```

```
main()
```

```
{ for(;;)
```

```
  putchar(getchar());
```

```
}
```

```
A>exp6-8
```

```
abc.(仿效返回)
```

```
abc.(执行结果)
```

### 6.3 条件编译

这类宏指令有: #if、#ifdef、#ifndef、#endif、#else、#elif。它们按一定方式组合,构成条件编译程序结构。下面介绍这些程序结构。

(1) #if~#endif 结构 这种结构的用法如下:

```
#if 常量表达式
    程序模块
#endif
```

如果常量表达式的值不为 0,则 Turbo C 编译该程序模块;否则,不编译该程序模块。如练习 6.9 所示。

#### 〔练习 6.9〕

```
C>type exp6_9.c
#define MAX 10
#if MAX==0
#define MAX 5
#endif
main()
{ int i=6;
  do {
    printf("i=%d\n",i++);
  } while(i<MAX);
}
```

由于 MAX 先用 #define 定义为 10,其条件编译模块不被编译,故 MAX 仍保持为 10,执行结果如下:

```
C>exp6_9
i=6
i=7
i=8
i=9
```

如把 #define MAX 10 删去,则如练习 6.10。

#### 〔练习 6.10〕

```
A>type exp6_10.c
#if MAX==0
#define MAX 5
#endif
main()
{ int i=2;
  do{ printf("i=%d\n",i++);
    }while(i<MAX);
  }
A>exp6_10
i=2
```

```
i=3
```

```
i=4
```

(2) #if~#else~#endif 结构 这种结构的用法如下:

```
#if 常量表达式
```

```
语句 1
```

```
#else
```

```
语句 2
```

```
#endif
```

如果常量表达式的值不为 0,则编译语句 1;否则,编译语句 2,如练习 6.11 所示。

#### [练习 6.11]

```
A>type exp6-11.c
```

```
#define MAX 100
```

```
#if MAX>10
```

```
    #define MAX 10
```

```
#else
```

```
    #define MAX 5
```

```
#endif
```

```
main()
```

```
{ int i=6;
```

```
  do{printf("i=%d\n",i++);
```

```
    } while(i<MAX);
```

```
}
```

因为设定 MAX=100,根据 #if~#else~#endif 的功能,使 MAX=10,故程序执行结果如下:

```
A>exp6-11
```

```
i=6
```

```
i=7
```

```
i=8
```

```
i=9
```

(3) #if~#elif~#endif 结构 这种结构的用法如下:

```
#if 常量表达式 1
```

```
语句 1
```

```
#elif 常量表达式 2
```

```
语句 2
```

```
#endif
```

这里, #elif 可进行多次重复,如练习 6.12。

#### [练习 6.12]

```
A>type exp6-12.c
```

```
#define MAX 4
```

```
#if MAX>10
```

```
    #define MAX 10
```

```
#elif MAX>3
```

```
    #define MAX 3
```

```

#elif MAX>2
#define MAX 2
#endif
main()
{ int i=0;
  do{ printf("i= %d\n",i++);
    } while (i<MAX);
}
A>exp6_12
i=0
i=1
i=2

```

(4) #ifdef、#ifndef 这两条宏指令是用来识别宏名是否被定义过了。它们均可取代 #if，构成条件编译结构。对于 #ifdef，当其后跟的标识符被 #define 定义过时为真；对于 #ifndef，只有其后跟的标识符没有被 #define 定义过才为真。如练习 6.13 和 6.14。

#### 〔练习 6.13〕

```

A>type exp6_13.c
#define MAX 4
main()
{ #ifdef MAX
  printf("define MAX\n");
# else
  printf("not defined MAX\n");
#endif
}
A>exp6_13
define MAX

```

#### 〔练习 6.14〕

```

A>type exp6_14.c
#define MAX 4
main()
{ #ifndef MAX
  printf("not define MAX\n");
# else
  printf("defined MAX\n");
#endif
}
A>exp6_14
defined MAX

```

### 6.4 #line

这条宏指令可用来修改 Turbo C 中所定义的 \_\_LINE\_\_ 和 \_\_FILE\_\_ 等宏名的内容，用法如下：

#line 行号 "文件名"

注意该行号为源程序中当前行号,文件名为当前编译的文件名。

#### 〔练习 6.15〕

```
A>type exp6_15.c
main()
{ printf("line=%d\n",__LINE__);
  {
    #line 1000 "sub_block"
    printf("line=%d\n%s\n",__LINE__,__FILE__);
  }
}
```

A>exp6\_15

line=2

line=5

A:\exp6\_15.c

从执行结果可以看出,尽管我们在该程序 4 行中所指定的文件并非当前编译的文件 exp6\_15.c,但编译程序还是把\_\_FILE\_\_和\_\_LINE\_\_分别按当前编译文件名和源程序中当前行号进行处理的。再做一个练习。

#### 〔练习 6.16〕

```
C>type exp6_16.c
#line 100
main()
{ printf("LINE=%d\n",__LINE__);
}
```

C>exp6\_16

LINE=3

### 6.5 #pragma

该宏指令的用法如下:

#pragma 名字

在 Turbo C 中有两个名字可使用,即:

(1) #pragma inline 该宏指令对于 Turbo C 来说,表示程序内含有 inline 汇编码,等效于指定编译任选项 -B。

(2) #pragma warn 该宏指令与许可编译任选项 -wxxx 所指定的 xxx 相同,但由 #pragma warn 所指定的 xxx 优先。例如,若编译如下程序,则会因变量 i 不确定而发生警告。

#### 〔练习 6.17〕

```
C>type exp6_17.c
/* pragma warn-def */
main()
{ int i;
  if(i>=100) printf("i=%d\n",i);
}
```

编译如下程序则不会出现警告。

### 〔练习 6.18〕

```
C>type exp6_18.c
#pragma warn -def
main()
{ int i;
  if(i>=100)
    printf("i=%d\n",i);
}
```

## 6.6 #error

这条宏指令的用法是：

#error 错误信息

在编译过程中，一旦遇到这条宏指令，便会中止编译，读该宏指令，显示出错误信息后而停止。

### 〔练习 6.19〕

```
C>type exp6_19.c
main()
{ printf("Before #error");
  #error error—message
  printf("After #error");
}
```

信息窗口显示内容如下：

Compiling C:\EXP6-19.C:

Error C:\EXP6-18.C 3:Error directive:  
error—message in function main

## 6.7 内部宏名

ANSI 标准规定了 5 个内部宏名，即：\_\_LINE\_\_，\_\_FILE\_\_，\_\_DATE\_\_，\_\_TIME\_\_，\_\_STDC\_\_。Turbo C 增补了 10 个内部宏名，即：\_\_CDECL\_\_，\_\_COMPACT\_\_，\_\_HUGE\_\_，\_\_LARGE\_\_，\_\_MEDIUM\_\_，\_\_MSDOS\_\_，\_\_PASCAL\_\_，\_\_SMALL\_\_，\_\_TINY\_\_，\_\_TURBOC\_\_。

下面说明各内部宏名的定义值。

\_\_LINE\_\_ 存放当前编译的行号

\_\_FILE\_\_ 存放当前编译的文件名

\_\_DATE\_\_ 存放源文件被编译为目标文件的日期，存放形式为月/日/年。

\_\_TIME\_\_ 存放源文件被转换为目标文件的开始时间，以时:分:秒形式存放。

\_\_STDC\_\_ 若存有 10 进制常数 1，则代表使用的环境工具是标准工具程序；否则，为非标准工具程序。

\_\_CDECL\_\_ 使用标准 C 调用协议，即没有使用 pascal 选择项时，\_\_CDECL\_\_ 有定义；否则无定义。

\_\_TINY\_\_、\_\_SMALL\_\_、\_\_COMPACT\_\_、\_\_MEDIUM\_\_、\_\_LARGE\_\_、\_\_HUGE\_\_ 这 6 个内部宏名是表示存储模式的，因此，它们当中只有对应于编译中所定义的存储模式的那一个才被

定义。

\_\_MSDOS\_\_ 使用 MS-DOS 版 Turbo C 时被定义为 1。

\_\_PASCAL\_\_ 使用 Pascal 调用协议时, \_\_PASCAL\_\_ 被定义;否则不被定义。

TURBOC 存有使用的 Turbo C 的版本号。

#### [练习 6.20]

A>type exp6\_20.c

main()

```
{ printf(" %s %d %s %s\n", __FILE__, __LINE__, __DATE__, __TIME__);
```

```
    printf("Program being compiled using the  ");
```

```
#ifdef __TINY__
```

```
    printf("tiny model");
```

```
#endif
```

```
#ifdef __SMALL__
```

```
    printf("small model");
```

```
#endif
```

```
#ifdef __COMPACT__
```

```
    printf("compact model");
```

```
#endif
```

```
#ifdef __MEDIUM__
```

```
    printf("medium model");
```

```
#endif
```

```
#ifdef __LARGE__
```

```
    printf("large model");
```

```
#endif
```

```
#ifdef __HUGE__
```

```
    printf("huge model");
```

```
#endif
```

```
    printf("\n");
```

```
}
```

C>exp6\_20

EXP6\_20.c 3 Jan 01 1980 00:16:20

Program being compiled using the small model

## 第七章 结 构 数 据

二章介绍了基本数据类型,这里介绍结构数据。结构数据包括枚举型、数组型、结构型、共体型和指针型。本章介绍前四种。

### 7.1 枚举

枚举型也叫枚举型,其数据结构定义形式如下:

```
enum 枚举类型名 {枚举项表} 变量表
```

枚举类型名和变量表是任选项。例子如下所示:

```
enum color {red,green,blue};  
enum color tv;
```

这里 color 是枚举类型名, tv 为该枚举类型的一个变量。对于枚举变量,可以进行如下赋值和比较运算。

```
tv=red;  
if(tv==green) printf("tv color's green\n");
```

这时,若打印枚举项的值,即:

```
printf("%d %d %d",red,green,blue);
```

则输出为 0、1、2。可见,枚举型数据为整型常量。其枚举项表中各个标识符不仅能按 0、1、2...取值,而且能设定为任何整型常量。对于上例,若按如下初始化:

```
enum color {red=100,green,blue};
```

则 red、green、blue 就分别对应 100、101、102。

枚举型变量可用于控制结构中,如在 for 结构中:

```
main()  
{ enum color tv;  
  for(tv=red;tv<=blue;tv++)  
  {  
    :  
  }  
}
```

又如用于 switch 结构中:

```
switch(tv)  
{ case red:语句 1 break;  
  case green:语句 2 break;  
  case blue:语句 3 break;  
  default:语句 4  
}
```

巧妙地运用 enum 型结构数据,可使程序易读易懂,如练习 7.1 所示。



### [练习 7.1]

A>type exp7-1.c

main()

```
{ enum color{red,blue,green} c_char;
  /* enum type test 1 */
  printf("%d %d %d\n",red,blue,green);
  /* enum type test 2 */
  c_char=red;
  if(c_char==red) printf("RED\n");
  /* enum type test 3 */
  for(c_char=red;c_char<=green;c_char++)
    printf("%d",c_char);
  /* enum type test 4 */
  for(c_char=red; c_char<=green;c_char++)
  { switch(c_char)
    { case red;
      printf("\nred\n");
      break;
      case blue;
      printf("blue\n");
      break;
      case green;
      printf("green\n");
      break;
    }
  }
  printf("program end. \n");
}
```

A>exp7-1

0 1 2

RED

0 1 2

red

blue

green

program end.

## 7.2 数组

所谓数组在 C 语言中是指同一类型数据的集合。Turbo C 能定义任何维数的数组。

(1)一维数组 一维数组的定义格式为:

数据类型 数组名[尺寸];

例如:char name[16];

就定义了一个一维数组,其每个元素都是字符型数据,一共有 16 个这样的元素,数组名为 name。注

意,该数组的 16 个元素是从 name[0]开始,到 name[15]为止的。实例见练习 7.2。

### 〔练习 7.2〕

A>type exp7\_2.c

```
main()
{ char ch[10];
  int i;
  for(i=0;i<10;i++)
  { ch[i]='a'+i;
    printf("%c",ch[i]);
  }
}
```

A>exp7\_2

abcdefghij

使用 scanf() 函数给字符型数组输入一个字符串的程序,如练习 7.3。

### 〔练习 7.3〕

A>type exp7\_3.c

```
main()
{ char name[30]
  printf("What is your name:");
  scanf("%s",name);
  printf("Hello,%s\n",name);
}
```

A>exp7\_3

What is your name:WANG

Hello,WANG

注意,由于 name 为数组名,其值是数组本身的地址,因此,当作 scanf() 函数的参数时,其前面不需要取址算符 &。

(2)二维数组 其定义格式为:

数据类型 数组名[尺寸 1][尺寸 2];

例如:int mat[5][5];

就定义了一个 2 维整型数组;其行数为 5,列数亦为 5,共有 25 个元素;每个元素都是整型,如图 7.1。

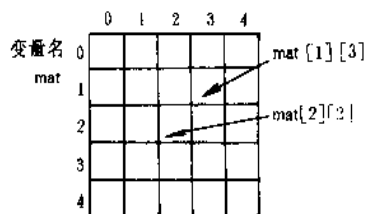


图 7.1 2 维数组的元素

二维数组赋值和输出的程序,如练习 7.4 所示。

### 〔练习 7.4〕

A>type exp7\_4.c

```
main()
{ int row=5,column=5,i,j;
  int mat[5][5];
  for(i=0;i<row;i++)
  { for(j=0;j<column;j++)
    { mat[i][j]=i*row+j;

```

```

        printf(" %2d ",mat[i][j]);
    }
    printf("\n");
}
}

```

C>exp7\_4

```

0   1   2   3   4
5   6   7   8   9
10  11  12  13  14
15  16  17  18  19
20  21  22  23  24

```

若对练习 7.4 进行修改,使程序执行结果为:

```

0   1   2   3   4   sum=10   accum=10
5   6   7   8   9   sum=35   accum=45
10  11  12  13  14   sum=60   accum=105
15  16  17  18  19   sum=85   accum=190
20  21  22  23  24   sum=110  accum=300

```

则程序如练习 7.5 所示。

#### [练习 7.5]

A>type exp7\_5.c

```

main()
{ int mat[5][5];
  int row=5,column=5,i,j;
  for(i=0;i<row;i++)
  { for(j=0;j<column;j++)
    { mat[i][j]=i * row+j;
      printf(" %2d ",mat[i][j]);
    }
    sum(mat[i]);
  }
}

sum(v)
int v[];
{ int i,ss=0,column=5;
  static int sss=0;
  for(i=0;i<column;i++)
  ss+=v[i];
  sss+=ss;
  printf("sum=%3d  accum=%3d\n",ss,sss);
}

```

该程序中,语句 `sum(mat[i])` 是函数调用,在函数 `sum()` 中引进实参 `mat[i]`,以计算 `i` 行元素之和和截止到 `i` 行的累加和。可见,二维数组的每一行可作为一维数组使用。

(3) 三维数组 其定义格式为:

数据类型 数组名[尺寸1][尺寸2][尺寸3];

例如: `int array[3][4][5];`

就定义了一个3维数组,它是由  $3 \times 4 \times 5 = 60$  个元素组成的数组,如图 7.2 所示。

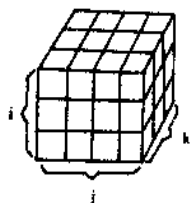


图 7.2 3 维数组  $[i][j][k]$  的元素

3 维数组的程序如练习 7.6 所示。

### (练习 7.6)

A>type exp7\_6.c

main()

{ int m=5,i,j,k;

int array[5][5][5];

for(i=0;i<m;i++)

{ for(j=0;j<m;j++)

{ for(k=0;k<m;k++)

{ array[i][j][k]=i\*m\*m+j\*m+k;

printf("%5d",array[i][j][k]);

}

printf("\n");

}

}

}

A>exp7\_6

0	1	2	3	4
5	6	7	8	9
10	11	12	13	14
15	16	17	18	19
20	21	22	23	24
25	26	27	28	29
30	31	32	33	34
35	36	37	38	39
40	41	42	43	44
45	46	47	48	49
50	51	52	53	54
55	56	57	58	59
60	61	62	63	64
65	66	67	68	69
70	71	72	73	74
75	76	77	78	79
80	81	82	83	84
85	86	87	88	89
90	91	92	93	94
95	96	97	98	99
100	101	102	103	104
105	106	107	108	109

110	111	112	113	114
115	116	117	118	119
120	121	122	123	124

(4)数组的初始化 前面我们分别介绍了1~3维数组的定义格式,以此类推,可以定义任何维数的数组。这里,我们介绍一下数组的初始化方法。

①1维数组可按如下进行初始化:

```
int array[2]={0,1};
```

②2维数组可按如下进行初始化:

```
int array[2][2] = {{1,0},
                    {0,1}};
```

练习 7.7 所示的初始化方法也是允许的。

#### [练习 7.7]

```
A>type exp7_7.c
int max=4;
int array[4][4]={1},
                {1,1},
                {1,1,1},
                {1,1,1,1}
}

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

```
A>exp7_7.c
```

```
1 0 0 0
1 1 0 0
1 1 1 0
1 1 1 1
```

在2维数组的定义中,行数是可以缺省的,如练习 7.8 所示。

#### [练习 7.8]

```
A>type exp7_8.c
main()
{ int max=5;
  int array[][5]={1},
                {1,1},
                {1,1,1},
                {1,1,1,1},
                {1,1,1,1,1}};
```

```

int i,j;
for(i=0;i<max;i++)
{ for(j=0;j<max;j++)
    printf("%2d  ",array[i][j]);
    printf("\n");
}
}
A<exp7-8
1 0 0 0 0
1 1 0 0 0
1 1 1 0 0
1 1 1 1 0
1 1 1 1 1

```

可以看出,2维数组 array 是按 5 行 5 列处理的。

字符数组可用双引号引起来的字符串进行初始化,如练习 7.9 所示。

#### [练习 7.9]

```

A>type exp7-9.c
main()
{ char a1[]="array of character";
  char a2[5]="char";
  printf("%s\n%s",a1,a2);
}
A>exp7-9
array of character
char

```

程序中,数组 a1 可初始化为任意长的字符串,而数组 a2 只限 5 个字符。

(5)数组参数 C 语言中,数组可以做函数的参数,可使函数调用多个同类型的数据,如练习 7.10。

#### [练习 7.10]

```

A>type exp7-10.c
#include "time.h"
#include <stdio.h>
#include "stdlib.h"
#define MAX 10000
main()
{ int i,s[MAX];
  long n;
  srand(time(&n));
  for(i=0;i<MAX;i++)
    s[i]=rand();
  printf("mean=%d",mean(s));
}
mean(a)

```

```

int a[];
{ int i;
  long s=0L;
  for(i=0,i<MAX;i++)
  { s+=a[i];
  }
  s/=MAX;
  return((int)s);
}
A>exp7-10
mean=16371

```

程序中出现的库函数 `srand()` 和 `rand()` 的原型包含在 `stdio.h` 中,用法和功能分别是:

`void srand(unsigned seed);` 初始化随机数发生器。

`int rand(void);` 产生随机数,返回值为  $0 \sim 2^{15}-1$ , 返回周期为  $2^{32}$ 。

### 7.3 结构

结构是结构数据的一种常用形式,它是用户自由设定,由基本数据类型构造出来的组合数据类型。

(1) 结构类型的定义 结构数据类型的定义方法有两种。

〈第一种定义方法〉 其定义格式如下:

```
struct 结构数据类型名{成员表};
```

其中结构数据类型名也叫标记,是人为设定的;成员是根据需要设置的;成员的组合就是结构类型的实在模式。例如:

```

struct date{
    int month;
    int day;
    int year;
};

```

就定义了由 3 个整型数据组成的一个结构数据类型,起名就叫 `date`。

〈第 2 种定义方法〉 使用数据类型定义关键字定义,如:

```

typedef struct{
    int month;
    int day;
    int year;
}date;

```

这里同样定义了由 3 个整型数据所组成的一个结构数据类型 `date`。注意, `typedef` 可以给已经存在的数据类型起新的名字,但不能定义新的存储类别。

(2) 结构变量的定义 结构变量有 2 种定义方式,比如,要定义具有 `date` 这种数据类型的变量,方法如下。

〈第一种定义方法〉 与结构类型分开定义,即:

```
struct date d1,d2[5];
```

其中 d1 和 d2 分别是 date 这种类型的一个变量和一个数组。注意, d2[5] 的每个元素均为 date 这样的结构数据, 共有 5 个这样的元素。

(第二种定义方法) 与结构类型同时定义, 如:

```
struct date{  
    int month;  
    int day;  
    int yeat;  
}d1,d2 [3];
```

(3) 结构成员的初始化与调用 结构变量定义后, 便可使用成员运算符及赋值运算符, 对其成员赋初值, 如:

```
d1. month=2;  
d1. day=7;  
d1. year=1991;
```

可见, 使用结构成员运算符., 便可指定结构的成员, 即

结构变量. 成员名

使用 printf() 和 scanf() 这两个库函数, 也可以对结构成员进行输入与输出, 如练习 7.11 所示。

#### [练习 7.11]

A.>type exp7-11.c

```
struct date{  
    int month;  
    int day;  
    int year;};  
  
main()  
{ struct date d;  
  scanf("%d",&d.year);  
  scanf("%d",&d.month);  
  scanf("%d\n",&d.day);  
  printf("%d",d.year);  
  printf("%d",d.month);  
  printf("%d\n",d.day);
```

A>exp7-11

1991 1 23

1991 1 23

对于结构数组, 其赋值方法与一般变量相同。例如, 对于结构数组 d2[5], 要对其 3 号元素 d2[3] 的各成员赋值, 可按如下进行:

```
d2[3]. month=5;  
d2[3]. day=27;  
d2[3]. year=1990;
```

(4) 结构的嵌套使用 结构还可以做另一结构的成员, 即结构可以嵌套使用。在 Turbo C 中, 结构嵌套层次不限, 但常用的为 2 重和 3 重, 如:



```

struct date {
    int month;
    int day;
    int year;
};

struct address {
    char name[20];
    char city [50];
    struct date zip;
}p;

```

这里,在结构 address 中就嵌套了结构 date。这时,对变量 p 的赋值和调用,则如下所示:

```

p.zip.month=9;
printf("%d",p.zip.month);

```

结构嵌套的例子如练习 7.12 所示。

#### 〔练习 7.12〕

A>type exp7-12.c

```

struct date {
    int month;
    int day;
    int year;
};

struct date a1,a2[5];

struct class {
    char name[10];
    struct date a3;
    int grade;
};

typedef struct {
    char address[20];
    int tel;
    float num;
    struct date a4;
    struct class a5;
}ddate;

main()
{ printf("size of date=%d\n",sizeof(struct date ));
  printf("size of class=%d\n",sizeof(struct class));
  printf("size of ddate=%d\n",sizeof (ddate));
}

```

C>exp7-12

```

size of date =6
size of class=18

```

size of ddate=50

## 7.4 字段结构

C语言可以进行字单元内的位操作,因此,在象对待布尔(Boolean)变量那样,进行1位(true或false)操作的场合,可在一个字单元内放置多个 Boolean 变量,这样,可节省存储空间。此外,在控制外设时,也必须进行位操作。结构是适合位操作的数据类型。C语言允许使用不满一个字的数据作结构的成员,这样的成员叫字段。由字段构造的结构就叫字段结构,也叫位域。

(1)字段结构的定义 其定义格式如下所示:

```
struct 字段结构类型名{
    数据类型 字段1名:长度
    数据类型 字段2名:长度
    :
}字段结构变量名;
```

这里,结构成员都是不满一个字的整型数据,故叫字段。字段名后(冒号后)的长度表示该字段的长度,即占几位,其值当然要小于cpu字长。注意,字段的数据类型为int或unsigned,如练习7.13所示。

### 〔练习 7.13〕

```
C>type exp7-13.c
main()
{ struct bit
{ unsigned f1:3;
  unsigned f2:5;
  int f0:5;
  unsigned f3:1;
  unsigned f4:2;
}bit_code;
bit_code.f1=0x2;
bit_code.f2=0x6;
bit_code.f3=1;
printf("%x %x %x\n",bit_code.f1,bit_code.f2,bit_code.f3);
printf("%x\n",bit_code);
}
C>exp7-13
2 6 1
2032
```

(2)字段结构的用法及注意事项:

①字段不存在地址,故不能使用运算符&,因而,也无指向字段的指针。

②在PC机上,字段在一个字上的分配方向是从右至左,如上例,其字段f1、f2、f0、f3、f4的分布情况如图7.3所示。

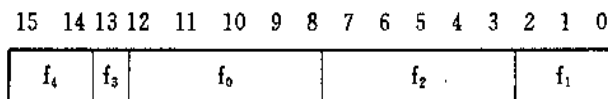


图 7.3 字段分配方向

③不能构造字段数组。

④如果字段未占满一个字存储单元,余下的位将保持原值不变。若余下的位已放不下某字段,则该字段开辟下一个存储单元使用。

⑤若想强制某字段放在一个字的开始部分,对 Turbo C 来说,只要在该字段的前面,放置一个长度为 16 的字段即可。

**〔练习 7.14〕**

C>type a;exp7-14.c

```
struct bitfield{
    unsigned d1:3;
    unsigned d2:4;
    unsigned d3:4;
    unsigned d4:16;
    unsigned d5:8;
    unsigned   :4;
    unsigned d6:5;
    unsigned d7:5;
}bit_code;
```

```
main()
{ unsigned *p;
  p=&bit_code;
  bit_code.d1=5;
  bit_code.d2=10;
  bit_code.d3=9;
  bit_code.d5=0xaa;
  bit_code.d6=12;
  bit_code.d7=0x0a;
  printf("%x\n", *p);
  printf("%x\n", *(p+1));
  printf("%x\n", *(p+2));
  printf("%x\n", *(p+3));
}
```

C>exp7-14

```
4d5
0
c0aa
14
```

## 7.5 共体

共体(union)是由使用同一存储空间的多个变量组成的结构数据。由于其各组员变量使用同一存储空间,故叫共体;有些专家直接翻译为联合。其定义例子如下所示:

```
union rec{
    int var1;
    int var2;
    char var3;
    char var4;
}
```

该定义确定了一个由 var1、var2、var3、var4 四个变量组成的共体类型 rec。四个变量的存储分配,如图 7.4 所示。

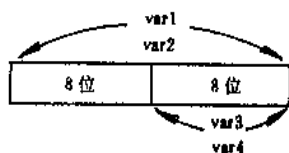


图 7.4 共体 rec 存储分配

又如,下面的定义:

```
union rec {
    int var1;
    char var2[2];
}
```

使变量 var1 所占用的低位字节和高位字节同时用来存储 var2[0]和 var2[1],其程序如练习 7.15 所示。

### [练习 7.15]

A>type exp7\_15.c

```
union rec{
    unsigned int var1;
    char var2[2];
}

main()
{union rec v;
 v.var1=0xaa66;
 printf("v.var1=%x\n",v.var1);
 printf("v.var2[0]=%x\n",v.var2[0]);
 printf("v.var2[1]=%x\n",v.var2[1]);
}
```

A>exp7\_15

```
v.var1=aa66
v.var2[0]=66
v.var2[1]=ffaa
```

可见,在共体中,同一存储空间要被多个变量使用。

模拟 8086 CPU 寄存器的结构设在标题文件 dos.h 中,如下所示:

```
struct WORDREGS{
```

```

        unsigned int ax,bx,cx,dx,si,di,cflag,flags;};

union REGS{
    struct WORDREGS x;
    struct BYTEREGS h;
};

struct SREGS{
    unsigned int es;
    unsigned int cs;
    unsigned int ss;
    unsigned int ds;
};

struct REGPACK {
    unsigned r_ax,r_bx,r_cx,r_dx;
    unsigned r_bp,r_si,r_di,r_ds,r_es,r_flags;
};

```

其中,共体 REGS 是由两个结构组成的。AX 寄存器的高 8 位和低 8 位分别用 ah 和 al 表示。使用共体 REGS,表示系统时间的程序如练习 7.16 所示。

#### [练习 7.16]

```

A>type exp7-16.c
#include "dos.h"
main()
{ union REGS in,out;
  in.h.ah=0x2c;
  intdos(&in,&out);
  printf("time,%2d,%2d,%2d",out.h.ch,out.h.cl,out.h.dh);
}
A>exp7-16
time,16:32:14

```

## 第八章 指针与数组

指针在 C 中占有重要位置。只有掌握了指针,才算把 C 学到手。这里,介绍指针结构及其各种用法。

### 8.1 一维数组的指针

下面的说明语句定义了一维数组 *s* 和指针变量 *p*。

```
char s[16], *p;
```

通过赋值语句:

```
p=s;
```

将在指针里赋上数组 *s* 的首地址,即指针 *p* 指向了数组 *s*。而且,数组元素 *s[i]* 可表示为  $*(p+i)$ ,元素 *s[5]* 就可用  $*(p+5)$  表示。因此,数组元素的读写,完全可由指针实现,如练习 8.1 所示。

#### 〔练习 8.1〕

```
A>type exp8_1.c
#define MAX 10
main()
{ int array[MAX], *ptr,i;
  ptr=array; /* 指针 ptr 指向数组 array */
  for(i=0;i<MAX;i++)array[i]=i; /* 变化数组下标,给数组赋值 */
  for(i=0;i<MAX;i++)printf("%3d ",array[i]);
  for(i=0;i<MAX;i++)*ptr++=i*i; /* 用指针给数组赋值 */
  printf("\n");
  for(i=0;i<MAX;i++)printf("%3d ",array[i]);
}
A>exp8_1
0  1  2  3  4  5  6  7  8  9
0  1  4  9 16 25 36 49 64 81
```

### 8.2 数组的动态分配

程序中设定的数组,到程序被编译时,其空间便被确定。但有时希望需要空间时再从称为堆的自由空间分配。为此,要使用存储管理库函数,诸如:

```
malloc(nbyte)
free(s)
```

来确定。这些函数能自由地占有或释放堆的存储空间,malloc()的调用形式如下:

```
int *p;
unsigned int nbyte;
p=malloc(nbyte);
```

结果把分配的存储空间的首址赋给指针 p。练习 8.2 是确定能存 1000 个整型数据存储空间的程序。

### 〔练习 8.2〕

```
a>type exp8-2.c
#include <stdio.h>
#include "stdio.h"
#define MAX 1000
main()
{ int *p, *s, i;
  s=p=malloc(MAX * sizeof(int));
  for(i=0; i<MAX; i++)
    *p++=1000+i;
  for(i=0; i<10; i++)
    printf("%x ", *(s+i));
  free(s);
}
```

A>exp8-2

3e8 3e9 3ea 3eb 3ec 3ed 3ee 3ef 3f0 3f1

执行该程序,函数 malloc()便确定了含有 1000 个整型元素的数组。当该数组不用时,该程序通过函数 free(),把所确定的存储空间返回给系统。

这样动态地分配堆空间,会更有效地利用存储空间,即该空间可多次用于不同目的,而且其数组的大小,可根据系统来设定,提高了程序的可移植性。这些函数,不仅可以动态分配数组,而且对于队列、链表、二叉树等,在动态分配存储空间时,也是必须的。

## 8.3 指针和字符串

在字符串的操作中,使用指针是很方便的。关于字符串的长度计算、拷贝、连接等操作,如练习 8.3 所示。

### 〔练习 8.3〕

```
A>type exp8-3.c
#define MAX 10
main()
{ char s[MAX], t[MAX], *p, *pp;
  int len=0;
  gets(s);
  p=s;
  /* 字符串长度计算 */
  while(*p++) len++;
  printf("length of string=%d\n", len);
  /* 字符串拷贝 */
  p=s; pp=t;
  while(*pp++=*p++);
  printf("string of t=%s\n", t);
  /* 字符串连接 */
}
```

```

p=s;pp=t;
while( *pp++ );
*pp--;
while( *pp++ == *p++ );
printf("string of t= %s\n",t);
}

```

A>exp8\_3

```

qwer
length of string=4
string of t=qwer
string of t=qwerqwer

```

程序中,对字符型数组 *s* 输入字符串,是使用函数 `gets()` 进行的;若使用指针,可按如下进行。

```

char *s;
s="string";

```

也可把这两个语句合并为:

```

char *s="string";

```

#### 8.4 指针的运算性质

适于指针的运算有如下四种。

(1) 指针加减一个整数 在 C 语言中,允许指针加减一个整数,其意义是当指针指向数组时,使指针相对原位置移动,以使其指向另一个数组元素。例如:

```

int a[10], *p;
p=a;

```

是使指针 *p* 指向了数组 *a*,即指向数组 *a* 的第一个(0号)元素。这时,若执行:

```

p+=i;

```

便使 *p* 指向数组 *a* 的 *i* 号元素。如练习 8.4 所示。

##### [练习 8.4]

A>type exp8\_4.c

```

#define W 10
main()
{ int a[W]={0,1,2,3,4,5,6,7,8,9}, *p;
  p=a;
  printf("%d,%d,%d\n", *p+3, *(p+3), *++p);
  printf("%d\n", *p++);
  printf("%d,%d,%d\n", *p, *(p+3), *(p-2));
}

```

A>exp8\_4

4,4,1

1

2,5,0

通过这个练习,应该对指针操作有如下了解。

① `* (p+3)` 与 `*p+3` 不同: `* (p+3)` 是取指针当前位置后面的第 3 个元素的值,而指针位置



并无改变, \*p+3 是先取 p 所指元素的值,再在该值上加 3。

② \*p++ 与 \*(p+1) 不同: \*p++ 是先取 p 所指元素的值,再把指针向后调整一个元素,且指针便固定在此位置上;而 \*(p+1) 则是取指针当前位置后面一个元素的值,指针位置不改变。

③ ++p 的结合性是 \*(++p)。因此,是先把指针向后调整一个元素,再取所指元素的值。

指针的这一运算性质,被广泛地用于堆栈的操作上。所谓堆栈是按照“先进后出”进行存取操作的存储空间。在堆栈操作中,使用指针变量是极为方便的,如练习 8.5 所示。程序中 push() 函数的功能是,把一个数据存入到指针所指定的堆栈空间内,且把指针向高地址移动一个位置。pop() 函数则把指针位置向低地址移动一个位置,取出当前指针所指定的值。

#### [练习 8.5]

```
A>type exp8_5.c
#include <stdio.h>
#define MAX 10
int *p; /* 指向自由区 */
int *t; /* 指向栈顶 */
int *b; /* 指向栈底 */
void push();
main()
{ int x,y,i,var;
  char s[80];
  p=(int *)malloc(MAX*sizeof(int)); /* 建栈 */
  if(!p)
  { printf("建栈失败");
    exit(1);
  }
  t=p;b=p+MAX-1
  printf("输入 10 个数:");
  for(i=0;i<10;i++)
  { scanf("%d",&var);
    push(var);
  }
  printf("弹出 5 个数:");
  for(i=0;i<5;i++)
  { x=pop();
    printf("%d",x);
  }
  printf("\n");
}
void push(i) /* 压栈 */
int i;
{ if (p>b)
  { printf("栈满");
    return;
  }
```

```

    *p=i;
    p++;
}
pop()/* 退栈 */
{ p--;
  if(p<t)
  { printf("栈空\n");
    return 0;
  }
  return *p;
}

```

A>exp8-5

输入 10 个数:1 2 3 4 5 6 7 8 9 1

弹出 5 个数:1 9 8 7 6

对于不同类型的指针,同样移动一个元素的位置,实际移动的距离(字节数)是完全不同的,如表 8.1 所示。

表 8.1 元素的大小

指针的数据类型	移动一个元素的实际距离
char	1
int	2
float	4
double	8

为了使指针移动时,能恰好指定相应元素,C 编译系统对指针所加减的整数乘上一个比例因子。这比例因子正是各数据类型所占字节数,如练习 8.6。

#### [练习 8.6]

A>type exp8-6.c

```

main()
{ char c[4]={'1234'}, *pc;
  int i[4]={1,2,3,4}, *pi;
  float f[4]={1,2,3,4}, *pf;
  double d[4]={1,2,3,4}, *pd;
  int n;
  pc=c;pi=i;pf=f;pd=d;
  for(n=0;n<4;n++)
    printf("c[%p]=%c\n",pc,*pc++);
    printf("\n");
  for(n=0;n<4;n++)
    printf("i[%p]=%i\n",pi,*pi++);
    printf("\n");
  for(n=0;n<4;n++)
    printf("f[%p]=%f\n",pf,*pf++);
    printf("\n");
  for(n=0;n<4;n++)
    printf("d[%p]=%f\n",pd,*pd++);
}

```

```

A>exp8_6
c[FF9F] =1
c[FFA0] =2
c[FFA1] =3
c[FFA2] =4

i[FFA4] =1
i[FFA6] =2
i[FFA8] =3
i[FFAA] =4

f[FFB0] =1.000000
f[FFB4] =2.000000
f[FFB8] =3.000000
f[FFBC] =4.000000

d[FFC6] =1.000000
d[FFCE] =2.000000
d[FFD6] =3.000000
d[FFDE] =4.000000

```

(2) 指针赋地址 这类运算有:

$p = \&var$ ; 把变量  $var$  的地址赋给指针  $p$ ;  
 $p = \&a[i]$ ; 把数组元素  $a[i]$  的地址赋给指针  $p$ ;  
 $p = pa$ ; 把数组  $pa$  的首地址赋给指针  $p$ ;  
 $p = px$ ; 把同类指针  $px$  的值赋给指针  $p$ 。  
 关于这类运算, 请看练习 8.7。

#### [练习 8.7]

```

A>type exp8_7.c
main()
{ int x=15, *p, *q, *r;
  int a[10]={1,2,3,4,5,6,7,8,9,10};
  q=a; p=q;
  printf("%p, %p, %p\n", a, q, p);
  p=&a[3]; q=&a[5]; r=&x;
  printf("%p, %p, %p\n", p, q, r);
  printf("%4d, %4d, %4d\n", *p, *q, *r);
}

```

```

A>exp8_7
FFCA,    FFCA,    FFCA
FFD0,    FFD4,    FFC8
    4,         6,         15

```

(3)两个指针比较运算 当两个指针指向同一数组时,可用关系运算符对它们进行比较。实际上,就是比较两个指针在该数组的相对位置。设  $px$  和  $py$  指向同一数组,那么,

$px < py$ ; 当  $px$  所指位置在  $py$  的之前时为真;  
 $px > py$ ; 当  $px$  所指位置在  $py$  的之后时为真;  
 $px \leq py$ ; 当  $px$  所指位置在  $py$  的之前或为同一位置时为真;  
 $px \geq py$ ; 当  $px$  所指位置在  $py$  的之后或为同一位置时为真;  
 $px == py$ ; 当  $px$  所指位置与  $py$  的相同时为真;  
 $px != py$ ; 当  $px$  所指位置与  $py$  的不相同时为真。

#### [练习 8.8]

```
A>type exp8_8.c
#define TRUE 1
#define FALSE 0
main()
{ if(isitpal("FOUR SCORE IN SEVEN YEARS")==TRUE)
  printf("string 1 is a pal indrome\n");
  if(isitpal("ABLE WAS I ERE I SAW ELBA")==TRUE)
  printf("string 2 is a pal indrome\n");
}
isitpal(str)
char *str;
{ char *str1=str;
  while(*str1!='\0') ++str1;
  --str1;
  while(str<str1)
    if(*str++!=*str1--)
      return(FALSE);
  return(TRUE);
}
A>exp8_8
string 2 is a pal indrome
```

(4)两个指针相减 当两个指针指向同一数组时,其差值是两指针相对移动的元素个数。利用这一运算性质,可以计算字符串的长度,如练习 8.9 所示。

#### [练习 8.9]

```
A>type exp8_9.c
main()
{char *s="abcdef ghijkl.\n";
char *p=s;
while(*p)
  p++;
printf("length of string s=%d\n",p-s);
}
A>exp8_9
length of string s=15
```

## 8.5 指针与多维数组

多维数组可用指针引用,这是因为多维数组在存储空间是按行存储的。用一般指针变量引用多维数组的程序,如练习 8.10 所示。

### 〔练习 8.10〕

```
A>type exp8_10.c
#include "time.h"
#include <stdio.h>
#include "stdlib.h"
#define ROW 4
#define COL 4
main()
{ int i,j,s[ROW][COL];
  long n;
  srand(time(&n));
  for(i=0;i<ROW;i++)
    for(j=0;j<COL;j++)
      s[i][j]=rand()/100;
  print_mat("mat s[][]=",s);
  trans(s);
  print_mat("trans mat=",s);
}
trans(a)
int a;
{ int i,j;
  for(i=0;i<ROW;i++)
    for(j=i+1;j<COL;j++)
      swap(a+i*ROW+j,a+j*ROW+i);
}
swap(a,b)
int *a,*b;
{ int t;
  t=*a;*a=*b;*b=t;
}
print_mat(s,mat)
char *s;
int mat[][col];
{ int i,j;
  printf("-----%s----\n",s);
  for(i=0;i<ROW;i++)
    { for(j=0;j<COL;j++)
      printf("%4d",mat[i][j]);
      printf("\n");
    }
```

```
A>exp8-10
```

```
-----mat s[][]=-----
```

```
199      177      263      84
242      119      115      262
239      165      44       288
73       68       29       60
```

```
-----trans mat=-----
```

```
1991      29682      -20224      0
21510      256      30465      0
239      165      44      288
73       68       29      60
```

## 8.6 指针数组

用户还可以定义指针数组,例如:

```
float *x[100];
```

就定义了一个指向浮点数的指针数组。可按如下所示,对该数组赋初值。

```
x[0]=a;
```

```
x[1]=b;
```

```
⋮
```

为了较清楚地理解指针数组的概念,这里,有必要再多说几句。假如我们定义了一个指针数组:

```
float *x[2];
```

注意,变量  $x$  不是二维数组,而是由两个指向一维  $\text{float}$  数组的指针所组成的数组。但对它适当地初始化,可用作二维数组。要想使它能表示  $2 \times 3$  的  $\text{float}$  型数组,只要按如下初始化即可。

```
x[0]=(float *)malloc(3 * sizeof(float));
```

```
x[1]=(float *)malloc(3 * sizeof(float));
```

现在  $x[0]$  和  $x[1]$  都指向能保存三个  $\text{float}$  型数据的存储空间,如图 8.11 所示。

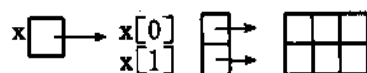


图 8.1 指针数组

其实例如练习 8.11 所示。

### [练习 8.11]

```
A>type exp8-11.c
```

```
char *err[]={ "EZERO 0    /* ERROR 0 */",
               "EIVLFNC 1 /* invalid function number */",
               "ENOFIL 2  /* File not found */",
               "ENOPATH 3 /* Path not found */",};
```

```
main()
```

```

{ int i;
  for(i=0;i<4;i++)
    printf("errno[%d]=%s\n",i,err[i]);
}

```

A>exp8-11

```

errno[0]=EZERO 0 /* ERROR 0 */
errno[1]=EIVLFNC 1 /* invalid function number */
errno[2]=ENOFILE 2 /* File not found */
errno[3]=ENOPATH 3 /* path not found */

```

引用命令行参数是使用指针数组的典型例子。所谓命令行即如下所示的执行命令：

A>程序名 参数1 参数2 .....

为引用命令行参数，主函数 main() 应按如下设置：

```

main(argc,argv)
int argc;
char * argv[];
{...}

```

其中，argc 的值等于参数总数+1；argv 是指针数组，其内容如下：

argv[0] 的内容为程序名的地址

argv[1] 的内容为参数 1 的地址

argv[2] 的内容为参数 2 的地址

作为例子，请看练习 8.12。

#### [练习 8.12]

A>type exp8-12.c

```

main(argc,argv)
int argc;
char * argv[];
{ while(argc--)
  printf("argv[%d]=%s\n",argc,argv[argc]);
}

```

A>exp8-12 arg1 arg2 arg3 arg4

argv[4]=arg4

argv[3]=arg3

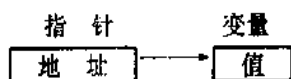
argv[2]=arg2

argv[1]=arg1

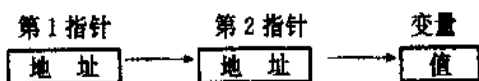
argv[0]=A:\EXP8-12.EXE

## 8.7 指向指针的指针

应该指出的是，指针数组是指向指针的指针，即形成一个指针链。前面我们所介绍的指针其值是某变量的地址，称之为一级指针；而在指向指针的指针里存放的是第二个指针的地址，其实际值是由第二指针指出的，称之为二级指针，如图 8.2 所示。



(a)一级指针



(b)二级指针

图 8.2 一级指针和二级指针

在C语言里,用  $**x$  表示  $x$  为指向指针的指针,其作用相当于  $*x[]$ 。例如,在程序中,我们定义了:

```
float **x;
```

则意味着  $*x$  的类型是指向 `float` 的指针,而  $**x$  的类型是 `float`,即用  $**x$  可引用值。这样的结构按如下初始化后,也可表示二维数组。

```
x=(float **)malloc(2*sizeof(float *));
x[0]=(float *)malloc(3*sizeof(float));
x[1]=(float *)malloc(3*sizeof(float));
```

关于指向指针的指针的用法,请看练习 8.13。

#### (练习 8.13)

A>type exp8-13.c

```
main()
{
    int **b,i,j;
    int a[3][3]={ {1,2,3},
                  {4,5,6},
                  {7,8,9},
                };
    b=(int **)malloc(3*sizeof(int *));
    b[0]=(int *)malloc(3*sizeof(int *));
    b[1]=(int *)malloc(3*sizeof(int *));
    b[2]=(int *)malloc(3*sizeof(int *));
    b[0]=a;
    b[1]=a+1;
    b[2]=a+2;
    printf("-----\n");
    for(i=0;i<3;i++)
        for(j=0;j<=3;j++)
            if(j==3)
                printf("\n");
            else
                printf("%d ",*(b+i+j));
    printf("-----\n");
}
```



A>exp8\_13

```
1 2 3
4 5 6
7 8 9
```

## 8.8 数组指针

下面的说明语句便定义了一个指向由三个 float 型数据所组成的数组的指针。

```
float (*x)[3];
```

它的值最初是不确定的,必须使用 malloc() 函数来确定。按如下所示初始化,便可用来表示 2 \* 3 的 2 维数组。

```
x=(float(*)[])malloc(2*3*sizeof(float));
```

这样,x 所指向的存储空间,不是保存一个由三个 float 型元素所组成的数组,而是连续存放这样的两个数组。x 的结构如图 8.3 所示。

练习 8.14 是求某年某月某日是这一年第几天的程序,该程序就使用了数组指针。

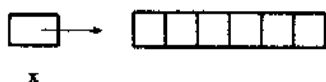


图 8.3 数组指针

### [练习 8.14]

A>type exp8\_14.c

```
int dtab[][13]={
    {0,31,28,31,30,31,30,31,31,30,31,30,31},
    {0,31,29,31,30,31,30,31,31,30,31,30,31},
};
```

```
main()
```

```
{ int y,m,d,yd;
  scanf("%d,%d,%d",&y,&m,&d);
  yd=dofy(dtab,y,m,d);
  printf("%d:%d:%d=%d\n",y,m,d,yd);
}
```

```
dofy(x,ty,tm,td)
```

```
int (*x)[13],ty,tm,td;
{ int i,j;
  j=ty%4==0&&ty%100!=0||ty%400==0;
  for(i=1;i<tm;i++)
    td+=*(x+j)+i;
  return(td);
}
```

A>exp8\_14

1991,2,11

现把本节所介绍的可用于表示二维数组的指针结构小结一下,如表 8.2 所示,供参考。

表 8.2 指针结构用于二维数组小结表

指针结构	名 称	行、列情况	左 值	效 率
float xa[2][3]	二维数组	静态数组(固定行列)	xa, xa[0]均不是左值	高
float * xb[2]	指针数组	固定行、动态列	xb 不是左值, xb[0]是左值	低,节省空间
float( * xc)[3]	数组指针	固定列、动态行	xc 是左值, xc[0]不是左值	低,节省空间
float * * xd	指针的指针	行、列皆为动态	xd, xd[0]皆为左值	低,节省空间

## 第九章 指针与函数、结构

### 9.1 函数指针

指针不仅可以指向一般变量、数组,而且可以指向函数。我们把指向函数的指针叫做函数指针,其一般定义形式如下:

数据类型 (\*标识符)();

例如: int (\*p)();

就表示 p 是一个指向函数的指针,且说明 p 所指向的函数的返回值是整型数据。

函数指针的用法,如练习 9.1 所示。

#### 〔练习 9.1〕

```
A>type exp9-1.c
main()
{ int (*p1)(),print1();
  int (*p2)(),print2();
  p1=print1;
  p2=print2;
  (*p1)(123);
  (*p2)(456);
}
print1(x)
int x;
{ printf("%d\n",x);}
print2(y)
int y;
{ printf("%d\n",y);}
A>exp9-1
123
456
```

在该程序的函数体中,

1、2 行为说明语句,说明 p1、p2 为函数指针,print1()和 print2()为函数;

3、4 行为函数指针赋值语句,用来使 p1 和 p2 分别指向函数 print1()和 print2();

5、6 行为函数语句方式调用,即利用函数指针 p1 和 p2,分别调用函数 print1()和 print2(),其中 123 和 456 为代替形参的实参。

再看一个例子,如练习 9.2 所示。

#### 〔练习 9.2〕

```
A>type exp9-2.c
```

```

#include "time.h"
#include <stdio.h>
#include "stdlib.h"
static int (* cmpf)();
#define MAX 10
main()
{ int quick(int *,int,int(*)()),cmp(int,int);
  int i,array[MAX];
  long now;
  srand((unsigned)time(&now));
  for(i=0;i<MAX;i++){
    printf("%d ",array[i]=rand());
  }
  quick(array,MAX,cmp);
  printf("\n\n");
  for(i=0;i<MAX;i++){
    printf("%d ",array[i]);
  }
  printf("\n\n");
}

quick(item,count,cmp)
int item[],count,(* cmp)();
{ cmpf=cmp;
  qs(0,count-1,item);
}

qs(l,u,item)
int l,u,item[];
{ int i,j,x,y;
  i=l;j=u;
  x=item[(l+u)/2];
  do { while((item[i]>x)&&(i<u)) i++;
    while((item[j]<x)&&(j>l)) j--;
    if(i<=j)
    { y=item[i];
      item[i]=item[j];
      item[j]=y;
      i++;j--;
    }
  }while(i<=j);
  if(l<j)qs(l,j,item);
  if(i<u)qs(i,u,item);
}

int cmp(x,y)
int x,y;
{ if(x>y) return(-1);
  else if(x==y) return(0);
}

```

```
return(1);
}
```

A>exp9\_2

15250 7082 21072 29365 24173 1478 22943 7166 10947 23544

29365 24173 23544 22943 21072 15250 10947 7166 7082 1478

该练习是快速排序程序。快速排序是靠函数 `qs()` 实现的, `qs()` 为递归函数, 2 次递归调用其本身。快速排序原理是以数组的中间元素为基础值, 把比中间元素值大的放在中间元素的一边, 小的放在另一边。把小的放在中间元素的左边, 是升序; 而把大的放在左边是降序。按升序还是按降序, `qs()` 中是靠连结 `item()` 与 `x` 的关系运算符决定的。这里, `qs()` 中是按降序处理的。第 1 次迭代的中间元素其值为 5, 以它为基准进行比较, 则把比 5 大的换到 5 的左边, 把小的放在 5 的右边, 如图 9.1 所示。

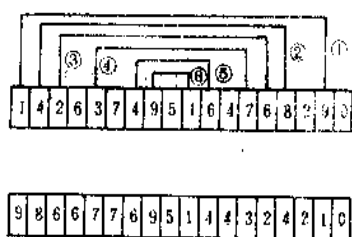


图 9.1 `qs()` 降序第 1 次排序后的情况

改变 `item[i]` 与 `x` 间、`item[j]` 与 `x` 间不等号的方向, 即可改为升序排序。

指向函数的指针用 `static` 说明为全局变量, 即:

```
static int (*cmpf)();
```

其中 `cmpf` 为指针名, 用它指向比较函数 `cmp()`, 即:

```
cmpf=cmp;
```

## 9.2 返回指针的函数

说明语句:

```
int *p();
```

表示 `p` 是一个函数, 一个返回整型指针的函数。返回指针的函数的用法, 如练习 9.3 所示。

### [练习 9.3]

A>type exp9\_3.c

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

```
main()
```

```
{ char *cpy(char *,char *);
```

```
char s[]="abcd";
```

```
char d[]="efgh";
```

```
printf("%s",cpy(s,d));
```

```
}
```

```
char *cpy(s,d)
```

```
char *s,*d;
```

```

{ int i;
  char *p;
  p=d;
  if(*s==NULL) exit(-1);
  do { *d++=*s++;
      }while(*s!=NULL);
  return(p);
}

```

A>exp 9\_3

abcd

该程序是从 s[ ] 中把字符串拷贝到 d[ ] 中。这时函数 cpy() 的返回值为指针 d。

### 9.3 指针参数与函数参数值传递方法

在 C 语言里,函数的参数值有两种传递方法。

(1) 值传递法(call by value) 值传递的例子如练习 9.4 所示。

〔练习 9.4〕

A>type exp9\_4.c

```

main()
{ int j,j;
  char a;
  i=110,j=220;a='c';
  func(i,j,a);
}

func(i,j,k)
int i,j;
char k;
{ printf("arg1=%d\narg2=%d\narg3=%c\n",i,j,k);
}

```

A>exp9\_4

arg1=110

arg2=220

arg3=c

该程序调用函数 func() 时,引用 3 个参数。

(2) 参考传递法(call by reference) 这是使用指针的值传递,其例如练习 9.5 所示。

〔练习 9.5〕

A>type exp9\_5.c

```

main()
{ int i=111,j=222;
  char a='a';
  func(&i,&j,&a);
  printf("i=%d\nj=%d\na=%c\n",i,j,a);
}

```

```
func(i,j,k)
int *i, *j;
char * k;
{ printf("arg1=%d\narg2=%d\narg3=%c\n", *i, *j, *k);
  (*i)++;
  *j++;
  *k++;
}
```

A>exp 9-5

arg1=111

arg2=222

arg3=a

i=112

j=222

a=a

从程序中,我们可以看到,函数 func()的三个参数 i、j、k 都是指针,也就是说 i、j、k 为指针参数。这样,在调用它时,必须使其参数与之对应,即也必须是指针类型,因此,要在一般变量 i、j、a 前面冠以取址运算符 &,即使其三个参数为:

&i,&j,&a

同时,我们看到,函数 func()的调用,只改变了变量 i 的值,并没有改变变量 j 和 a 的值。

不返回值的函数,可用关键 void 来说明,如练习 9.6 所示。

#### 〔练习 9.6〕

A>type exp9-6.c

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

```
main()
```

```
{ void pnt(char *);
```

```
  char s[]="abcd";
```

```
  pnt(s);
```

```
}
```

```
void pnt(s)
```

```
char *s;
```

```
{ do{ printf("%c", *s++);
```

```
  }while(*s!=NULL);
```

```
}
```

A>exp9-6

abcd

从练习 9.6 可以看出,在 Turbo C 中可用 void 类型来显式说明一个无返回值的函数,除此之外,void 还有如下一些用法。

①用来说明一个空参数表 如练习 9.7 所示。

#### 〔练习 9.7〕

A>type exp9-7.c

```
void putmsg(void)
```

```
{printf("Hello,world\n");
```

```

/
main()
{puts("exp9.7");
A>exp9.7
Hello, world

```

②用来作一个特殊结构,如:

```
(void) getch();
```

其功能为暂停执行,直到按任何键为止。

③说明一个 void 指针。注意,这不是空指针,而是建立一个数据目标为任何类型的指针,对它们的类型无需知道。可把任何指针赋值为 void 指针,反之亦然,而无需事先安排。

void 指针不能使用 \*。

## 9.4 指针与结构

(1)指针型结构成员 指针变量同样可以作结构成员。下面的结构定义中,就把指针变量 pname 当作了结构成员。

```

struct person{
    int number;
    char * pname;
}

```

这样的结构用法,如练习 9.8 所示。

### [练习 9.8]

```

A>type exp9_8.c
#include <stdio.h>
struct person{
    int number;
    char * pname;};

main()
{ int i;
  struct person qp[10];
  for(i=0;i<10;i++)
    qp[i].pname=NULL;
  qp[1].number=100;
  qp[1].pname="aoi";
  qp[2].number=200;
  qp[2].pname="akai";
  /* print out */
  i=1
  do { printf("%d  %s\n",qp[i].number,qp[i].pname);
      i++;
    }while(qp[i].pname!=NULL);
}

```



A>exp9\_8

100 aoi

200 akai

(2)结构指针 指针不仅可以指向变量、数组、指针、函数,也可以指向结构。我们把指向结构的指针,叫做结构指针。结构指针在链表、二叉树等数据结构中是很有用的。设一链表的结构为:

```
struct address{
    char name [20];
    char city [40];
    char zip [5];
    struct address * next;
};
```

如果定义如下指针 top:

```
struct address * top;
```

则指针 top 就是指向该结构的一个结构指针。这时,就可使用->运算符,引用该结构的各个成员,如下所示:

```
top->name
top->city
top->zip
top->next
```

连接结构数据可使用引用自身的结构,如练习 9.9 中的结构 address。练习 9.9 可连接 4 个新的结构,并能读出全部第一成员姓名。

#### [练习 9.9]

A>type exp9\_9.c

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

```
#include "stdlib.h"
```

```
#include "alloc.h"
```

```
struct addresss {char name[9];
    char city[7];
    char zip[7];
    struct address * next;
};
```

```
main()
```

```
{ struct address * new, * last=0;
  int i=0;
  do{ if((new=malloc(sizeof(struct address)))==0)
    { printf("out of memory\n");
      exit(0);
    }
  if(last==NULL)
    new->next=NULL;
  scanf("%s",new->name);
```

```

scanf("%s", new->city);
scanf("%s", new->zip);
new->next = last;
last = new;
}while((i++) < 3);
do{printf("%s\n", last->name);
last = last->next;
}while(last);
}

```

A>exp9-9

```

王宜宾    天津    300071
李文华    河北    050065
卢元来    北京    100073
林俞锡    东北    154600
林俞锡
卢元来
李文华
王宜宾

```

可以改变连接方向的程序,如练习 9.10 所示。

#### [练习 9.10]

A>type exp9-10.c

```

#include <stdio.h>
#include "stdlib.h"
#include "alloc.h"
struct address {  char name[9];
                  char city[7];
                  char zip[7];
                  struct address * next;
                };

main()
{ struct address * new, * last=0; * top=0;
  int i=0;
  do {if((new=malloc(sizeof(struct address)))==0)
    { printf("out of memory\n");
      exit(0);
    }
    if(top==NULL)
      new->next=NULL;
    scanf("%s", new->name);
    scanf("%s", new->city);
    scanf("%s", new->zip);
    last->next=new;
    last=new;
  }while((i++) < 3);
}

```

```

do{ printf("%s\n",top->name);
    top=top->next;
}while(top);
}

```

A>exp9\_10

李一民    天津市    300074

鲁晓春    北京市    100009

王里黎    河北省    050002

林太恣    山东省    400003

(null)

李一民

鲁晓春

王里黎

林太恣

## 第十章 文件管理

本章介绍文件的概念、标准输入/输出、文件输入/输出、低级文件 I/O 函数、随机文件的存取。

### 10.1 文件的概念

(1) 文件系统及其功能 文件是指具有名字(文件名)的一组信息序列。系统和用户都可以将具有一定功能的程序模块、一组数据或文字命名为一个文件。一般的操作系统中都有专门负责文件管理的文件管理系统(File Manager)。其基本功能有:

- ①能够建立新文件或删除旧文件;
- ②能对文件进行读写操作;
- ③能为文件自动分配物理空间,并建立逻辑结构与物理位置的映象关系;
- ④能对文件实施维护。

(2) 文件结构 文件结构分逻辑结构和物理结构。

①文件的逻辑结构 它有两种形式:一是记录式文件,亦称标准文件;另一种是非结构的流式文件。

记录式文件是一种结构文件。例如,DOS 系统的文件结构是由文件、记录块、记录三要素组成的,其结构如图 10.1 所示:

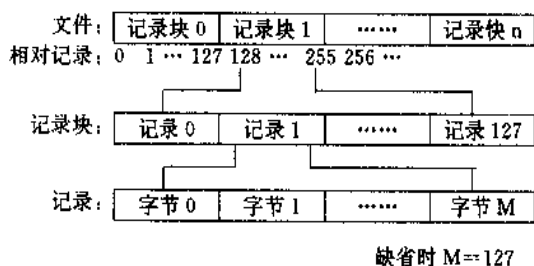


图 10.1 DOS 文件的逻辑结构

由该图可以看出:

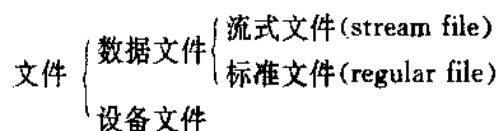
- 一个 DOS 文件由若干个记录块组成。
- 一个记录块由 128 条记录组成。
- 一个记录由若干字节(缺省时为 128)组成。所有记录的长度都相等的文件,叫定长记录文件,其文件长度取决于记录数;记录长度不相等的文件,叫变长记录文件,其文件长度为各记录长度的累加和。

非结构流式文件是字符的有序集合。Tur-boc C 提供了这种文件逻辑结构形式。流式

文件的长度是该文件所包含的字符数。这种文件,很便于操作系统进行管理;也适于用户进行文本处理,可不受任何约束地灵活组织其内部逻辑结构。

②文件的物理结构 它是指文件在外存上的存储结构,可分为连续、链接和索引三种组织方式。文件的读写方法有 2 种,即顺序读写和随机读写。

(3) C 语言的文件种类 C 语言中文件的分类如下:



在 C 语言中,I/O 设备可以当作文件来处理,这样的文件叫设备文件。设备文件既可按流式文件处理,又可按标准文件处理。常用的设备文件其名称、文件指针,如表 10.1 所示。

表 10.1 常用设备文件

设备	文件名	文件指针
键盘	CON: /KYBD:	stdin
显示器	CON: /SCRN:	stdout/stderr
打印机	PRN: /LPT1:	stdout
串行	AUX: /COM1:	

## 10.2 标准 I/O

前面我们介绍了数据的按格式输入输出。这里以文件处理为中心,介绍 I/O 处理。

(1)标准 I/O 所谓标准输入输出是指在操作系统所认可的所谓标准 I/O 设备上进行的输入输出。把标准输入文件拷贝到标准输出上的程序,如练习 10.1 所示。该程序是利用标准输入函数 `getchar()` 和标准输出函数 `putchar()` 实现的。程序中的 EOF 是文件结尾符。

### 〔练习 10.1〕

```
C>type exp10-1.c
#include <stdio.h>
main()
{ char c;
  while((c=getchar())!=EOF)
    putchar(c);
}
```

C>exp10.1

学习 Turbo C ,使用 Turbo C .

学习 Turbo C ,使用 Turbo C .

^ Z

C 语言除了支持标准输入、标准输出外,还支持输入输出重定向,以及文件追加、管道功能。

(2)输入输出重定向 输入重定向的符号为<,其用法如下:

`outf <inf`

功能是 `outf` 文件使用函数 `getchar()`、`scanf()`、`gets(char *)`、`cgets(char *)`,从文件 `inf` 中读取数据,且实现 `outf` 的功能。

输出重定向的符号为>,其用法如下:

`inf >outf`

功能是文件 `inf` 使用函数 `putchar()`、`printf()`、`puts(char *s)`、`cputs(char *s)`,把数据输出到文件 `outf` 中。

此外,<和>还可以同时使用,如:

prog <newin >newout

功能是文件 prog 从文件 newin 中读取数据,把文件 prog 的处理输出到文件 newout 中。

因为文件 exp10\_1c 中含有 putchar()函数,故执行 exp10\_1>a.txt 输出重定向命令,便可把从标准输入设备(键盘)上输入的数据,通过程序中函数 putchar(),输出到 a.txt 文件中;即改变了 putchar()函数的输出方向。如练习 10.2 所示。

**〔练习 10.2〕**

C>exp10\_1 >a.txt

学习 Turbo C,

使用 Turbo C.

C>type a.txt

学习 Turbo C,

使用 Turbo C.

^ Z

同样,命令:

A>exp10\_1 <a.txt >b.txt

改变了函数 getchar()的输入方向,从文件 a.txt 输入数据;同时亦改变了 putchar()的输出方向,把 exp10\_1 要输出的数据输出到文件 b.txt 中。如练习 10.3 所示。

**〔练习 10.3〕**

C>exp10\_1 <a.txt >b.txt

C>type b.txt

学习 Turbo C,

使用 Turbo C.

(3)文件追加 其符号为>>,用法如下:

fild >>output

功能是把文件 fild 的内容附加到文件 output 的原有内容之后。例如:

A>exp10\_1 <a.txt >>b.txt

执行结果是把文件 a.txt 中的数据,追加到文件 b.txt 的原有内容之后,如练习 10.4 所示。

**〔练习 10.4〕**

C>exp10\_1 <a.txt >>b.txt

C>type b.txt

学习 Turbo C,

使用 Turbo C.

学习 Turbo C,

使用 Turbo C.

(4)管道 其符号为|,用法如下:

inpr |outpr

功能是执行文件 inpr 和 outpr,且使 inpr 的标准输出作为 outpr 的标准输入,等效于:

inpr >temp

outpr <temp

del temp

文件 exp10\_5.c 是对字符串计数的程序,执行命令:

```
C>exp10_1 <exp10_1.c |exp10_5
```

可以实现对文件 exp10\_1.c 计数,如练习 10.5 所示。

#### 〔练习 10.5〕

```
C>type exp10_5.c
```

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

```
main()
```

```
{ int i=0;
```

```
  char c;
```

```
  while((c=getchar())!=EOF)
```

```
    if(c != '\n') i++;
```

```
  printf("total character=%d\n",i);
```

```
}
```

```
C>exp10_1 <exp10_1.c |exp10_5
```

```
total character=87
```

C 语言之所以能够实现 I/O 重定向、文件追加、管道功能,就在于其输入输出函数能够重新分配标准输入输出设备,如练习 10.6 所示。

#### 〔练习 10.6〕

```
C>type exp10_6.c
```

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

```
#include "ctype.h"
```

```
main()
```

```
{ char ch;
```

```
  while((ch=getchar())!=EOF)
```

```
    { if(ch=='&')
```

```
      putc(ch,stderr);
```

```
      else
```

```
        putchar(ch);
```

```
    }
```

```
}
```

```
C>exp10_6 >x.txt
```

```
abcdef
```

```
a&b&c&d
```

```
&&&
```

```
&&&
```

```
&&&
```

```
a b c d
```

```
^ Z
```

```
C>type x.txt
```

```
abcdef
```

```
ab&d
```

a b c d

由文件 exp10\_6.c 和执行 type 命令的结果来看,标准输出(stdout)分配给文件 x.txt,标准诊断输出分配给显示终端。可见,根据程序的功能,可以把从键盘输入的一部分字符(这里为 &)分离出来,输出到显示终端上(兼作标准诊断输出)。注意,当把标准输入(stdin)分配给其他文件,便不能从键盘进行输入操作;要想从键盘上输入,就必须把标准输入重新分配给键盘。

### 10.3 缓冲型文件的输入输出

经常用到的使用缓冲区的文件输入输出函数如表 10.2 所示。

表 10.2 常用的缓冲型文件 I/O 函数

函数名	功 能
fopen()	stream 的打开
fclose()	stream 的关闭
putc()	向 stream 输出一个字符
getc()	从 stream 输入一个字符
fseek()	查找 stream 的指定字节
fprintf()	输出到 stream
fscanf()	从 stream 输入
feof()	文件结尾(EOF)时为真
ferror()	发生错误时返回真
rewind()	把文件位置指示器定位在文件的开头
remove()	删除文件

用如下命令行:

exp10\_7 file1 file2

实现程序拷贝的程序如练习 10.7 所示。

#### 〔练习 10.7〕

```
C>type exp10_7.c
#include <stdio.h>
main(argc,argv)
int argc;
char *argv[];
{FILE *from,*to;
  char ch;
  if(argc!=3)
  { printf("Usage:exp10_7 from-file to-file\n");
    exit(1);
  }
  if((from=fopen(argv[1],"rb"))==NULL)
    { printf("Can't open from-file\n");
```



```

        exit(1);
    }
    if((to=fopen(argv[2],"wb"))==NULL)
    { printf("Can't open to- file\n");
      exit(1);
    }
    while(! feof(from))
    { ch=getc(from);
      putc(ch,to);
    }
    fclose(from);
    fclose(to);
}
C>type a.txt
学习 Turbo C,
使用 Turbo C.
C>exp10_7 a.txt exp.txt
C>type exp.txt
学习 Turbo C,
使用 Turbo C.

```

该程序的功能是把文件 file1 的内容拷贝到文件 file2 中。变量 from、to 是指向结构 FILE 的指针。因此，命令行第 1、第 2 参数是文件名，分别用 fopen() 函数打开。若能打开，则返回指向结构的指针值；若打开失败，返回 NULL 的值。若两个命令行参数所指定的文件都打开成功，则用 feof() 函数检查 EOF，便可进行文件之间内容的拷贝了。

函数 getw() 和 putw() 可用于整型数的输入输出，即输出用 putw(n, file)，功能是把变量 n 的值输出到文件指针 file 所指向的文件中。输入用 getw(file)，功能是从文件指针 file 所指向的文件中读取下一个数据。利用这两个函数进行整数输入输出的程序，如练习 10.8 所示。

#### 〔练习 10.8〕

```

C>type exp10_8.c
#include <stdio.h>
#define NMAX 20
main()
{ FILE *from, *to;
  char *fname="fint";
  int i;
  if((to=fopen(fname,"wb"))==NULL)
  { printf("Can't open to- file\n");
    exit(1);
  }
  for(i=0;i<NMAX;i++)
    putw(i,to);
  fclose(to);
  if((from=fopen(fname,"rb"))==NULL)

```

```

    { printf("Can't open from file\n");
      exit(1);
    }
    for(i=0;i<NMAX;i++)
        printf("%d ",getw(from));
    printf("\n");
}

```

C>exp10-8

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19

要随机存取文件,可使用能指定文件内存任意位置的函数 `fseek()`。该函数的用法如下所示:

```
int fseek(FILE *fp, long int n_byte, int org)
```

其中, `fp` 为文件指针, 打开文件时使用。 `n_byte` 为距 `org` 所设定场所的字节数。 `org` 可使用如下宏:

<code>SEEK_SET</code>	文件的开头位置
<code>SEEK_CUR</code>	文件的现在位置
<code>SEEK_END</code>	文件的最末位置

使用函数 `fseek()` 进行随机存取的程序如练习 10.9 所示。设文件名从命令行引进。读文件使用函数 `fread()`, 其用法如下:

```
int fread(void *buf, int num_bytes, int count, FILE *fp)
```

功能是从文件指针 `fp` 所指定的文件中读取每字段为 `num_bytes` 个字节, 一共 `count` 个字段的数, 放到指针 `buf` 所指定的缓冲区中。

#### [练习 10.9]

C>TYPE EXP10-9.C

```

/* example for fseek(), fread() */
#include <stdio.h>
#include "ctype.h"
#define SIZE 128
unsigned char buf[SIZE];
main(argc, argv)
int argc;
char *argv[];
{ FILE *fp;
  int sect, read_num;
  if(argc != 2)
  { printf("Usage: exp10-9 filename\n");
    exit(1);
  }
  if((fp=fopen(argv[1], "rb")) == NULL)
  { printf("Can't open file\n");
    exit(1);
  }
  do{ printf("Enter sector:");

```

```

scanf("%ld",&sect);
if(fseek(fp,sect * SIZE,SEEK_SET))
{ printf("fseek error\n");
  exit(1);
}
if((read_num=fread(buf,1,SIZE,fp))!=SIZE)
{ printf("EOF reached\n");
  exit(1);
}
display(read_num);
}while(sect>=0);
}
display(n)
int n;
{ int i,j;
  for(i=0;i<(n/16);i++)
  { for(j=0;j<16;j++)
    printf("%3x",buf[i * 16+j]);
    printf(" ");
    for(j=0;j<16;j++)
    { if(isprint(buf[i * 16+j]))
      printf("%c",buf[i * 16+j]);
      else printf(".");
    }
    printf("\n");
  }
}

```

A>exp10\_9 exp10\_9.obj

Enter sector:0

```

80 b 0 9 45 58 50 31 30 5f 39 2e 43 15 88 1d ....EXP10_9.C...
0 0 0 19 54 43 38 36 20 42 6f 72 60 61 6e 64 ....TC86 Borlend
20 54 75 72 62 6f 20 43 20 32 2e 30 20 fc 88 11 Turbo C 2.0...
0 0 e9 c3 52 ea 10 9 45 58 50 31 30 5f 39 2e ....R...EXP10_9.
43 3 88 21 0 0 e9 0 10 1d 11 19 45 3a 5c 54 C..!.....E;\T
55 52 42 4f 43 5c 49 4e 43 4c 55 44 45 5c 53 54 URBOC\INCLUDE\ST
44 49 4f 2e 48 b8 88 22 0 0 e9 0 10 1d 11 1a DIO.H..*.....
45 3a 5c 54 55 52 42 4f 43 5c 49 4e 43 4c 55 44 E;\TURBOC\INCLUD

```

Enter sector:1

```

45 5c 53 54 44 41 52 47 2e 48 74 88 21 0 0 e9 E\STDARG.H. {...
0 10 1d 11 19 45 3a 5c 54 55 52 42 4f 43 5c 49 .....E;\TURBOC\I
4e 43 4c 55 44 45 5c 43 54 59 50 45 2e 48 b6 88 NCLUDE\CTYPE.H..
6 0 0 e5 1 0 0 8c 88 a 0 0 e3 19 0 2 .....
0 15 8 4 4f 88 a 0 0 e3 18 0 2 0 15 19 ....0.....
4 3f 88 15 0 0 e6 4 61 72 67 76 18 a 6 00 .?...argv....
4 61 72 67 63 4 a 4 0 ee 88 6 0 0 e5 1 .argc.....
6 0 86 88 48 0 0 e2 0 5 60 65 76 65 6c 4 ....H.....level.

```

Enter sector:0

在使用文件打开函数 fopen() 时,用参数 "rb" 指定为只读打开二进制文件模式。

删除文件,使用如下函数:

```
int remove(char * filename);
```

其应用程序如练习 10.10 所示。

#### [练习 10.10]

```
C>dir *.bak
Volume in drive C has no Label
Directory of C:\
EXP7_13 BAK    428    1-01-80    12:22a
EXP10_1 BAK     97    1-01-80    12:05a
EXP6_19 BAK    549    1-01-80    12:02a
EXP10_6 BAK    182    1-01-80    12:15a
      4 FILE(S) 12830720 bytes free
```

```
C>type exp10_10.c
#include <stdio.h>
#include "dos.h"
main()
{ char fp[80];
  printf("enter remove filename,");
  gets(fp);
  remove(fp);
}
```

```
C>exp10_10
enter remove filename,exp10_6.bak
```

```
C>dir *.bak
Volume in drive C has no label
Directory of C:\
EXP7_13 BAK    428    1-01-80    12:22a
EXP10_1 BAK     97    1-01-80    12:04a
EXP6_19 BAK    549    1-01-80    12:02a
      3 File(s) 12832768 bytes free
```

### 10.4 非缓冲型文件的输入输出

10.3 中所介绍的文件输入输出函数,使用系统自动设定的缓冲区。这里介绍向用户指定的缓冲区直接进行输入输出的函数。这样的输入输出叫无缓冲区的文件输入输出。这些函数叫低级输入输出函数,也叫操作系统连接的文件子程序。这些函数有:creat()、close()、open()、read()、write()、setmode()、rename()、unlink()。

关于这些函数的功能、用法和返回值,请参考清华大学出版社出版的《IBM PC C 语言例题习题 库函数》。

这些低级输入输出函数,使用如下标题文件:

```
io.h
fcntl.h
\sys\stat.h
```

文件打开的一般形式如下:

```
int open (char * filename,int access,[int permiss]);
```

其中 filename 为文件名,access 指定文件存取模式,取文件 fcntl.h 中所定义的任何一种,如图 10.2 所示:

```
/* The first three can only be set by open */
#define O_RDONLY 1
#define O_WRONLY 2
#define O_RDWR 4
/* Flag values for open only */
#define O_CREAT 0x0100 /* create and open file */
#define O_TRUNC 0x0200 /* open with truncation */
#define O_EXCL 0x0400 /* exclusive open */
/* a file in append mode may be written to only at its end. */
#define O_APPEND 0x0800 /* to end of file */
/* MSDOS special bits */
#define O_CHANGED 0x1000 /* user may read these bits, but */
#define O_DEVICE 0x2000 /* only RTL\io functions may touch */
#define O_TEXT 0x4000 /* CR—LF translation */
#define O_BINARY 0x8000 /* no translation */
/* DOS 3.x options */
#define O_NOINHERIT 0x80
#define O_DENYALL 0x10
#define O_DENYWRITE 0x20
#define O_DENYREAD 0x30
#define O_DENYNONE 0x40
```

图 10.2 文件存取模式

另外,当指定为 O\_CREAT 时,permisss 可指定为如下存取模式当中一个,如图 10.3 所示。

```
#define S_IREAD 0x0100 /* owner may read */
#define S_IWRITE 0x0080 /* owner may write */
```

图 10.3 文件的存取模式

或者指定为:

S\_IREAD | S\_IWRITE

open()函数的返回值为文件句柄,读、写或关闭时使用。关闭文件时,使用 close()函数:

```
int close(int fd);
```

文件读写时,分别使用 read()和 write():

```
int read(int fd,void * buf,int size);
```

```
int write (int fd,void * buf,int size);
```

使用低级输入输出函数的例子,如练习 10.11 所示。该程序的流程为:文件的生成、数据的写进、文件的关闭、文件以读方式打开、数据的读出、文件的关闭。

### [练习 10.11]

C>type exp10\_11.c

```
#include <stdio.h>
#include "io.h"
#include "fcntl.h"
#include "sys\stat.h"
#include "string.h"
#define SIZE 80
#define ERR -1

main()
{ int i,fp;
  char *fname="test.f";
  char buff[SIZE];
  if((fp=creat(fname,S_IWRITE | S_IREAD))==ERR)
  { printf("Can't create %s",fname);
    exit(1);
  }
  /* write to file */
  printf("Enter string-->\n");
  do{ gets(buff);
     for(i=strlen(buff);i<SIZE;i++)
     buff[i]='\0';
     if(write(fp,buff,SIZE)!=SIZE)
     { printf("Error on write()\n");
       exit(1);
     }
  }while(strcmp(buff,"end"));
  close(fp);
  /* read from file */
  if((fp=open(fname,O_RDONLY))==ERR)
  { printf("Error on read()\n");
    exit(1);
  }
  while(1)
  { if(read(fp, buff,SIZE)==0) break;
    printf("%s\n",buff);
  }
  close(fp);
}
```

C>exp10\_11

Enter string-->

111111111111

11111111

111111

```

111
11
1
end
111111111111
11111111
111111
111
11
1
end

```

## 10.5 文件的随机存取

要想实现文件的随机存取,就得使用能使文件指针定位在任何位置的函数 `lseek()`,其用法如下:

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

该函数把对应句柄值的文件指针从 `fromwhere` 位置移动 `offset` 个字节。其中, `fromwhere` 按如下取值:

```

SEEK_SET(0)    文件的开头
SEEK_CUR(1)    当前文件指针位置
SEEK_END(2)    文件的末尾

```

函数的返回值为文件指针的新位置距文件开头的偏移量。要知道文件指针的当前位置,使用函数 `tell()`:

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

使用这些函数的程序,如 练习 10.12 所示。该程序根据输入的所要打开文件的扇区号,便可显示其内容。若 `lseek` 的返回值为 `-1`,则意味着发生错误。该程序既能读出文本文件,也可以读出二进制文件。

### [练习 10.12]

```

C>type exp10_12.c
#include "io.h"
#include "fcntl.h"
#define SIZE 80
#define ERR -1
main()
{ int i, fh, n;
  char buf[SIZE], s[80];
  long offset;
  printf("Enter filename:");
  gets(s);
  if((fh=open(s, O_RDONLY)) == ERR)
  { printf("Can't open %s\n", s);
    exit(1);
  }
}

```

```

}
while(1)
{ printf("Enter the sector number:");
  offset=(long)(atoi(gets(s))*SIZE);
  if(lseek(fh,offset,SEEK_SET)==-1)
  { printf("lseek() error\n");
    exit(1);
  }
  if((n=read(fh,buf,SIZE))==ERR)
  printf("read error\n");
  printf("current file pointer=%ld\n",tell(fh));
  for(i=offset;i<offset+SIZE;i++)
  { printf("%3x ",buf[i]);
    if((i%10)==9)
      printf("\n");
  }
  if(eof(fh)==1) break;
}
close(fh);
}

```

C>exp10\_12

Enter filename:exp10\_12.obj

Enter the sector number:0

current file pointer=80

80	c	0	a	45	58	50	31	30	5f
31	32	2e	43	e9	88	1d	0	0	0
19	54	43	38	36	20	42	6f	72	6c
61	6e	64	20	54	75	72	62	6f	20
43	20	32	2e	30	20	fc	88	12	0
0	e9	0	57	ea	1c	a	45	58	50
31	30	5f	31	32	2e	43	95	88	1e
0	0	e9	0	10	1d	11	16	45	3a

Enter the sector number:1

current file pointer=160

31	0	70	31	30	5f	31	32	2e	6f
62	6a	0	ec	83	ec	4	ff	76	6
ff	76	4	9a	84	20	36	38	89	56
fe	89	46	fc	b0	f	50	52	ff	76
fc	ff	76	a	ff	76	8	52	ff	76
fc	33	c0	50	b8	6b	0	50	b8	a4
0	50	e	e8	ce	ff	4c	5	10	0
0	0	0	0	d8	ff	15	6	10	0

Enter the sector number: ^ C



从执行结果来看,文件指针的位置为 80 的整数倍。但是,实现输出的字节数,0 扇区和 2 扇区是不同的,并不都是 80 个字节。原因是 0AH 前面的 0DH 给删除掉了。若把 open()说明为:

```
if(fh=open(s,O_RDONLY|O_BINARY))!=ERR)
{ printf("Can't open %s\n",s);
  exit(1);
}
```

则每个扇区的目标码便是 80 个字节,且上例中出现的 0AH,也变成了 0DH、0AH 了。

# 第十一章 字符屏幕管理

Turbo C 有较强的字符屏幕管理功能,这些功能是靠函数实现的,本章就介绍有关函数及其用法。

## 11.1 PC 机显示适配器及其模式

IBM PC 系列微机提供了多种显示适配器,Turbo C 2.0 支持如下 8 种显示适配器,即:

- ①CGA——彩色图形卡
- ②MCGA——多色图形阵列
- ③EGA——增强图形适配器,包括 EGA、EGA64、EGAMONO 三种
- ④VGA——视频图形阵列
- ⑤HERCMONO——大力神图形适配器
- ⑥ATT400——AT&T400 线图形适配器
- ⑦PC3270——3270 PC 图形适配器
- ⑧IBM8514——IBM8514 图形适配器

这些适配器支持不同模式的屏幕显示(简称屏显)。标准屏显模式如表 11.1 所示。

表 11.1 IBM PC 机显示器模式

模式	模 式	分辨率	适配器
0	字符,黑白	40×25	CGA, EGA, VGA, MCGA
1	字符,16 色	40×25	CGA, EGA, VGA, MCGA
2	字符,黑白	80×25	CGA, EGA, VGA, MCGA
3	字符,16 色	80×25	CGA, EGA, VGA, MCGA
4	图形,4 色	320×200	CGA, EGA, VGA, MCGA
5	图形,4 灰度	320×200	CGA, EGA, VGA, MCGA
6	图形,黑白	640×200	CGA, EGA, VGA, MCGA
7	字符,黑白	80×25	EGA, VGA
13	图形,16 色	320×200	EGA, VGA
14	图形,16 色	640×200	EGA, VGA
15	图形,单色	640×350	EGA, VGA

从表 11.1 可以看出,屏显的一些模式是文本模式,一些是图形模式。文本模式是进行字符操作,最小单位是字符;图形模式可用来绘图,其最小单位为像素。图形模式下,左上角定位为 0.0;文本模式下,左上角定位为 1.1。两种模式的屏幕座标如图 11.1 所示。

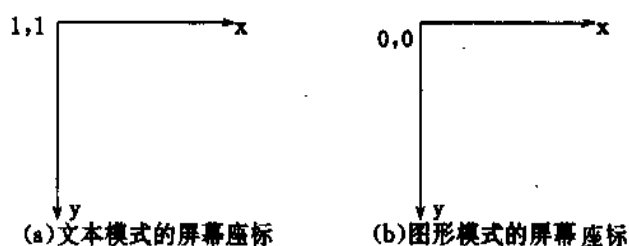


图 11.1 屏幕的坐标

## 11.2 用于窗口的 I/O 函数

用于窗口的 I/O 函数如表 11.2 所示。

表 11.2 用于窗口的 I/O 函数

函 数	功 能
cprintf()	按格式将输出送到当前窗口
cputs()	将字符串送到当前窗口
putch()	将字符送到当前窗口
getche()	读一个字符且回显到当前窗口
cgets()	读字符串且回显到当前窗口

使用这类函数时, 请注意以下几点:

①进行全屏幕操作时, 无论是使用窗口 I/O 函数, 还是使用标准 I/O 函数, 效果是一样的。但要进行小于屏幕的窗口操作, 就要使用窗口函数。

②窗口函数的用法, 与相应的标准 I/O 函数完全相同, 因为它们就是由相应标准 I/O 函数修改过来的。

③这些窗口函数不能改变 I/O 方向, 这一点与标准 I/O 函数不同。

## 11.3 屏幕操作函数

字符屏幕操作函数如表 11.3 所示。

表 11.3 屏幕操作函数

函 数	功 能
clrscr()	清除字符窗口
clrscr()	清除从光标至行尾的字符
delline()	删除光标所在行
gettext()	将屏幕上一个区域的文字拷贝到内存
gotoxy()	将光标移到指定位置
insline()	在光标所在行的下方插入一空行
movetext()	将屏幕上一个区域的文字拷贝到另一个区域
puttext()	将内存中的文字拷贝到屏幕上的一个区域
textmode()	将屏幕设置为字符操作
window()	定义字符操作窗口

使用这类函数时, 请注意以下几点:

- ①不论窗口大小、位置如何,其左上角坐标都按(1,1)处理。
- ②由于篇幅限定,这里不给出各函数的原型,使用时,请查阅有关资料。
- ③使用这类函数,要注意坐标值的超界问题。
- ④调用 window()函数时所用的坐标是屏幕的绝对坐标;而 window()调用成功后,则所有坐标都被转换为相对于该窗口的坐标。
- ⑤函数 textmode()的原型是:

void textmode(int mode);

参数 mode 的取值必须从表 11.4 中选取,可以用符号,也可用整型值。

**表 11.4 字符屏显模式**

模 式	符号	数值
40 列黑白	BW40	0
40 列彩色	C40	1
80 列黑白	BW80	2
80 列彩色	C80	3
80 列单色	MONO	7
回到原模式	LAST	-1

下面,练习一下上述两类函数的用法。

#### 〔练习 11.1〕

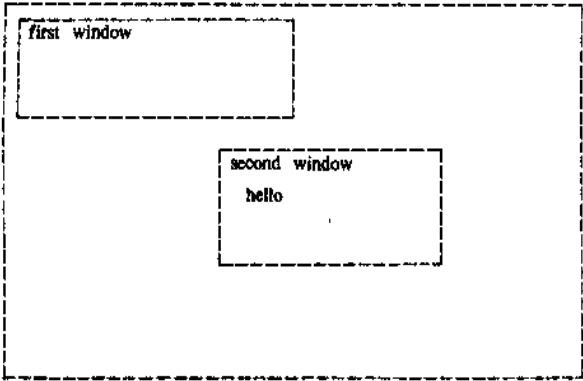
```
C>>type exp 11-1.c
#include "conio.h"
void border(int,int,int,int);
main()
{ clrscr();
  border(1,1,79,25);
  window(3,2,40,9);
  border(3,2,40,9);
  gotoxy(3,2);
  cprintf("first window");
  window(30,10,60,18);
  border(30,10,60,18);
  gotoxy(3,2);
  cprintf("second window");
  gotoxy(5,4);
  cprintf("hello");
  getch();
}
void border(int startx,int starty,int endx,int endy)
{ register int i;
  gotoxy(1,1);
  for(i=0;i<=endx-startx;i++)
    putch('--');
```

```

gotoxy(1, endy - starty);
for(i=0; i<=endx - startx; i++)
    putchar(' ');
for(i=2; i<endy - starty; i++)
{
    gotoxy(1, i);
    putchar(' ');
    gotoxy(endx - startx + 1, i);
    putchar(' ');
}
}

```

该程序执行后，屏幕上显示如下：



### 11.4 字符属性控制

所谓字符属性是指字符的颜色、亮度、闪烁，以及背景颜色等有关特征。为实现字符属性的控制，Turbl C2.0 设置了有关函数，如表 11.5 所示。

表 11.5 字符属性控制函数

函 数	功 能
highvideo()	将字符设置成高亮度
lowvideo()	将字符设置成低亮度
normvideo()	将字符设置成正常亮度
textattr()	同时设置字符和背景颜色
textbackground()	设置背景颜色
textcolor()	设置字符颜色

使用这类函数时，要注意以下几点：

①颜色符号及其所对应的数值如表 11.6 所示。

表 11.6 颜色符号及其所对应的数值

颜 色	符 号	数 值
黑	BLACK	0
蓝	BLUE	1
绿	GREEN	2
青	CYAN	3
红	RED	4
洋红	MAGENTA	5
棕	BROWN	6
淡灰	LIGHTGRAY	7
深灰	DARKGRAY	8
淡蓝	LIGHTBLUE	9
淡绿	LIGHTGREEN	10
淡青	LIGHTCYAN	11
淡红	LIGHTRED	12
淡洋红	LIGHTMAGENTA	13
黄	YELLOW	14
白	WHITE	15
闪烁	BLINK	128

②颜色的选择只影响到其下面要写的字符颜色或是背景颜色。

③函数 `textattr()` 可同时设置字符颜色、背景颜色以及闪烁与否。其原型如下：

```
void textattr(int attribute);
```

其参数 `attribute` 的 2 进制编码是：

7	6	5	4	3	2	1	0
闪烁	背景颜色						字符颜色

这意味着 `attribute` 的值等于字符颜色或背景颜色的数值 \* 16 的结果, 再或 `BLINK(128)`。例如, 下面的语句可使字符为红色且闪烁, 背景为蓝色。

```
textattr(RED|BLINK|BLUE * 16);
```

读者可从下面的练习看到这组函数各自的功能。

#### [练习 11.2]

```
c>type exp.11-2.c
#include <conio.h>
main()
{
    window(1,1,40,60);
    textbackground(1);
    clrscr();
    textcolor(YELLOW);
    gotoxy(20,2);
    lowvideo();
    cprintf("LOWVIDEO");
    gotoxy(15,4);
    normvideo();
    cprintf("Normvideo Normvideo");
    gotoxy(10,6);
```

```

highvideo();
cprintf("Higvideo Highvideo Highvideo");
gotoxy(20,8);
textattr(0x8e);
cprintf("Textattr")
getch();
}

```

## 11.5 字符屏显状态

返回屏幕状态的函数如表 11.7 所示。

表 11.7 返回屏显状态的函数

函 数	功 能
gettextinfo()	返回当前字符窗口信息
wherex()	返回光标处的 x 坐标
wherey()	返回光标处的 y 坐标

其中,函数 gettextinfo()的原型是:

```
void gettextinfo(struct text_info * info);
```

其参数是结构 text\_info 的指针;结构 text\_info 定义在文件 conio.h 中;该结构的定义如下所示。

```

struct text_info{
    unsigned char winleft;           /* left X coordinate */
    unsigned char wintop;            /* top Y coordinate */
    unsigned char winright;          /* right X coordinate */
    unsigned char winbottom;         /* bottom Y coordinate */
    unsigned char attribute;         /* text attribute */
    unsigned char normattr;          /* normal attribute */
    unsigned char curmode;            /*current video mode */
    unsigned char screenheight;      /*height of screen in lines */
    unsigned char screenwidth;       /* width of screen in chars */
    unsigned char curx;               /* cursor's X coordinate */
    unsigned char cury;               /* curxor's Y coordinate */
};

```

函数 gettextinfo()正确调用的例子如下所示。

```

struct text_info screen_status;
gettextinfo(&screen_status);

```

## 11.6 directvideo 变量

该变量用于控制屏幕输出的执行方式。当其值为 true 或缺省时,所有屏幕输出都通过屏显 RAM 直接进行,速度快;当其值为 false 时,屏幕输出要借助 BIOS 例行程序进行,而不用屏幕 RAM。

## 第十二章 绘 图

Turbo C 有很强的绘图功能,这些功能也是靠函数实现的。本章就介绍有关函数及其用法。

### 12.1 Turbo C 的绘图系统

(1)绘图系统的组成 该系统由两部分组成。一部分是和硬件无关的图形函数库 `graphics.lib`, 其文件名为 `graphics.h`。任何使用图形函数的 C 程序必须包含它,以便能使程序的目标文件与 `graphics.lib` 中的有关函数链接成可执行文件。另一部分是与具体适配器有关的图形设备驱动程序。该程序以文件形式存于磁盘中,其扩展名为 `BGI`(Borland Graphic Interface)。例如,适配器 EGA 的驱动程序其文件名为 `EGA.BGI`。

(2)绘图系统的初始化 要使用 Turbo C 绘图系统,就必须事先对其初始化。这包括分配内存,从磁盘调入当前适配器的驱动程序,设置图形模式等工作。初始化是通过函数 `initgraph()` 实现的。该函数的原型是:

```
void far initgraph(int far *driver,int far *mode,char far *path);
```

其中,参数 `driver` 用来指定所用适配器的驱动程序。各种适配器的驱动程序文件名及其所定义的数值如下所示。

文件名	数值
DETECT	0
CGA	1
MCGA	2
EGA	3
EGA64	4
EGAMONO	5
RESERVED	6
HERCMONO	7
ATT400	8
VGA	9
PC3270	10

如果选用 `DETECT`,函数 `initgraph()` 会自动选择当前系统的适配器驱动程序,且选用分辨率为最大的屏显模式。

参数 `path` 用来指明进入图形驱动程序的路径。如果参数 `path` 为空串,即 "",则在当前目录下寻找该驱动软件。

参数 `mode` 用来指定图形模式。Turbo C 支持的图形模式如表 12.1 所示。



表 12.1 Turbo C 的图形模式。

适配器	图形模式	模式值	分辨率
CGA	CGAC0	0	320×200
	CGAC1	1	320×200
	CGAC2	2	320×200
	CGAC3	3	640×200
MCGA	MCGAC0	0	320×200
	MCGAC1	1	320×200
	MCGAC2	2	320×200
	MCGAC3	3	320×200
	MCGAMED	4	640×200
	MCGAHI	5	640×480
EGA	EGALO	0	640×200
	EGAHI	1	640×350
EGA64	EGA64LO	0	640×200
	EGA64HI	1	640×350
EGAMONO	EGAMONOH	3	640×350
HERC	HERCMONOH	0	720×348
ATT400	ATT400C0	0	320×200
	ATT400C1	1	320×200
	ATT400C2	2	320×200
	ATT400C3	3	320×200
	ATT400CMED	4	640×200
	ATT400CHI	5	640×400
VGA	VGALO	0	640×200
	VGAMED	1	640×350
	VGAHI	2	640×480
PC3270	PC3270HI	0	720×350
IBM8514	IBM8514LO	0	640×480
	IBM8514HI	1	1024×760

这样,从图形驱动程序的寻找方式来看,绘图系统便有两种初始化方式,即:

①自动方式 这种方式适用于不知道屏显适配器的情况,具体程序如下:

```
#include "graphics.h"
main()
{
    int driver=DETECT,mode;
    initgraph(&driver,&mode,"");
    ;
}
```

②给定方式 这种方式适用于知道屏显适配器种类的情况。例如,已知系统所配置的适配器为 EGA,欲将屏幕置为高分辨率,就可用如下程序启动图形驱动程序。

```
#include "graphics.h"
main()
{ int driver,mode;
  driver=EGA;
  mode=EGAHI;
  initgraph(&driver,&mode,"");
}
```

```

        :
    }

```

(3)绘图系统的终止 当你要终止图形模式并返回到文本模式时,可使用如下两个函数。

①closegraph() 其功能是:

- 释放图形函数所占内存;
- 恢复屏显模式为调用 initgraph()之前的模式;
- 可继续在文本模式下运行程序。

②restorecrtmode()其功能是:

- 恢复屏显模式为调用 initgraph()之前的模式;
- 结束程序运行。

## 12.2 调色板和颜色的设置

(1)调色板及其大小 调色板由屏幕可同时显示的颜色构成,其大小就是可同时显示的颜色数。CGA 的四种图形模式就是给了四块调色板,每块调色板给了四种颜色,如表 12.2 所示。

表 12.2 CGA 调色板及其颜色值

颜色值 调色板	0	1	2	3
CGAC0	背景	绿	红	黄
CGAC1	背景	青	洋红	白
CGAC2	背景	淡绿	淡红	黄
CGAC3	背景	淡绿	淡洋红	白

在 EGA/VGA16 色图形模式中,一块调色板可有 16 种颜色。它们是从 64 种颜色中选择出来的。EGA/VGA 的调色板及其颜色如表 12.3 所示。

表 12.3 EGA/VGA 调色板及其颜色

颜色	符号名	值
黑	EGA_BLACK	0
蓝	EGA_BLUE	1
绿	EGA_GREEN	2
青	EGA_CYAN	3
红	EGA_RED	4
洋红	EGA_MAGENTA	5
棕	EGA_BROWN	20
浅灰	EGA_LIGHTGRAY	7
深灰	EGA_DARKGRAY	56
浅蓝	EGA_LIGHTBLUE	57
浅绿	EGA_LIGHTGREEN	58
浅青	EGA_LIGHTCYAN	59
浅红	EGA_LIGHTRED	60
浅洋红	EGA_LIGHTMAGENTA	61
黄	EGA_YELLOW	62
白	EGA_WHITE	63

采用调色板方式的好处是,一可使画面色彩丰富;二修改颜色方便,例如,在 EGAHI 模式中,

调色板第3项为 EGA\_BROWN(20),即所有值为3的象点的颜色都为棕色。如果把调色板的第3项改为 EGA\_YELLOW(62),则所有棕色点便都改为黄色。

(2)调色板颜色的设置,不论是设置调色板,还是改变调色板,都可以用函数实现。这些函数有:

①setpalette() 该函数原型是:

```
setpalette(int index,int color);
```

其功能是把颜色值 index 的值改为代表 color 色,如在 EGA 中,下面的语句设置使5代表青色。

```
setpalette(5,EGA_CYAN);
```

注意,对于 CGA,只能改变背景色,如下面的语句是把背景色改为绿色。

```
setpalette(0,GREEN);
```

②setbkcolor() 该函数原型是

```
setbkcolor(int color);
```

其功能是把背景色设为 color 色。color 值只能是表 12.4 中的 16 种之一。

表 12.4 16 种背景色

颜色值	颜色
0	黑(BLACK)
1	蓝(BLUE)
2	绿(GREEN)
3	青(CYAN)
4	红(RED)
5	洋红(MAGENTA)
6	棕(BROWN)
7	浅灰(LIGHTGRAY)
8	深灰(DARKGRAY)
9	浅蓝(LIGHTBLUE)
10	浅绿(LIGHTGREEN)
11	浅青(LIGHTCYAN)
12	浅红(LIGHTRED)
13	浅洋红(LIGHTMAGENTA)
14	黄(YELLOW)
15	白(WHITE)

③setallpalette() 该函数的原型是:

```
setallpalette(struct palettetype *p);
```

功能是整个设置一个 EGA/VGA 调色板。参数 p 是指向结构数据 palettetype 的指针。该结构定义如下:

```
struct palettetype
{ unsigned char size;
  signed char colors[16];
```

使用时,必须把 size 置为调色板的颜色数目,把 colors[16]的各元素初始化为相应的颜色值。例如,下面的程序是把 EGA/VGA 调色板改为其前 16 色。

```
struct palettetype *p;
int i;
for(i=0;i<16;i++)
    p.colors[i]=i;
p.size=16;
setallpalette (&p);
```

④setcolor()该函数的原型是:

```
setcolor(int color);
```

其功能是设置当前画线的颜色。

### 12.3 基本图形函数

(1)画线模式函数 其原型为:

```
setlinestyle(int style,unsigned pattern,int thickness);
```

功能是以其参数所指定的方式画线。其中,style 用来设置画线的模式,这些模式如表 12.5 所示。

表 12.5 线段类型

名 字	类 型	值
SOLID-LINE	直 线	0
DOTTED-LINE	虚 线	1
CENTER-LINE	中心线	2
DASHED-LINE	短横线	3
USERBIT-LINE	自定义	4

如果 style 取自定义的值 USERBIT-LINE,则线的模式由参数 pattern 中的 16 位数决定,即每一位对应线段的一个象点,为 1 的位,其对应象点以画线颜色画出。否则,pattern 取 0 值。

参数 thickness 确定线的宽度,有两种取值,如下所示。

```
NORM_WIDTH      1 个象素宽
THICK_WIDTH     3 个象素宽
```

(2)直线函数line() 其原型为:

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

功能是从点(x<sub>0</sub>,y<sub>0</sub>)到点(x<sub>1</sub>,y<sub>1</sub>)画一条直线。

〔练习 12.1〕 画一由四种类型线段组成的直线。

C>type eap12-1.c

```
#include "graphics.h"
main()
{
    int driver,mode;
    int i;
    driver=0;
    mode=0;
    initgraph(&driver,&mode,"");
    for(i=0;i<4;i++){
        setlinestyle(i,0,1);
        line(i*50,100,i*50+50,100);
    }
    getch();
    restorecrtmode();
}
```

(3)圆函数 circle() 其原型为:

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

功能是以点(x,y)为圆心,radius 为半径画圆。

(4)圆弧函数 arc() 其原型为:

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

功能是以点(x,y)为圆心,radius 为半径,从角 stangle 开始,反时针画弧到角 endangle。注意,角度的屏幕坐标如图 12.1 所示。

(5)椭圆函数 ellipse() 其原型为:

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

功能是以点(x,y)为圆心,xradius、yradius 分别为 x、y 方向上的半径,从角 stangle 开始画椭圆到角 endangle。

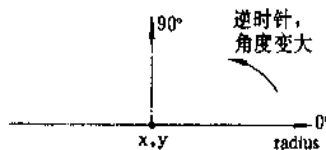


图 12.1 角度的屏幕坐标

(6)长方形函数 rectangle() 其原型为:

rectangle(int x<sub>1</sub>,int y<sub>1</sub>,int x<sub>2</sub>,int y<sub>2</sub>);

功能是以点(x<sub>1</sub>,y<sub>1</sub>)为右上角,点(x<sub>2</sub>,y<sub>2</sub>)为左下角画正方形。

(7)多边形函数 drawpoly() 其原型为:

drawpoly(int m,int \*p);

功能是以指针 p 所指数据对(x 坐标在前)为顶点坐标画一顶点数为 m 的多边形。

〔练习 12.2〕 画一由数组 s 定义的多边形。

```
C>type exp12-2.c
#include "graphics.h"
main()
{
    int driver,mode;
    int s[10]={ 10,10,
                100,80,
                200,200,
                350,90,
                10,10};
    driver=DETECT;mode=0;
    initgraph(&driver,&mode,"");
    drawpoly(5,s);
    getch();
}
```

## 12.4 填充模式函数

(1)填色模式函数 setfillstyle() 其原型为:

setfillstyle(int pattern,int color);

功能是为各种图形设置填充模式 pattern 和颜色 color。各种模式及其值如表 12.6 所示。

表 12.6 pattern 及其值

名 称	值	含 义
EMPTY-FILL	0	用背景色填充
SOLID-FILL	1	实填充
LINE-FILL	2	用线“-”填充
LTSLASH-FILL	3	用斜杠填充
SLASH-FILL	4	用粗斜杠填充
BKSLASH-FILL	5	用粗反斜杠填充
LTBKSLASH-FILL	6	用反斜杠填充
HATCH-FILL	7	用网格线填充
XHATCH-FILL	8	用斜网格线填充
INTERLEAVE-FILL	9	用间隔点填充
WIDE_DOT-FILL	10	用稀疏点填充
CLOSE_DOT-FILL	11	用密集点填充
USER-FILL	12	自定义模式

(2)填充图形函数 floodfill() 其原型为:

floodfill(int x,int y,int border);

功能是以区域内任意一点(x,y)的模式和颜色填充以 border 颜色为边界的有界区域。若点(x,y)不在有界区域内,则对整个屏幕处理。

〔练习 12.3〕 用交叉阴影线和浅红色填充一椭圆。

```
C>type exp12_3.c
#include "graphics.h"
main()
{
    int driver,mode;
    driver=DETECT;
    mode=0;
    initgraph(&driver,&mode,"");
    ellipse(100,100,0,360,80,40);
    setfillstyle(XHATCH_FILL,LIGHTRED);
    floodfill(100,100,WHITE);
    getch();
}
```

(3)模式设置函数 setfillpattern() 其原型为:

setfillpattern(char \*upattern,int color);

功能是当函数 setfillstyle()函数的参数 pattern 取 USER\_FILL 时,用其指针参数 upattern 所指向的 8 个字节数据组成的 8×8 点阵和 color 指定的颜色确定填充模式。

〔练习 12.4〕

```
C>type exp12_4.c
#include "graphics.h"
main()
{
    int driver,mode;
    char p[8]={10,20,30,40,50,60,70,80};
    driver=DETECT;
    mode=0;
    initgraph(&driver,&mode,"");
    setcolor(GREEN);
    rectangle(100,200,200,300);
    setfillpattern(p,RED);
    floodfill(150,250,GREEN);
    getch();
}
```

## 12.5 着色图形函数

(1)直方图函数 bar() 其原型为:

bar(int x<sub>1</sub>,int y<sub>1</sub>,int x<sub>2</sub>,int y<sub>2</sub>);

功能是以点(x<sub>1</sub>,y<sub>1</sub>)为右上角,点(x<sub>2</sub>,y<sub>2</sub>)为左下角,按当前颜色着色画一方图。

(2)立方图函数 bar3d() 其原型为:

bar3d(int x<sub>1</sub>,int y<sub>1</sub>,int x<sub>2</sub>,int y<sub>2</sub>,int dep,int top);

功能是以点 $(x_1, y_1)$ 为右上角,点 $(x_2, y_2)$ 为左下角,dep 为厚度,按当前颜色着色画一立方图,参数top 非零,有顶盖否则无顶盖。

(3)多边形函数 fillpoly() 其原型为:

fillpoly(int point, int \* p);

功能同函数 drawpoly()一样画多边形且按当前颜色着色。

(4)扇形函数 pieslice() 其原型为:

pieslice(int x, int y, int stangle, int endangle, int radius);

功能是以点 $(x, y)$ 为圆心, radius 为半径, stangle 角度为起点, endangle 角度为终点, 当前填充模式和颜色画一扇形。

〔练习 12.5〕 画一每  $45^\circ$  为一种颜色的完整圆。

```
C>type exp12-5.c
#include "graphics.h"
main()
{
    int driver, mode;
    struct palettetype p;
    int i, sta, end;
    driver = DETECT;
    mode = 0;
    initgraph(&driver, &mode, "");
    sta = 0; end = 45;
    for(i=0; i<8; i++)
    {
        setfillstyle(SOLID_FILL, i);
        pieslice(300, 200, sta, end, 100);
        sta += 45;
        end += 45;
    }
    getch();
    restorecrtmode();
}
```

〔练习 12.6〕 综合练习:画一踢足球的人。

程序如下:



```

#include <graphics.h>
main()
{
    int driver, mode, i, j;
    driver=DETECT;
    initgraph (& driver, & mode, " ");
    circle (315, 20, 40);
    line (255, 37, 375, 37);
    line (255, 37, 255, 107);
    line (255, 107, 375, 107);
    line (375, 107, 375, 37);
    bar (375, 37, 495, 57);
    bar (145, 37, 255, 57);
    line (315, 107, 255, 137);
    line (255, 137, 255, 177);
    line (255, 177, 195, 177);
    line (195, 177, 195, 137);
    line (195, 137, 255, 107);
    line (315, 107, 300, 147);
    line (300, 147, 320, 187);
    line (320, 187, 380, 187);
    line (380, 187, 360, 147);
    line (360, 147, 375, 107);
    circle (260, 177, 40);
}

```

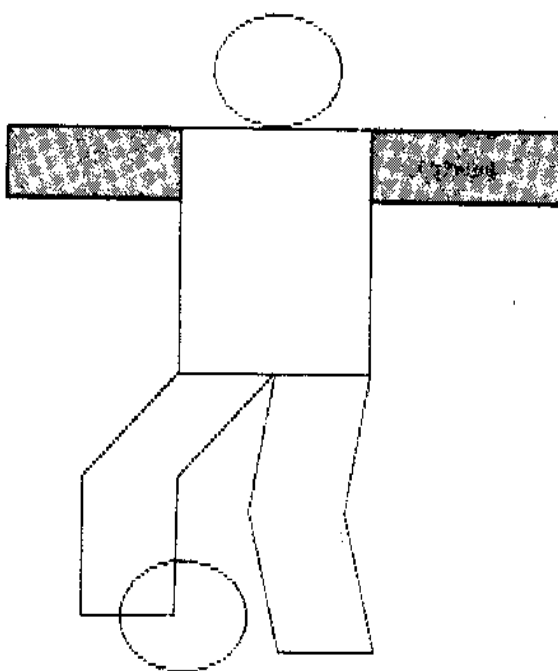


图 12.2 练习 12.6 执行打印结果

## 12.6 图形模式下的汉字输出

目前,国内流行的 Turbo C 2.0 为未汉化的编译程序,使用它如何解决汉字共面是大家所关心的问题之一。我们可以用前面所介绍的绘图函数,绘制矢量字,其方法请大家参考清华大学出版社出版的《IBM PC C 语言例题 习题 库函数》一书。还有一个很便当的办法。这就是在图形模式下使用 gotoxy() 函数定位,用 printf() 函数打印汉字。不过,由于编译系统未汉化,这就必须在中文 DOS 下编辑 C 源文件,再用 Turbo C 编译,如练习 12.7 所示。

### 〔练习 12.7〕

```

E>type exp13-14.c
#include <stdio.h>
#include <conio.h>
#include <graphics.h>
main()
{
    void * p;
    int gd=DETECT,gm;
    initgraph(&gd,&gm,"");

```

```

p="输出汉字";
printf("%s",p);
gotoxy(25,10);
printf("%s",p);
gotoxy(50,20);
puts(p);
getch();
restorecrtmode();
}

```

执行结果如下:

E>exp 12.7

输出汉字

输出汉字

输出汉字

使用上述方法是能够解决大家在项目中所遇到的有关实际问题的。

## 12.7 图形屏幕管理函数

(1)清屏函数 cleardevice() 其原型是:

```
cleardevice();
```

功能是清除屏幕,并把光标位置重新设置在(0,0)处。

(2)存储图形函数 getimage() 其原形是:

```
getimage(int left,int top,int right,int bottom, void * buf);
```

功能是把左上角坐标为(left,top),右下角坐标为(right,bottom)的部分屏幕图形拷贝到由 buf 所指向的内存区域。

(3)重显图象函数 putimage() 其原型是:

```
putimage(int x,int y,void * buf,int op);
```

功能是把原存储在由 buf 所指向的内存中的图形,以 op 所指定的方式显示在以(x,y)为起点的屏幕上。op 值及其含义如表 12.7 所示。

表 12.7 图象重显方式

op 值	名字	含义
0	COPY PUT	复制
1	XOR PUT	与屏幕图象异或
2	OR PUT	与屏幕图象或
3	AND PUT	与屏幕图象与
4	NOT PUT	拷贝源图象的非

(4)计算存储图形所需字节数函数 imagesize() 其原型是:

```
imagesize(int left,int top, int right,int bottom);
```

功能是给出存储左上角为(left,top),右下角为(right,bottom)的一块图形所需要的字节数。

〔练习 12.8〕 用实例说明 `getimage()`、`imagesize()` 和 `putimage()` 的用法。

```
C>type expl2_8.c
#include "graphics.h"
void box(int,int,int,int,int);
main()
{
    int driver,mode;
    int size;
    void *buf;
    driver=DETECT;
    mode=0;
    initgraph(&driver,&mode,"");
    box(20,20,200,200,15);
    setcolor(RED);
    line(20,20,200,200);
    setcolor(GREEN);
    line(20,20,200,20);
    getch();
    size=imagesize(20,20,200,200);
    if(size!==-1)
    { buf=malloc(size);
      if(buf)
      { getimage(20,20,200,200,buf);
        putimage(100,100,buf,COPY_PUT);
        putimage(300,50,buf,COPY_PUT);
      }
    }
    outtext("press a key");
    getch();
    restorecrtmode();
}

void box(int startx,int starty,int endx,int endy,int color)
{ setcolor(color);
  rectangle(startx,starty,endx,endy);
}
```

(5)指定图形输出活动页函数 `setactivepage()` 其原型是:

`setactivepage(int page);`

功能是图形函数的输出放在参数 `page` 所指定的页上。如:

`setactivepage(2);`

就指定图形函数的输出放到 2 页上。注意只有 EGA 和 VGA 的某些模式才支持多图形页。

(6)指定可见图形页数函数 `setvisualpage()` 其原型是:

```
setvisualpage(int page);
```

功能是指定在屏幕上显示的为 page 页。如：

```
setvisualpage(1);
```

指定显示 1 页。

(7)设置当前图形输出视区函数 `setviewport()` 其原型是：

```
setviewport(int left,int top,int right,int bottom,int clip);
```

功能是以座标(left,top)为左上角、座标(right,bottom)为右下角建立新的图形视区。当 clip 为 1 时,超出视口的输出自动剪掉;当 clip 为 0 时,超出视口的输出不做裁剪。

(8)清除当前视口函数 `clearviewport()` 其原型是：

```
clearviewport(void);
```

功能是清除当前视口,并把光标置为(0,0)。

## 12.8 图形信息管理函数

(1)存储当前视口信息函数 `getviewsettings()` 其原型是：

```
getviewsettings(struct viewporttype *info);
```

其中结构 `viewporttype` 在 `graphics.h` 中定义,如下所示：

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

其中,(left,top)为左上角;(right,bottom)为右下角,clipflag 为超出视口部分裁剪标志,即其值为 1,要裁剪掉,否则保留。

(2)存储图形文字信息函数 `gettextsettings()` 其原型是：

```
gettextsettings(struct textsettingstype *info);
```

结构 `textsettingstype` 定义在 `graphics.h` 文件中,如下所示：

```
struct textsettingstype
{ int font;
  int direction;
  int charsize;
  int horiz;
  int vert;
}
```

其中,成员 `font` 定义在 `graphics.h` 文件中,如前所述;成员 `direction` 表示文字方向:水平文字为 `HORIZ_DIR`(缺省值),垂直文字为 `VERT_DIR`;成员 `charsize` 表示字符尺寸的系数;horiz 和 vert 指明文字与当前位置(cp)的关系,取值如表 12.8 所示。

表 12.8 表示文字位置的宏

值	宏名	含义
0	LEFT-TEXT	CP 在左边
1	CENTER-TEXT	CP 在中心
2	RIGHT-TEXT	CP 在右边
3	BOTTOM-TEXT	CP 在底部
4	TOP-TEXT	CP 在顶部

该函数的用法如下所示：

```
struct textsettingstype t;
gettextsettings(&t);
```

(3)返回指定象素颜色函数 `getpixel()` 其原型是：

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

功能是返回点(x,y)的颜色。

〔练习 12.9〕 综合练习：该程序利用视口显示菜单，供用户选择。其菜单如下：

- 1——三维图
- 2——二维图
- 3——退出

并根据用户的输入，取得各方图的坐标，完成统计功能。

```
C>type b:opp.c
#include <graphics.h>
#include "stdio.h"
#define posx(i) (i * 50)
#define posy(j) (j * 20)
void box(int,int,int,int,int);
main()
{
    float s[5];
    void *e,*f;
    int driver,mode,i,j,c,d,x1,y1,x2,y2,v,x,h,a,b,t;
    driver=DETECT;
    initgraph(&driver,&mode,"");
    for(x=0;x<1000;x++)
    {
        box(0,0,639,349,1);
        setviewport(20,20,200,200,1);
        box(100,50,460,160,4);
        outtextxy(140,70,"Now please choose.");
        outtextxy(140,80,"Input v=1,2,3.");
        outtextxy(140,90,"1;cubic;2:rectangle;3:quit.");
        outtextxy(140,100,"Press keyboard.");
        outtextxy(140,150,"please input.");
```

```

gotoxy(160,150);
getch();
clearviewport();
for(i=0;i<=4;i++)
{
    printf("Now input the period value=");
    scanf("%f",&s[i]);
    if(i>0)
        s[i]=s[i]/s[0]*4.0;
}
s[0]=4;
printf("please input v=");
scanf("%d",&v);
for(h=0;h<25;h++)
    printf("\n");
switch(v)
{
    case 1:
        line(100,180,1000,180);
        line(100,180,100,3);
        c=0;f=&c;d=0;e=&d;
        for(i=2;i<=12;i++)
        {
            itoa(c,e,10);
            outtextxy(posx(i),184,e);
            line(posx(i),180,posx(i),175);
            c++;
        };
        d=1;
        for(j=8;j>=1;j--)
        {
            itoa(d,f,10);
            outtextxy(90,posy(j),f);
            line(100,posy(j),105,posy(j));
            d++;
        };
        for(i=0,i<=4;i++)
        {
            d=turn(s[i]);
            a=50*d;b=20*d;c=a-50;
            bar3b(a,b,c,180,20,1);
        }
        getch();
        clearviewport();
        break;
    case 2:

```

```

    line(100,180,1000,180);
    line(100,180,100,3);
    c=0;f=&c;d=0;e=&d;
    for(i=2;i<=12;i++)
    { itoa(c,e,10);
      outtextxy(posx(i),184,e);
      line(posx(i),180,posx(i),175);
      c++;
    };
    d=1;
    for(j=-8;j>=1;j++)
    { itoa(d,f,10);
      outtextxy(90,posy(j),f);
      line(100,posy(j),105,posy(j));
      d++;
    };
    for(i=0;i<=4;i--)
    {
      d=turn(i);
      a=50*d;b=10*d;c=a-50;
      rectangle(a,b,c,180);
    }
    getch();
    clearviewport();
    break;
default:
    printf("It's default.\n");
    goto close;
};
};
close :getch();
clearviewport();
}
void box (int sx,int sy,int ex,int ey,int color)
{
    setcolor(color);
    line(sx,sy,sx,ey);
    line(sx,sy,ex,sy);
    line(ex,sy,ex,ey);
    line(ex,ey,sx,ey);
}
turn(ll)
float ll;
{

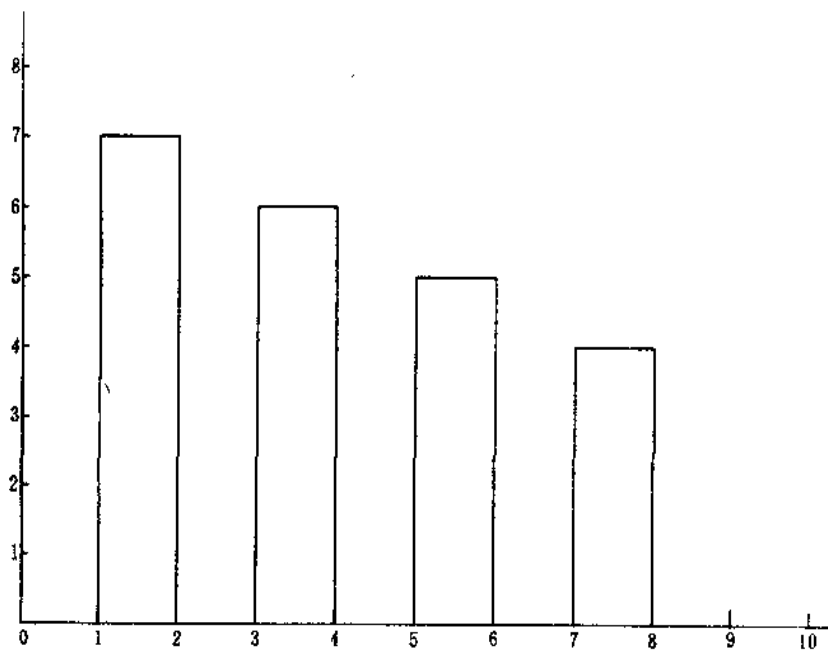
```

```

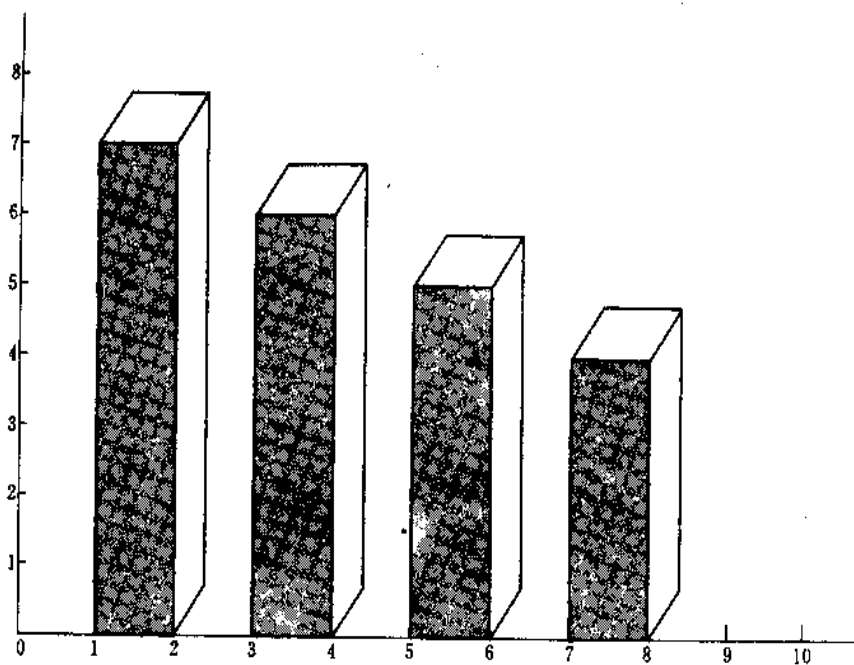
float pp;int qq;
pp=11;
for(;ll>=1;ll--)
if(ll>=0.5) qq=pp-ll+1;
else qq=pp-ll;
return(qq);
}

```

该程序执行结果如图 12.3 所示。



(a) 直方图



(b) 立方图

图 12.3 练习 12.8 的打印结果



## 第十三章 C 与汇编语言的混合编程

本章介绍 C 语言与汇编语言的混合编程方法。

### 13.1 C 调用汇编语言子程序

这里研究的是 Turbo C 如何调用汇编语言子程序(过程)。这就是探讨被调用的汇编语言子程序的格式、C 程序如何把实参传递给汇编语言子程序、汇编语言子程序如何把返回值返回给 C 程序诸问题。

(1) 汇编语言子程序的格式 要想了解 Turbo C 对被调用的汇编语言子程序所要求的格式,只要使用带选择项 -S 的命令行 Turbo C 编译程序 Tcc,编译一个最小 C 程序(一个空函数)即可,如练习 13.1 所示。

〔练习 13.1〕 首先建立一个只含有空函数 sub() 的 c 源文件 exp13\_1.c,然后用如下命令:

Tcc -S exp13\_1.c

编译该文件,即得到该文件所对应的汇编语言程序。

```
E>type exp13_1.c
sub()
{
}

E>tcc -S exp13_1.c
Turbo C Version 2.0 Copyright (c) 1987,1988 Borland International
exp13_1.c:
    Available memory 400272
E>
        ifndef      ?? version
? debug    macro
        endm
        endif
        ? debug    S    "exp13_1.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 E9E24962180965787031335F342E63
_BSS      ends
_TEXT     segment    byte public 'CODE'
;         ? debug    L 1
_sub      proc       near
@1:
;         ? debug    L 3
          ret
_sub      endp
_TEXT     ends
          ? debug    C E9
_DATA     segment word public 'DATA'
s@        label      byte
_DATA     ends
_TEXT     segment    byte public 'CODE'
_TEXT     ends
          public     _sub
          end

```

从该程序可以看到汇编语言程序模块有如下特点:

①有程序模块定义,即:

```
? debug S "exp13 1.c"
```

汇编语言程序模块

end

汇编语言源程序一般是用两条伪操作指令定义程序模块的,即

```
name 程序模块名
```

汇编语言程序模块

end (标号)

其中 name 是用来定义程序模块名的。该指令可以缺省。缺省时,若模块中使用了 TITLE 语句(列表输出的页标题指令),则页标题就是模块名;若未使用 TITLE 语句,该程序的源文件名就是程序模块名。

一个模块是一个独立的汇编单位。end 是汇编结束指令,即汇编程序汇编该程序到 end 指令为止,即使其后有语句也不进行汇编。由于这里的模块不是主模块,无需指明其执行起始地址,故 end 后面不用跟随执行起始地址。

②程序模块分段定义 一个模块中分有代码段、数据段、附加数据段和堆栈段,这样可以把程序模块根据逻辑上的段的定义分别装入由段寄存器 CS、DS、ES 和 SS 所指定的不同的物理段中;

一个物理段占 64K 字节。这样,使程序便于管理。

段的定义格式如下:

段名 segment [定位类型] [连接方式] ['类别']

段模块

段名 ends

其中段名是人们任意设定的。

定位类型表示段的起始地址要求,有 page、para、word 和 byte 四种类型,起始地址分别是

page:   ××××   ××××   ××××   0000       0000B  
para:   ××××   ××××   ××××   ××××   0000B  
word:   ××××   ××××   ××××   ××××   ×××0B  
byte:   ××××   ××××   ××××   ××××   ××××B

分别表示以页、段、字、字节为起始地址。若此项缺省,则隐含为 para。

连接方式用来告诉链接程序:本段与其它段的关系,有 NONE、PUBLIC、COMMON、AT 表达式、STACK 和 MEMORY 6 种方式,其含义分别是:

NONE 表示本段与其它段在逻辑上没有关系,每段都有自己的基址。这是隐含方式。

PUBLIC 链接程序把本段与同名同类别的其它段连接成一个段。

STACK 表示此段为堆栈段。其连接方式与 PUBLIC 相同,连接时将所有 STACK 方式的名段连接成一个段。程序中必须至少有一个 STACK 段,否则需要用户用指令初始化 SS 和 SP;若有多,初始化时,SS 指向第一个 STACK 段。

COMMON 链接程序为本段和同名同类别的其它段指定相同的基址,因而本段将与同名同类别的其它段相覆盖。段的长度为最长的 COMMON 段的长度。

AT 表达式 链接程序把本段装在表达式的值所指定的段地址上(偏移量按 0 处理)。

MEMORY 链接程序把本段定位在其它所有段之上(即地址较大的区域)。若有多个 MEMORY 段,则第一个按 MEMORY 方式处理,其余均按 COMMON 方式处理。

类别名可由用户任意设定。链接程序把类别名相同的段(未必段名相同)放在连续的存储区内,但仍为不同的段(连接方式为 public、common 的段除外)。

③不同名的段可集合成为一组 用伪操作命令 group 实现,其用法如下:

组名 group 段名,段名,……

如程序中出现的:

DGROUP group \_DATA, \_BSS

其含义就是把 \_BSS 段和 \_DATA 段合并成组名为 DGROUP 的一组。把若干段定义为一组,可以使这些段都装在同一个物理段中,得到较紧凑的代码,这样,组内各段之间的跳转就可以看作是段内跳转了。

④把 C 语言函数定义为汇编语言的过程 汇编语言中的过程其定义格式如下:

过程名 proc near/far

过程体

ret

过程名 endp

这里的进程名就是原 C 函数名。

near/far 是用来表示过程的跳转或调用类型属性的。near 表明过程为段内跳转或调用,称为近调用(跳转);far 表明过程为段间跳转或调用,称为远调用(跳转)。跳转或调用类型是 near 还是 far 与 Turbo C 的存储模式有关。Turbo C 有 6 种存储模式,各种存储模式所对应的跳转或调用类型如表 13.1 所示。

表 13.1 Turbo C 2.0 存储模式与跳转/调用类型

存储模式	段寄存器内容	类型
Tiny(最小模式)	CS=DS=SS=ES	near
Small(小模式)	CS! =DS DS=SS=ES	near
Compact(紧凑模式)	CS! =DS! =SS! =ES	near
Medium(中模式)	CS! =DS! =SS! =ES	far
Large(大模式)	CS! =DS! =SS! =ES	far
Huge 特大模式	CS! =DS! =SS! =ES	far

ret 为返回指令,用于实现从过程返回到调用的程序。注意,一个过程可允许使用多个 ret 指令;而且就是使用一个 ret 指令也不是必须放在过程体的最后。

(2)C 程序和汇编语言子程序之间的参数传递 这个问题就是介绍 Turbo C 是如何把实参从 C 程序传递给汇编语言子程序的。调用函数时,C 语言是通过堆栈(堆栈是在内存中开辟的一个专门用来存放数据的区或,有关问题将在 16 章讨论)进行参数传递的,C 程序调用汇编语言子程序其参数传递过程也是如此。了解这一过程的最好方法还是用 Tcc 把有调用关系的 C 程序编译成汇编语言程序,然后分析参数的传递过程。如练习 13.2。

〔练习 13.2〕 用 Tcc -S 对文件 exp13\_2.c 进行编译。exp13\_2.c 内容如下:

```
C>type exp13_2.c
```

```
int su;
main()
{ su=sub(5,2);
}
sub(int a,int b)
{ int x;
  x=a-b;
  return(x);
}
```

编译命令及其信息如下:

```
C>tcc -S exp13_2.c
```

```
Turbo C Version 2.0 Copyright(c) 1987,1988 Borland International
exp13_2.c
```

```
Available memory 408548
```

C 语言源文件经 Tcc -S 命令编译后变为汇编语言文件,这里便产生 exp13\_5.asm 文件,其内容如下:

```
C>type exp13_2.asm
```

```
ifndef ?? version
```

```

? debug    macro
           endm
           endif
           ? debug S "exp13_2.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 E9536F70180965787031335F352E63
_BSS       ends
_TEXT      segment byte public 'CODE'
;          ? debug L 2
_main      proc    near
;          ? debug L 3
           mov     ax, 2
           push    ax
           mov     ax, 5
           push    ax
           call    near ptr _sub
           pop     cx
           pop     cx
           mov     word ptr DGROUP, _su, ax
@1;
;          ? debug L 4
           ret
_main      endp
;          ? debug L 5
_sub       proc    near
           push    bp
           mov     bp, sp
           push    si
;          ? debug L 7
           mov     si, word ptr [bp+4]
           sub     si, word ptr [bp+6]
;          ? debug L 8
           mov     ax, si
           jmp     short @2

```

```

@2:
;      ? debug L 9
      pop     si
      pop     bp
      ret
-sub   endp
-TEXT ends
-BSS   segment word public 'BSS'
-su    label   word
      db      2 dup (?)
-BSS   ends
      ? debug C E9
-Data segment word public 'DATA'
s@     label   byte
-Data ends
-TEXT segment byte public 'CODE'
-TEXT ends
      public  _main
      public  _su
      public  _sub
      end

```

从文件 exp13\_2.asm 可以看出:

①C 语言函数被编译为汇编语言的过程 即主函数 main()变为过程 \_main;函数 sub()变为过程 \_sub。注意,用 Tcc 编译后所有标识符和变量名前面都有下划线\_,所以,我们在编写由 C 程序调用的汇编语言子程序时,函数名和变量名前面也都要加下划线。

②参数的传递过程 调用函数(这里就是主函数 main())把被调用函数的实参从右向左,重复使用如下两条指令逐一压栈(即先压入最末一个参数)。

```
mov ax,参数或参数地址
```

```
push ax
```

各种类型数据在栈区所占的字节数如表 13.2 所示。

表 13.2 各种类型数据在栈区所占字节数

数据类型	所占字节数
char	2
signed char	2
unsigned char	2
short	2
signed short	2
unsigned short	2
int	2
signed int	2
unsigned int	2
long	4
unsigned long	4
float	4
double	8
(near)pointer	2(仅偏移量)
(far)pointer	4(段地址和偏移量)
数组	2(按指向数组的指针)

被调用函数(即过程 sub)被执行时用 mov 或其他指令再从栈区取出实参使用。这里首先用指令:

```
mov si,word ptr [bp+4]
```

从栈区取出第一参数的值(实参 5),传送到 si 寄存器中,然后用指令:

```
sub si,word ptr [bp+6]
```

从第一参数值中减去从栈中取出的第二参数的值(实参 2)。至此,程序既完成了实参由调用函数到被调用函数(过程)的传递,又实现了被调用函数的功能。

ptr 是类型运算符,用来定义变量、标号或地址表达式所指定的操作数的类型,也可以使它们临时具有与原定义不同的类型,但并不改变操作数的段地址和偏移量。其用法是:

```
类型 ptr 地址表达式
```

其中类型对于变量来说,可以是 byte(字节型)、word(字型)或 dword(双字型);对于标号来说,可以是 near 或 far。地址表达式是由运算符连接变量、标号、常量,以及[BP]、[BX]、[DI]、[SI]所组成的值为存储器地址的式子,其特例是变量、标号或[BP]、[BX]、[DI]、[SI]。

(3)汇编语言子程序的返回值 汇编语言子程序如有返回值,根据其类型,应按表 13.3 存放。

表 13.3 返回值所使用的寄存器

数据类型	存放的寄存器
char	AX
unsigned char	AX
short	AX
unsigned short	AX
int	AX
unsigned int	AX
long	AX(低位字) DX(高位字)
unsigned long	AX(低位字) DX(高位字)
float	AX(低位字) DX(高位字)
double	8087 栈或仿真程序的 TOS 上
struct/union	值的地址
near pointer	AX
far pointer	AX(偏移量) DX(段地址)

返回值的处理过程即传递过程是这样的:

①在汇编语言子程序(过程)中,把要返回给 C 程序的返回值,用 mov 指令传送到表 13.3 所指定的寄存器中。

②在 C 程序的调用函数中,再用 mov 指令从该寄存器中取出,传递给指定的变量。

(4)C 程序调用汇编语言子程序(过程)的过程 通过分析练习 13.2,可以清楚地了解到其调用过程:

①把要传递给汇编语言子程序的实参按自右向左的顺序压栈 该程序中被调用的汇编语言子程序的实参为 5 和 2,故实参的压栈顺序为:

```

mov ax,2
push ax
move ax,5
push ax

```

这样,压栈操作后,栈区的情况如图 13.1 所示。

②压栈保存 C 程序的断点 这是为执行完汇编语言子程序后返回时使用。由于每条即将执行的指令的地址偏移量都是存放在指令指针寄存器 IP 中,其段地址存放在代码段寄存器 CS 中,这里的断点也不例外,只是这里为近调用,段地址不变,故压栈的数据只有 IP 的内容。这时,栈区情况变为如图 13.2 所示。

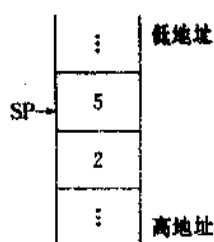


图 13.1

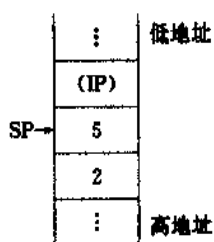


图 13.2

③把汇编语言子程序首地址的偏移量和段地址分别送入寄存器 IP 和 CS 这里为近调用,故只把汇编语言子程序首地址的偏移量送入 IP 即可,于是,便转去执行汇编语言子程序。

④执行汇编语言子程序 这里要做四件事:

• 保护寄存器现场 这就是将汇编语言子程序将要使用的寄存器内容压栈。Turbo C 的调用协议允许子程序(过程)使用寄存器 SI 和 DI,用来放局部变量。即使局部变量不是寄存器类别,Turbo C 的编译程序在其优化部分会自动将它们变为寄存器存储类别。如果寄存器变量多于两个,则多余部分会自动转到栈中存储。调用协议还规定,子程序(过程)执行时,栈指针 SP 的当前值要传送到基址指针寄存器 BP 中,以便使用寄存器 BP 的间接寻址,从栈中读取实参,供子程序使用。这样,子程序在一般的情况下应有如下指令:

```

push bp
push di
push si

```

由于该汇编语言子程序中只有一个局部变量 x,仅用寄存器 SI 存放其值,就足够了。因此,在汇编语言子程序的开始部分有如下三条指令,用来保护 SI 和 BP 的内容,以及使用 BP 指向栈区某单元。

```

push bp
mov bp,sp
push si

```

这时栈区情况如图 13.3 所示。

• 读取实参并进行功能操作 保护断点和有关寄存器内容统称为保护现场。保护现场后汇编语言子程序就可以读取 C 程序传递给它的实参并进行功能操作了。根据此时的栈区情况,使用指令:

```

mov si,word ptr [bp+4]

```



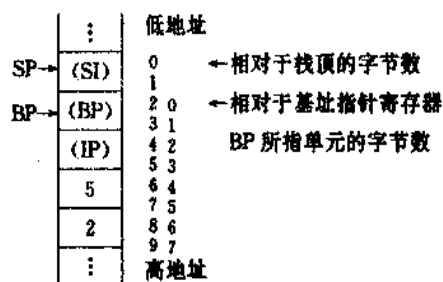


图 13.3

便可以从栈区相对于 BP 内容为 4 个字节的栈单元中取出第一个参数值(实参 5)存在 si 中。下条指令就是从 si 中减去相对于 BP 内容为 6 个字节的单元中取出的第二参数(实参 2),结果(即该子程序的返回值)存在 si 中,即:

```
sub si,word ptr [bp+6]
```

• 保存返回值 返回值要根据表 13.3 传送到相应的寄存器中保存,以便返回到调用的函数。这里,由于返回值为 int 型,使用如下指令把返回值存入 AX。

```
mov ax,si
```

• 恢复寄存器现场 在汇编语言子程序返回之前,要恢复寄存器现场,指令如下所示:

```
pop si
pop bp
```

这时,栈区的情况如图 13.4 所示:

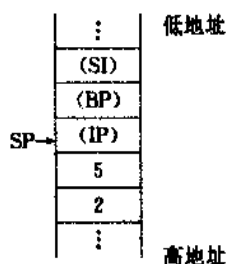


图 13.4

⑤返回 C 程序 在子程序的最后用 ret 指令恢复 CS 和 IP 的原内容。这里为近调用,亦即近返回,故只恢复 IP 的值。IP 放上原断点的偏移量,程序立即返回到 C 程序原断点处去执行。这里,ret 指令执行后,栈区的情况如图 13.5 所示。

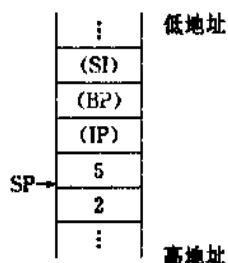


图 13.5

⑥恢复栈区原状态 返回到 C 程序后,还要把栈区恢复到原状态,这里,使用两条 POP CX 指令实现。这时,栈区的情况如图 13.6 所示。

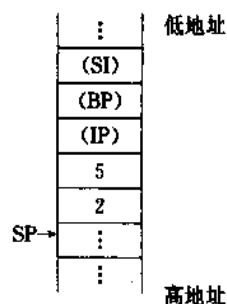


图 13.6

⑦把子程序的返回值传送给 C 程序 在该程序中,是把汇编语言子程序放在 AX 中的返回值传递给 C 程序的外部存储类别全局变量 su。指令如下:

```
mov word ptr DGROUP:_su,ax
```

### (5)C 程序调用汇编语言子程序的实现步骤

这里以 C 程序调用除法汇编语言子程序为例来说明,如练习 13. 所示。

〔练习 13. 3〕 C 程序调用除法汇编语言子程序的实现步骤有如下五步:

①编辑 C 程序及汇编语言子程序。

• C 程序如下:

```
E>type exp13_3.c
#include <stdio.h>
main()
{
    printf("%d\n",divide(10,2));
}
```

• 汇编语言子程序如下:

```
E>type af.asm
name divide
_text segment byte public 'code'
dgroup group _bss, _data
    assume cs:_text,ds:dgroup,ss:dgroup
_text ends
_data segment word public 'data'
_d@ label byte
_data ends
_bss segment word public 'bss'
_b@ label byte
_bss ends
_text segment byte public 'code'
_divide proc near
    push bp
    mov bp,sp
    mov ax,[bp+4]
    cwd
```

```

@1: idiv word ptr[bp+6]
    pop bp
    ret
_divide endp
_text ends
_data segment word public 'data'
_s@ label byte
_data ends
_text segment byte public 'code'
    public divide
_text ends
end

```

②用带-c 选择项的 Tcc 编译 C 源文件,使其变为目标文件,命令如下:

```

E>tcc -linclude -c exp13-3.c
Turbo C Version 2.0 Copyright(c) 1987,1988 Borland International
exp13-3.c

```

Available memory 388324

该命令执行结果使文件 exp13-3.c 变为 exp13-3.obj 文件。

③用宏汇编程序 MASM 编译汇编语言源文件,命令如下:

```

E>masm af.asm
Turbo Assembler Version 1.0 Copyright(c) 1988 by Borland International
Assembling file: AF.ASM
Error messages: None
warning messages: None
Remaining memory: 446K

```

编译结果使文件 af.asm 变为 af.obj 文件。

④链接目标文件和库文件。库文件因存储模式不同而有所区别,如表 13.4 所示。

表 13.4 各存储模式所对应的库文件

存储模式	启动库文件	浮点运算库文件	运行库文件
极小模式	c0t.obj	maths.lib	cs.lib
小模式	c0s.obj	maths.lib	cs.lib
紧凑模式	c0c.obj	maths.lib	cc.lib
中模式	c0m.obj	maths.lib	cm.lib
大模式	c0l.obj	maths.lib	cl.lib
特大模式	c0h.obj	maths.lib	ch.lib

链接程序可用 DOS 的 link,也可用 Turbo C 的 Tlink。使用 Tlink 时,如系统拥有 8087 协处理器,则需文件 FP87.LIB;如需仿真 8087,则需文件 EMU.LIB。以小模式为例,命令如下:

```

E>tlink lib\c0s exp13-3 af,cf,.lib\cs
Turbo Link Version 2.0 Copyright (c) 1987,1988 Borland International

```

链接后得到执行文件 cf.exe。

⑤运行执行文件,命令和执行结果如下:

E>cf

5

### 13.2 汇编语言程序调用 C 函数

(1)调用 C 函数的汇编语言程序的编写方法 要使一个汇编语言程序能调用 C 函数,该程序就要包括如下内容。

①使用如下格式说明被调用的 C 函数及其调用类型属性:

extrn \_C 函数名 调用类型

注意,原被调用的 C 函数名前要加下划线符。调用类型用 near 或 far 来表示,用 near 还是 far 取决于存储模式,仍是按表 13.1 处理。

②函数实参压栈,注意:

- 一定要按从右至左的顺序进行;
- 当参数为常量时,使用两条指令操作:

```
mov ax,常量  
push ax
```

- 当参数为变量时,使用一条指令操作:

```
push 变量
```

- 当参数为地址时,也使用两条指令操作:

```
lea ax,地址参数  
push ax
```

③调用 C 函数,指令如下:

```
call _C 函数名
```

④清栈,使用如下指令:

```
add sp,n
```

其中 n 为参数所占的字节数。

⑤传送结果到指定变量,使用如下指令:

```
mov 变量,ax  
mov 变量+2,dx
```

若传送的数据其大小为一个字,则只使用第一条指令即可。

(2)汇编语言程序调用 C 函数的实现步骤 可按如下顺序进行。

- ①编辑 C 函数;
- ②按(1)所述方法编写调用该 C 函数的汇编语言程序;
- ③用带 -c 选择项的 Tcc 编译该 C 函数的源文件,使其变为目标文件;
- ④用宏汇编 MASM(或 TASM)编译由步骤②所编写的汇编语言程序,也使其变为目标文件;
- ⑤用 link 链接由步骤③和④所产生的目标文件,生成执行文件;
- ⑥运行执行文件。

#### 〔练习 13.4〕

①编写一个从两个整数中选取大者的 C 函数,程序如下:

E>type max.c

```

max(int a,int y)
{
    if(x>y)
        return(x)
    else
        return(y)
}

```

②编辑调用①中所编写的 C 函数 max() 的汇编语言程序,如下所示。

E>type exp13\_4.asm

```

.data segment word public 'data'
y    dw 0
.data ends

.text segment byte public 'code'
    assume cs:_data,ds:_data
    extrn max,near
    mov ax,3
    push ax
    mov ax,2
    push ax
    call _max
    add sp,4
    add ax,0
    daa
    mov y,ax,
    mov dx,y
    or di,30h
    mov ah,2
    int 21h
    mov ax,4c00h
    int 21h
.text ends
end

```

③用带 -c 选择项的 Tcc 编译上述 C 函数所在的文件,命令和编译信息如下:

E>tcc -c max.c

Turbo C Version 2.0 Copyright(c) 1987,1988 Borland International

max.c:

Available memory 399010

④用 MASM(TASM)编译程序汇编调用上述 C 函数的汇编语言源程序,命令和编译信息如下:

E>tasm exp 13\_4.asm

Turbo Assembler Version 1.0 (c) 1988 by Borland International

Assembling file: EXP13\_4.ASM

Error messages: None

Warning messages: None

Remaining memory: 447K

⑤用 link (Tlink) 链接③和④所生成的目标文件, 命令和链接信息如下:

E>link

The COMPAQ Personal Computer Linker

Version 2.40 (C)Copyright Compaq Computer Corporation 1982,1986

(C)Copyright Microsoft Corp. 1981,1986

Object Modules [ .OBJ ]: exp13\_4 + max

Run File [ EXP13\_4 . EXE ]:

List File [ NUL . MAP ]:

Libraries [ . LIB ]:

Warning: no stack segment

⑥运行⑤中生成的执行文件, 命令和运行结果如下:

E>exp13\_4

3

E>

### 13.3 C 程序中插入指令行

Turbo C 也允许 C 程序中直接插入指令行。这叫行间汇编 (in-line assemble)。这是一种简便易行的混合编程方法。这里就介绍这种用法。

#### (1) 编程方法

①汇编行前要使用关键字 asm 加以说明。使用时, 一般一行一条指令, 这时指令末尾的分号可省略。一行中也可以有多条指令, 但每条指令前都要有 asm, 且指令之间要用分号隔开。

②行间汇编适用于所有 8086/8088 指令

③指令的操作数可直接引用 C 变量、参数、结构成员, 甚至宏名, 跳转指令可直接使用 C 标号。

④行间汇编指令的注释要用 C 语言的注释方法, 即把注释用 /\* \*/ 括起来。

#### (2) 实现步骤

①编辑含有行间汇编的 C 源文件;

②用带有 -B 选择项的 Tcc 编译①步所编写的 C 程序, 使其变为目标文件。

若在文件的开头使用 #pragma inline, 预先通知 C 编译程序; C 程序中使用了行间汇编, 则可以使用带有 -c 选择项的 Tcc, 把含有行间汇编的 C 源文件编译为目标文件。

③用 link (或 Tlink) 链接目标文件和库文件, 得到执行文件。

④运行执行文件, 得到执行结果。

上述操作步骤如练习 13.5 所示。

〔练习 13.5〕该程序的功能和练习 13.3 完全相同, 只是把原汇编子程序改为用行间汇编编写的 C 函数。程序和执行结果如下所示。

E>type exp13\_5.c

#include <stdio.h>

main()

```

{
    int a,b;
    a=10;
    b=2;
    printf("A/B=%d/%d=%d\n",a,b,div(10,2));
}

```

div(int x,int y)

```

{
    int num;
    asm mov ax,x
    asm mov bx,y
    asm cwd
    asm idiv bx
    asm mov num,ax
    return(num);
}

```

E>tcc --linclude -B --exp13\_5 exp13\_5.c

Turbo C Version 2.0 Copyright(c) 1987 1988 Borland International

exp13\_5.c;

Turbo Assembler Version 1.0 Copyright(c) 1988 by Borland International

Assembling file: EXP13\_5.ASM

Error messages: None

warning messages: None

Remaining memory: 295K

Turbo Link Version 2.0 Copyright(c) 1987,1988 Borland International

Available memory 387742

E>exp13\_5

A/B=10/2=5

程序中使用宏指令 #pragma inline 后,可用 Tcc -c 编译,如练习 13.6 所示。

### 〔练习 13.6〕

E>type exp13\_6.c

#include <stdio.h>

main()

```

{
    int a,b;
    a=10;
    b=2;
    printf("A/B=%d/%d=%d\n",a,b,div(10,2));
}

```

div(int x,int y)

```

{
    #pragma inline
    int num;
    asm mov ax,x

```

```

asm mov bx,y
asm cwd
asm idiv bx
asm mov num,ax
return(num);
}
E>tcc -linclude -c exp13_6.c
Turbo C Version 2.0 Copyright(c) 1987 1988 Borland International
exp13_6.c:
Turbo Assembler Version 1.0 Copyright(c) 1988 by Borland International
Assembling file: EXP13_6.ASM
Error messages: None
warning messages: None
Remaining memory: 295K
Available memory 372924
E>tlink lib\c0s exp13_6,exp13_6,lib\cs
Turbo Link Version 2.0 Copyright(c) 1987,1988 Borland International
E>exp13_6
A/B=10/2=5

```

(3)注意事项 到这里,我们已经了解了C语言和汇编语言的三种混合编程方法。前两种方法的基本思路是,C模块和汇编模块分别用各自的编译程序编译成目标码,然后在目标码级别上进行链接,得到可执行文件。而在使用行间汇编这种混合编程方法中,是硬要汇编程序去识别C语言所设定的变量、宏,这当然就会带来麻烦。现把使用这种方法时的主要问题提醒给大家注意。

①要正确使用汇编指令,特别提醒大家注意指令的源操作数和目标操作数的类型。譬如ADD指令和MOV指令,它们只能进行寄存器与寄存器、寄存器与存储器之间的的运算和把立即数加(传送)到寄存器或存储器的运算;但不能进行两个存储单元之间的数据运算。因此,如果x和y是用C语句所定义的变量,则指令mov x,y和add x,y对宏汇编来说都是不允许的,请看练习13.7。

#### [练习 13.7]

```

D>type exp13_7.c
main()
{ int y,x=1;
  asm mov y,x;
  printf("%d\n",y);
}

```

该程序编译情况如下:

```

D>Tcc -B exp13_7.c
Turbo C version 2.0 copyright (c) 1987 1988 Borland International
exp13_7.c
Microsoft (R)Macro Assembler Version 5.00
Copyright(C) Microsoft Corp 1981-1985,1987 All rights reserved
exp13_7.ASM(28);error A2052 ;Improper operand type
exp13_7.ASM(36);error A2006;phase error between passes

```



51604+213724 Bytes symbol space free

0 Warning Errors

2 Severe Errors

Available memory 391912

解决这类问题的办法有两个：一是把源操作数和目标操作数其中的一个定义为寄存器存储类别变量，如练习 13.8 所示。二是借助累加器 AX，如练习 13.9 所示。

### 〔练习 13.8〕

D>type exp13\_8.c

main()

{ int x=1;

register int y;

asm mov y,x;asm mov x,word ptr 2

printf("%d\t%d\n",x,y);

}

D>Tcc -B exp13\_8.c

Turbo C version 2.0 copyright(c) 1987 1988 Borland International

exp13\_8.c

Microsoft (R) Macro Assembler Version 5.00

Copyright (C) Microsoft Corp 1981-1985,1987 All rights reserved

51614+213714 Bytes symbol space free

0 Warning Errors

0 Severe Errors

D>Tlink \kc\lib\c0s exp 13\_8,li,,\tc\lib \cs

Turbo link Version 2.0 Copyright(c) 1987,1988 Borland International

D>li

2 1

### 〔练习 13.9〕

E>type exp13\_9.C

struct s{

int a;

int b;

int c;

}ma;

main()

{

ma.a=1

asm mov ax,ma.a

asm mov ma.b,ax

asm mov ma.c,ax

printf("%d\t%d\t%d\n",ma.a,ma.b,ma.c);

}

E>tcc -B -eexp13\_9 exp13\_9.c

Turbo C Version 2.0 Copyright(c) 1987 1988 Borland International

```
exp13_9.c:
Turbo Assembler Version 1.0 Copyright(c) 1988 by Borland International
Assembling file: EXP13_9.ASM
Error messages:    None
warning messages:  None
Remaining memory: 260K
Turbo Link Version 2.0 Copyright(c) 1987,1988 Borland International
    Available memory 398342
```

```
E>exp13_9
```

```
1  1  1
```

```
E>
```

行间汇编指令使用到 C 定义的宏时,也需注意这类问题,如练习 13.10 所示。

### 〔练习 13.10〕

```
E>type exp13_10.c
```

```
#define EXP 0xf0
main()
{
    int exp1=2;
    asm mov ax,exp1
    asm add ax,EXP
    asm mov exp1,ax
    printf("%d\n",exp1);
}
```

```
E>tcc -linclude -B -eexp13_10 exp13_10.c
```

```
Turbo C Version 2.0 Copyright(c) 1987,1988 Borland International
```

```
exp13_8.c:
Turbo Assembler Version 1.0 Copyright(c) 1988 by Borland International
Assembling file:    EXP13_10.ASM
Error messages:    None
warning messages:  None
Remaining memory: 261K
Turbo Link Version 2.0 Copyright(c) 1987,1988 Borland International
    Available memory 399034
```

```
E>exp13_10
```

```
242
```

如果把该练习中的三条行间汇编指令合并为一条:asm add exp1,EXP,虽然也能得出正确结果,但编译时出现了警告信息,如下所示。

```
Assembling file:    EXP13.10.ASM
* Warning * EXP13_10.ASM(31) Argument needs type override
Error messages:    None
Warning messages:  1
Remaining memory: 261K
```

②行间汇编指令虽然可直接使用 C 变量,但却不能用 C 的方式对其赋值。譬如:

• 行间汇编指令使用 C 定义的整型变量时,要说明其大小。变量大小为 8 位,用 byte ptr 说明;16 位用 word ptr 说明;大于 16 位可用两条 mov 指令实现;对 les、lea 指令要使用 dword ptr。如练习 13.11 所示。

#### 〔练习 13.11〕

```
E>type exp13_11.c
#include <stdio.h>
main()
{
    int var1,var2;
    asm mov var1,word ptr 3
    asm push var1
    asm pop var2
    printf("%d\t%d\n",var1,var2);
}
E>tcc -linclude -B -eexp13_11 exp13_11.c
Turbo C Version 2.0 Copyright(c) 1987,1988 Borland International
exp13_11.c:
Turbo Assembler Version 1.0 Copyright(c) 1988 by Borland International
Assembling file:   EXP13_11.ASM
Error messages:    None
Warning messages:  None
Remaining memory: 261K
Turbo Link Version 2.0 Copyright (c) 1987,1988 Borland International
    Available memory 389152
E>exp13_11
3    3
```

• C 定义的浮点型变量,不能简单地使用 mov 指令赋值。因为汇编程序并不能这样简单地处理浮点数。练习 13.12 就说明了这个问题。

#### 〔练习 13.12〕

```
E>type exp13_12.c
#include <stdio.h>
main()
{
    int var1;
    float var2;
    asm mov var1,word ptr 3
    asm mov var2,3.0
    printf("%d\t%d\n",var1,var2);
}
E>tcc -linclude -B -eexp13_12 exp13_12.c
Turbo C Version 2.0 Copyright(c) 1987,1988 Borland International
exp13_12.c:
```

Turbo Assembler Version 1.0 Copyright(c) 1988 by Borland International

Assembling file: EXP13-12.ASM

\* \* Error \* \* EXP13-12.ASM(33) Illegal number

Error messages: 1

warning messages: None

Remaining memory: 260K

Available memory 388554

③行间汇编可以使用跳转和循环指令,但只能在本函数体内跳转或循环,如练习 13.13 所示。

### [练习 13.13]

E>type exp 13-13.c

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

```
main()
```

```
{  
    char *s;  
    asm jmp c  
    b: x="jump d,";  
    goto d;  
    c: printf("jump c,\n");  
    asm jnz b  
    d: printf("%s\n",s);  
}
```

E>tcc -B -cexp13-13 exp13-13.c

Turbo C Version 2.0 Copyright(c) 1987,1988 Borland International

exp13-13.c;

Turbo Assembler Version 1.0 Copyright(c) 1988 by Borland International

Assembling file: EXP13-13.ASM

Error messages: None

warning messages: None

Remaining memory: 260K

Turbo Link 2.0 Copyright(c) 1987,1988 Borland International

Available memory 388936

E>exp 13-13

```
jump c;
```

```
jump d;
```

E>

## 第十四章 PC 机硬件资源管理

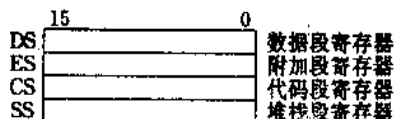
计算机系统资源包括硬件资源和软件资源。硬件资源,一般指 CPU、内存、I/O 设备(包括外存)。软件资源,一般指操作系统、实用程序、编译系统和专用程序。本章在介绍 PC 机硬件资源的基础上,探讨 C 语言管理硬件资源的四种方法。

### 14.1 PC 硬件资源

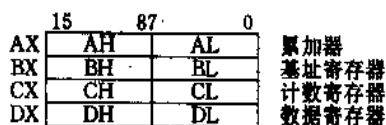
#### (1) 8088、8086、80286 CPU 寄存器

这三种 CPU,寄存器相同,各有 14 个 16 位寄存器,它们分别是:

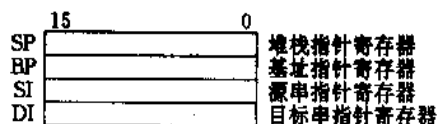
①段寄存器 有 DS、ES、CS、和 SS 四个,即:



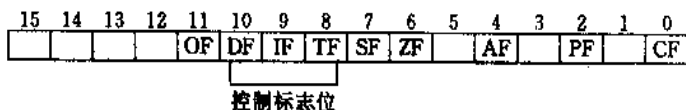
②数据寄存器 有 AX、BX、CX 和 DX 四个,即:



③指针寄存器 有 SP、BP、SI 和 DI、四个,即:



④状态标志寄存器 这亦是一个 16 位寄存器,各位标志如下所示:



其中:

- CF 是进位标志位:若当前运算结果的最高位产生进位或借位,则 CF 为 1;否则为 0;
- PF 是奇偶标志位:若当前运算结果 1 的个数为偶数,则 PF 为 1;否则为 0;
- AF 是辅助进位标志位:若当前的运算低 4 位产生进位或借位,则 AF 为 1;否则为 0;
- ZF 是零标志位:若当前运算结果为 0,则 ZF 为 1;否则为 0;
- SF 是符号标志位:若当前运算结果为负数,则 SF 为 1;否则为 0;
- TF 是单步工作标志位:若将其置位(TF 为 1),则 CPU 处于单步工作方式;若将其复位(TF 为 0),则 CPU 将正常工作。
- IF 是开中断标志位:若将其置位,则 CPU 响应外设的中断请求;若将其复位,CPU 关中断;
- DF 是方向标志位:若其值为 1,则数据串按递减顺序处理;若其值为 0,则按递增顺序处理;
- OF 是溢出标志位:若当前运算结果产生算术溢出,则 OF 为 1;否则为 0。

⑤指令指针寄存器 IP 上述四类寄存器都可用指令直接进行存取操作,而该寄存器却不能用指令访问。其功能是,存放当前指令的偏移量。若当前指令的逻辑地址为 CS:IP,则其物理地址就是  $(CS) \times 16 + (IP)$ 。

## (2)80386 CPU 寄存器

它有 10 个 32 位寄存器和 6 个 16 位寄存器,即:

①数据寄存器 有 EAX、EBX、ECX 和 EDX 四个 32 位寄存器,即

	31	16	15	0	
EAX	AX				扩展累加器
EBX	BX				扩展基址寄存器
ECX	CX				扩展计数寄存器
EDX	DX				扩展数据寄存器

②指针寄存器 有 ESP、EBP、ESI 和 EDI 四个 32 位寄存器,即:

	31	16	15	0	
ESP	SP				堆栈指针寄存器
EBP	BP				基址指针寄存器
ESI	SI				源串指针寄存器
EDI	DI				目标串指针寄存器

③段寄存器 有 CS、SS、ES、DS、FS 和 GS 六个 16 位寄存器,即:

	15	0	
CS			代码段寄存器
SS			堆栈段寄存器
ES			附加段寄存器
DS			数据段寄存器(1)
FS			数据段寄存器(2)
GS			数据段寄存器(3)

④状态标志寄存器 是 32 位寄存器,各标志位如下所示:

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	NT	IO	PL	OF	DF	IF	TF	SF	ZF		AF		PF		CF	

EFLAGS	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
															VM	RF

其中,0~15 位与 8088 状态标志寄存器完全兼容,只扩展了三个标志位,即:

- NT 为插入作业标志位;
- IO 为输入/输出标志位;
- PL 为特权标志位;

16~31 位为 80386 扩展的状态标志,只有两位有用,即:

- VM 为虚拟 8088 模式标志位;
- RF 为继续标志位;

⑤指令指针寄存器 亦是 32 位寄存器,即:

	31	16	15	0
EIP	IP			

## (3)系统内存分配

PC/XT、PC/AT 以及 PS/2 的内存映象,如图 14.1 所示。

000000	系统板 RAM(640K)	因型号大小可以从 64K 到 640K
09FFFF 0A0000	显示适配器 RAM(128K)	EGA 和 VHA 全部使用, CGA 和 MDA 部分使用
0BFFFF 0C0000	I/O 口 ROM(128K)	<ul style="list-style-type: none"> <li>• C0000—C3FFF EGA BIOS</li> <li>• C6000—C63FF PGA 通讯区</li> <li>• C8000—CBFFF 硬盘 BIOS</li> <li>• D0000—D7FFF 群集控制器 BIOS</li> <li>• D0000—DFFFF PCjr 扩展盒式磁带机</li> </ul>
0DFFFF 0E0000	系统保留 ROM(64K)	在 AT 和 PS/2 中用于标准盒式磁带机
0EFFFF 0F0000	系统 ROM(64K)	高存储区 ROM 的复本
0FFFFF 100000	系统板 RAM(384K)	仅模式 50、60 和 80 使用
15FFFF 160000	扩充 RAM(14.5K)	仅 AT 和 PS/2 使用
FDFFFF FE0000	系统保留 ROM(64K)	仅 AT 和 PS/2 使用
FEFFFF FF0000	ROM BIOS(64K)	仅 AT 和 PS/2 使用
FFFFFF		

图 14.1 内存映象

#### (4) I/O 端口地址

PC/XT、PC/AT 以及 PS/2 的 I/O 端口地址如表 14.1 所示。

表 14.1 I/O 端口地址

地址范围	XT 机	AT 机	PS/2	实际地址
00—0FH	DMA 控制器 (8237A—5)	DMA 控制器(8237A—5)	DMA 控制器	
10—1FH		为系统板保留	DMA 控制器	
20—2FH	中断控制器 (8259A)	中断控制器(8259A)	中断控制器 (8259A)	仅用 20、21H
30—3FH		中断控制器(8259A)		
40—4FH	定时器(8253—5)	定时器(8254—2)	系统定时器	XT 仅用 40—43H, PS/2 仅用 40、42—44、47H
50—5FH	定时器(8253—5)	定时器(8254—2)		
60—6FH	并行接口 (8255A—5)	键盘(8042)	键盘	XT 仅用 60—63H, PS/2 仅用 60、61、64H
70—7FH		定时计数器、NMI 中断	定时计数器 NMI	PS/2 仅用 70、71H, 保留 74—76H
80—8FH	DMA 页寄存器	DMA 页寄存器(74LS612)	DMA 页寄存器	XT 仅用 80—83H, AT 和 PS/2 仅用 81—83、87、89—8B、8FH
90—9FH	DMA 页寄存器	DMA 页寄存器	I/O 通道	PS/2 仅用 90—94、96、97H
A0—AFH	NMI 中断	中断控制器 2(8259A)	中断控制器 (8259)	PS/2 仅用 A0、A1H
B0—BFH		中断控制器 2(8259A)		
C0—CFH	保留	DMA 控制器 2(8237A—5)	DMA 控制器	
D0—DFH		DMA 控制器 2(8237A—5)	DMA 控制器	

续表

地址范围	XT 机	AT 机	PS/2	实际地址
E0—EFH	保留	为系统板保留		
F0—FFH		协处理器(80287)	协处理器(80X87)	AT 仅用 F0、F1、F8—FFH
100—10FH		用于 I/O 通道	可编程选择	PS/2 仅用 100—107H
110—1EFH		用于 I/O 通道		
1F0—1FFH		硬盘		
200—20FH	游戏卡 I/O 口	游戏卡 I/O 口		游戏卡仅用 200—207H
210—21FH	扩展单元	21F 保留		XT 仅用 210—217H
220—24FH	保留	用于 I/O 通道		
250—25FH		用于 I/O 通道		
260—26FH		用于 I/O 通道		
270—27FH	并行打印机 2	并行打印机 2	并行口 3	除 PS/2 仅用 278—27BH 外,其余全部使用 278—27FH
280—28FH		用于 I/O 通道		
290—29FH		用于 I/O 通道		
2A0—2AFH		用于 I/O 通道		
2B0—BFH	交替 EGA	交替 EGA		
2C0—2CFH	交替 EGA	交替 EGA		
2D0—2DFH	交替 EGA (也用 3270)	交替 EGA		
2E0—2EFH		GPIO,数据采集口 0		AT 仅用 2E1、2E2、2E3H
2F0—2FFH	第 2 异步适配器	串行口 2	串行口 2 (RS232C)	全部仅用 2F8—2FFH
300—30FH	原型插件板	原型插件板		
310—31FH	原型插件板	原型插件板		
320—32FH	硬盘适配器	用于 I/O 通道		
330—33FH		用于 I/O 通道		
340—34FH		用于 I/O 通道		
350—35FH		用于 I/O 通道		
360—36FH		PC 网(低地址)		
370—37FH	并行打印机	并行打印机 1	并行口 2	除 PS/2 仅用 378—37B 外,其余使用 378—37FH
380—38FH	SDlc 或第 2 双工同步控制器	SDlc 或第 2 双工同步控制器		
390—39FH		群集控制适配器		
3A0—3AFH	第 1 双工同步控制器	第 1 双工同步控制器		
3B0—3BFH	MDA	MDA	视频子系统, 并行口 1	
3C0—3CFH	EGA	EGA	视频子系统	
3D0—3DFH	CGA	CGA	视频子系统	
3E0—3EFH	3E0—3E7 保留	用于 I/O 通道		
3F0—3FFH	软盘适配器, 第 1 异步串行口	软盘适配器, 第 1 异步串行口	磁盘驱动控制 器,串行口 1	磁盘用 3F0—3F7H,异步通信用 3F8—3FFH

#### 14.2 使用指令直接访问硬件

C 语言管理硬件的能力很强,可以说,使用 C 语言,可以管理 PC 机的全部硬件。从本节开始,介绍 C 语言管理 PC 机硬件的四种方法,即:

- 调用汇编语言子程序或插入指令行,使用指令直接访问硬件;
- 利用管理硬件的库函数,访问相应的硬件;
- 使用函数 `int86()` 和 `int86x()`,执行指定的 ROM 中的 BIOS 软中断,访问相应硬件;
- 使用库函数 `bdos()`,执行指定的 DOS 系统功能调用,访问相应硬件。



本节先介绍第一种方法,即调用汇编语言子程序或插入指令行,使用指令直接访问硬件的方法。要使用这种方法,用指令直接控制硬件,就要熟悉 PC 机的指令、硬件接口以及内存分布。下面,通过实例来说明。

〔练习 14.1〕 调用特定汇编语言子程序,使 PC 机音响系统演奏歌曲。

(1)PC 机音响系统的硬件组成 PC 机音响系统其硬件是由并行接口 8255 的 B 口、D 触发器 74LS175、8253—5 定时器 2、与门 74LS08、功率放大器 75477 和喇叭构成的,如图 14.2 所示。

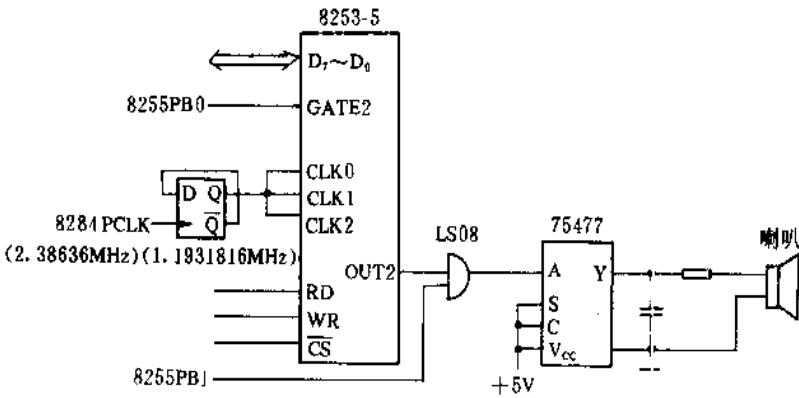


图 14.2 PC 机音响系统

(2)产生乐曲原理

① 8253 时钟信号的产生 来自时钟发生器 8284 的时钟信号 PCLK (2.3863635MHz),经 74LS175 二分频后,变为定时器 8253—5 的时钟信号 (1.1931817MHz)。

② 音符频率 通过给定时器 8253 写入控制字,确定定时器 2 以方式 3 工作。方式 3 为分频工作方式,即定时器 2 的输出端输出方波,其频率为:

$$f_{out} = \frac{1.1931817 \times 10^6}{TC} \quad (1)$$

其中,TC 为给定时间常数。

各音符所对应的频率高低,如表 14.2 所示。

表 14.2 音符频率表

音符	1	2	3	4	5	6	7	1	2	3	4	5	6	7	1
频率	131	147	165	175	196	220	247	262	294	330	349	392	440	494	523

利用公式(1)可以计算出各音符所对应的 TC 值,用 OUT 指令把 TC 值写进 8253。在方式 3 下,当门控信号 GATE2 为高电平时,即可在 OUT2 端产生相应频率的信号。该信号通过 75477 放大器放大,即可驱动喇叭发出相应音符的音响。门控信号 GATE2 的高电平是由并行接口芯片 8255 的 PB0 提供的。

③ 音符节拍 各音符除了音调的高低外,还有一个发音时间长短的问题,即乐曲的节拍。各音符发音时间的长短是由与门 LS08 控制的。其控制端 PB1 为高电平时,该门打开;为低电平时,该门关闭。例如,《两只老虎》的曲子:

1=C 4/4

1	2	3	1	1	2	3	1	3	4	5	—	3	4	5	—
56	54	3	1	56	54	3	1	2	5	1	—	2	5	1	—

其节拍是 4/4,即每小节为 4 拍。就是全音音符为 4 拍,2 分音符为 2 拍,4 分音符为 1 拍,8 分音符为半拍。若把全音音符定为 1 秒,则各种音符所对应的时间,如表 14.3 所示。

表 14.3 各音符所需时间

音符	全音音符	2 分音符	4 分音符	8 分音符
时间	1000ms	500ms	250ms	125ms

我们知道,PC 机系统时钟的频率为 4.77MHz,即其周期为 210ns。又知,loop 指令在跳转时需要 17 个时钟周期,顺序执行时需要 5 个时钟周期;mov 指令在把立即数传送到寄存器时需要 4 个时钟周期。故用下面两条指令可获得 10ms 的延时。

```
MOV CX,2801
DL10ms; LOOP DL 10ms
```

有了这个延时子程序,便可用重复该子程序的执行来获得各音符的发音时间。若把重复该子程序的次数作为各音符发音时间的数据,则表 14.3 就变成表 14.4。

表 14.4 各音符时间数据

音符	全音音符	2 分音符	4 分音符	8 分音符
时间数据	100	50	25	13

从给 PB1 提供高电平打开与门 74LS08 起,开始计时,到相应时间用给 PB1 提供低电平,关闭 74LS08,即可达到发音时间的控制。8255 的 B 端口地址为 61H,故 PB0、PB1 的电平输出可用如下指令实现。

```
OUT 61H,AL
```

(3)实现程序 用主函数调用乐曲汇编语言子程序,即可产生乐曲。练习 14.1 是产生《两只老虎》乐曲的程序。

#### 〔练习 14.1〕

##### ①汇编语言子程序:

```
C:\TC15\TU>type to_tiger.asm
```

```
name to_tiger
stack segment
    dw 100 dup(?)
stack ends
data segment
    bg db 0ah,0dh,"two tiger:$"
    freq dw 2 dup(262,294,330,262)
    dw 2 dup(330,349,392)
```

```

        dw 2 dup(392,440,392,349,330,262)
        dw 2 dup(294,196,262),0
time dw 10 dup(25),50,25,25,50
        dw 2 dup(13,13,13,13,25,25)
        dw 2 dup (25,25,50)
data ends
code segment
        assume cs:code,ds:data
start    proc far
        push ds
        mov ax,0
        push ax
        mov ax,data
        mov dx,ax
        mov dx,offset bg
        mov ah ,9
        int 21h
        mov si,offset freq
        mov bp,offset time
        call sing
        ret
start    endp
sing     proc near
        push di
        push si
        push bp
        push bx
pp:      mov di,[si]
        cmp di,0
        je end_sing
        mov bx ,ds:[bp]
        call sound
        add si,2
        add bp,2
        jmp pp
end_sing: pop bx
        pop bp
        pop si
        pop di
        ret
sing     endp
sound    proc near
        push  ax
        push  bx

```

```

        push    cx
        push    dx
        push    di
        mov     al,0b6h
        out     43h,al
        mov     dx,12h
        mov     ax,34dch
        div     di
        out     42h,al
        in      al,61h
        mov     ah,al
        or      al,3
        out     61h,al
delay:   mov     cx,2801
dl10ms:  loop    dl10ms
        dec     bx
        jnz     delay
        mov     al,ah
        out     61h,al
        po      di
        pop     dx
        pop     cx
        pop     bx
        pop     ax
        ret
sound    endp
code     ends
        end star

```

②主文件,即含有主函数 main()的 C 文件,如下所示。

C:\TC15\TU>type exp14-1.C

```
#include "stdio.h"
```

```
main()
```

```
{
    to_tiger();
}
```

用插入汇编指令行的方法实现练习 14.1 的功能的程序,如练习 14.2 所示。

#### [练习 14.2]

C:\TC15\TU>type exp14-2.c

```
main()
```

```
{ int *pf, *pt;
  int tc;
  int u,v;
```

```

int freq[]={262,294,330,262,262,294,330,262,330,349,392,330,349,392,392,440,392,349,330,262,
            292,440,392,249,330,262,294,196,262,294,196,262,0};
int time[]={25,25,25,25,25,25,25,25,25,25,50,25,25,50,13,13,13,13,25,25,13,13,13,13,25,25,
            25,25,50,25,25,50};

    pf=freq;
    pt=time;
    u=*pf;
    v=*pt;
sound:  asm mov    di,u
        asm mov    bx,v
        asm mov    al,0b6h
        asm out    43h,al
        asm mov    dx,12h
        asm mov    ax,34dch
        asm div    di
        asm out    42h,al
        asm in     al,61h
        asm mov    ah,al
        asm or     al,3
        asm out    61h,al
delay:  asm mov    cx,2801
dl10ms: asm loop   dl10ms
        asm dec    bx
        asm jnz    delay
        asm mov    al,ah
        asm out    61h,al
        pf++;
        pt++;
        u=*pf;
        v=*pt;
        if(*pf!=0)
            goto sound;
}

```

### 14.3 利用库函数管理相应硬件

本节介绍 C 语言管理 PC 机硬件的第二种方法,即利用 C 编译系统所带有的库函数管理相应硬件。有关函数包括内存管理函数、寄存器管理函数和 I/O 端口管理函数。下面分别介绍它们的用法和功能。

#### (1) 内存动态分配函数

Turbo C 定义的内存动态分配函数如表 14.5 所示。

表 14.5 内存动态分配函数

类 别	函 数
分配存储空间	void * calloc(unsigned num,size)
	void * malloc(unsigned size)
改变存储空间	void * realloc(void * ptr,unsigned size)
归还存储空间	void * free(void * ptr)

ANSI 标准指出,这些函数都返回 void 指针。void 指针具有一般性,可以指向任何类型的数据。有些 C 编译系统定义其返回一个 char 指针。但注意,无论是哪种情况,当把它们赋给其它类型指针时,都必须进行显式类型转换。

这些函数是从位于程序内存常驻区和栈之间的自由存储区(称为堆)中分配空间。之所以叫动态分配是指需要时分配,不需要时再归还给系统。

①函数 calloc() 该函数原型在 stdlib.h 中定义,其类型和参数定义如下:

```
void * calloc(num,size)
unsigned int num;
unsigned int size;
```

功能:在堆中为 num 个大小为 size 的数据分配足够的存储空间。

返回:返回指向所分配的自由空间首字节的指针。若空间不够,则返回空指针。

②函数 free() 该函数原型也在 stdlib.h 中定义,其类型和参数定义如下:

```
void free(void * ptr);
```

功能:该函数释放由指针 ptr 所指定的内存空间,并把这部分空间返回给堆,以便重新分配时使用。

返回:无返回值。

注意:参数中所用到的指针,只能是诸如 calloc()、malloc()函数分配空间时所用到的指针。

下面,练习使用一下 calloc()和 free()这两个函数,如练习 14.3 所示。

#### 〔练习 14.3〕

```
C>type exp 14_3.c
#include <stdio.h>
#include <stdlib.h>
#define N 9
#define S 1
main()
{
    extern void * calloc();
    extern void free();
    extern char * fgets();
    extern int fputs();
    char * buffer;
    buffer=calloc(N,S);
    /* 分配 N*S 个字节 */
    if(! buffer)
    {
```

```

    printf("空间不够,分配失败!!");
    abort();
}
fputs("分配成功,输入数据后,敲 Ctrl-Z 结束\n",stdout);
while(fgets(buffer,N,stdin))
    fputs(buffer,stdout);
free(buffer);
}

```

该程序执行结果如下所示:

C>exp14\_3

分配成功,输入数据后,敲 Ctrl-Z 结束

apodifj

apodifj

apodifj

apodifj

asdoifuapsoiefpq

asdoifuapsoiefpq

aposieutpqowiertupqowierupqw

aposieutpqowiertupqowierupqw

^ Z

C>

③函数 malloc() 该函数原型也在 stdlib.h 中定义,其类型和参数定义如下:

```
void * malloc(unsigned size);
```

功能:在堆中分配大小为 size 的内存空间。

返回:指向所分配的内存空间的首字节的指针。

注意:使用所分配的内存空间前,要先测试指向该区域的指针是否为空指针,若为空,则用 exit

)使程序终止执行。

〔练习 14.4〕 这是练习 malloc()函数的用法的程序。

C>type exp14\_4.c

```
#include <stdlib.h>
```

```
struct address{
```

```
    char name[9];
```

```
    char city[7];
```

```
    char zip[6];
```

```
}sl;
```

```
main()
```

```
{
```

```
    struct address *t;
```

```
    t=(struct address *)malloc(sizeof(sl));
```

```
    if(t==NULL)
```

```
{
```

```
    printf("分配错误,终止程序!! \n");
```

```
    exit(0);
```

```

}
gets(t);
printf("%s\n",t);
free(t);
}

```

该程序执行结果如下所示:

```

C>exp14-4
z. xcvz,xmcvnm.
z. xcvz,xmcvnm.
C>

```

④函数 `realloc()` 该函数的原型也在 `stdlib.h` 中定义,其类型和参数定义如下:

```
void *realloc(void *ptr,unsigned newsize);
```

功能:把由指针 `ptr` 所指向的堆的存储空间,由原来所分配的大小,改为 `newsize` 个字节。

返回:指向所分配的内存空间的指针。可能是原内存块,也可能是新的内存块。原内存块不够用时,该函数会开辟新的内存块,并把原内存块的数据拷贝到新块中,即原数据不会丢失。若无足够的存储空间,该函数返回空指针。

注意:要验证 `realloc` 是否成功。

〔练习 14.5〕 这是练习 `realloc()` 函数的用法程序。

```

c>exp14-5.c
#include <stdlib.h>
main()
{
    extern void *malloc(), *realloc();
    extern void free();
    char *ptr;
    unsigned s;
    s=10;
    ptr=(char *)malloc(s);
    if(ptr==NULL)
    {
        printf("分配失败,终止程序!\n");
        exit(0);
    }
    strcpy(ptr,"这是练习 3");
    printf("%s\n",ptr);
    s=30;
    ptr=(char *)realloc(ptr,s);
    strcat(ptr,"练习 realloc()函数");
    printf("%s\n",ptr);
    free(ptr);
}

```

```
C>exp14-5
```

这是练习 3



这是练习 3 练习 realloc()函数

C>

## (2)内存单元读取函数

一般的 C 编译系统都有这类库函数。它们并非是 ANSI 标准定义的。这类函数有：

- peek()
- peekb()
- poke()
- pokeb()

在 Turbo C 中,它们的原型在 dos.h 中,其类型和参数定义分别是:

```
int peek(int seg,unsigned offset);  
char peekb (int seg,unsigned offset);  
void poke(int seg,unsigned offset,int word);  
void pokeb(int seg,unsigned offset,int byte);
```

它们的功能分别是:

peek()——返回地址为 seg:offset 的字单元的值;

peekb()——返回地址为 seg:offset 的字节单元的值;

poke()——把 word 的 16 位值放到地址为 seg:offset 的字单元中;

pokdb()——把 byte 的 8 位值放到地址为 seg:offset 的字节单元中。

〔练习 14.6〕 这是利用 peekb()和 pokeb()这两个函数对 PC 机 1MB 内存进行读写的程序。

源程序如下:

```
C>type ext14_6.c  
#include "dos.h"  
#include "stdio.h"  
#define STRLONG 100  
main()  
{  
    unsigned i,j,min,max;  
    int test();  
    void modi();  
    char temp[STRLONG],a;  
    for(;;)  
    {  
        printf("1.....查找字符串\n");  
        printf("2.....修改内存单元\n");  
        printf("3.....退出\n");  
        printf("请选择:");  
        scanf("%c",&a);  
        if(a=='3')  
            return;  
        else  
            if(a=='1')
```

```

    {
        printf("请输入查找范围\n");
        printf("下限:");
        scanf("%x",&min);
        printf("上限:");
        scanf("%x",&max);
        printf("请输入要查找的字符串\n");
        gets(temp);
        gets(temp);
        printf("系统正在查找,请稍等...\n");
        for(i=min;i<max;++i)
            for(j=0;j<=0xf;++j)
                if(test(i,j,temp)==0)
                    printf("地址为:%x,%x\n",i,j);
    }
else
    if(a=='2')
    {
        printf("请输入段地址:");
        scanf("%x",&i);
        printf("请输入偏移量:");
        scanf("%x",&j);
        printf("请输入填充内容:");
        gets(temp);
        gets(temp);
        modi(i,j,temp);
    }
}

int test(i,j,k)
unsigned i,j;
char k[STRLONG];
{
    int t;
    char k1[STRLONG];
    for(t=0;t<strlen(k);++t)
        k1[t]=peekb(i,j+t);
    k1[strlen(k)]='\0';
    return strcmp(k,k1);
}

void modi(seg,off,temp)
unsigned seg,off;
char temp[STRLONG];
{

```

```

int t;
for (t=0;t<strlen(temp);++t)
    pokeb(seg,off+t,temp[t]);
return;
}

```

### (3) 寄存器管理函数

在 C 编译系统中, 寄存器管理函数不多见; 在一些 C 编译系统中常见有函数 `segread()`。在 Turbo C 中, 其类型和参数定义如下:

```
void segread(struct SREGS *sregp);
```

功能是把段寄存器的当前值拷贝到 `sregp` 指向的结构 `SREGS` 中。

在 Turbo C 中, 函数 `segread()` 和结构类型 `SREGS` 都是在 `dos.h` 文件中定义的。

〔练习 14.7〕 这是用 `segread()` 函数查看 CPU 中的段寄存器内容的程序。

```
C>type exp14_7.c
```

```

#include (dos.h)
main()
{
    extern void * malloc();
    extern void segread();
    struct SREGS * sregp;
    sregp=(struct SREGS *)malloc(sizeof(sregp));
    segread(sregp);
    printf("PC 机, 段寄存器内容: \n");
    printf("cs: %4x\nds: %4x\nes: %4x\nss: %4x\n", sregp->cs, sregp->ds, sregp->es, sregp->ss);
}

```

该程序执行结果如下所示:

```
C>exp14_7
```

PC 机, 段寄存器内容:

```
cs:21e8
```

```
ds:2322
```

```
es:2322
```

```
ss:2322
```

```
C>
```

### (4) I/O 管理函数

一般的 C 编译系统都有 I/O 端口管理函数, 只是名称大同小异。Turbo C 的 I/O 端口管理函数如表 14.6 所示。

表 14.6 Turbo C 的 I/O 端口管理函数

函数名称和参数	功 能
int inport(int por)	从 por 端口读取一个字
int inportb(int por)	从 por 端口读取一个字节
void output(int por, int wor)	把 wor(字)的值输出到 por 端口
void outputb(int por, char b)	把 b(字节)的值输出到 por 端口

〔练习 14.8〕 通过本章第 2 节的学习,我们已经了解了 PC 机音响系统的组成、产生乐曲的原理,以及用汇编语言实现的方法。本练习将用本节介绍的 I/O 端口管理函数,用纯 C 语言实现演奏《友谊地久天长》乐曲。源程序如下:

```
C>type exp14_8.c
#include "stdio.h"
#include "stdlib.h"
unsigned freq[88]={
    196,262,262,262,330,294,262,294,330,294,262,262,330,394,440,
    440,394,330,330,262,294,262,294,330,294,262,440,440,394,262,
    440,394,330,330,262,294,262,294,440,394,330,330,394,440,
    523,394,330,330,262,294,262,294,330,294,262,440,440,394,262,
    440,394,330,330,262,294,262,294,440,394,330,330,394,440,
    523,394,330,330,262,294,262,294,330,294,262,440,440,394,262,
},
int dely[88]={
    25,38,12,25,25,38,12,25,12,12,50,25,25,25,50,
    25,38,12,12,12,38,12,25,12,12,38,12,25,25,100,
    25,38,12,12,12,38,12,25,25,38,12,25,25,100,
    25,38,12,12,12,38,12,25,12,12,38,12,25,25,100,
    25,38,12,12,12,38,12,25,25,38,12,25,25,100,
    25,38,12,12,12,38,12,25,12,12,38,12,25,100,
};
main()
{
    int i;
    unsigned on;
    system("cls");
    gotoxy(4,12);
    printf("歌曲名:友谊地久天长\n");
    for(i=0;i<88;i++)
    {
        outportb(0x43,0xb6);
        freq[i]=0x1234dc/freq[i];
        outportb(0x42,freq[i]&0x00ff);
        freq[i]=freq[i]>>8;
        outportb(0x42,freq[i]);
        on=inportb(0x61);
        outportb(0x61,on|3);
        delay(dely[i]*25);
        outportb(0x61,on);
    }
}
```

1

#### 14.4 执行 BIOS 软中断访问相应硬件

这种方法是使用库函数 `int86()` 和 `int86x()`, 执行指定的 ROM 中的 BIOS 软中断, 访问相应的硬件。

(1) 库函数 `int86()` 和 `int86x()` 这两个函数的功能都是根据给定的中断号, 执行一个形式为:

int 中断号

的软中断指令。

两个函数的具体情况如下:

① `int86()`

用法: `#include <dos.h>`

`int86(int intr_num, union REGS * inregs, union REGS * outregs)`

功能: 执行由参数 `intr_num` 指定的软中断。

过程: • 把联合变量 `inregs` 中的内容拷贝到 CPU 寄存器中;

• 执行 `intr_num` 号中断;

• 从中断返回时, CPU 寄存器的值存放在联合变量 `outregs` 中。

返回: AX 的值

② `int86x()`

用法: `#include <dos.h>`

`int86x(int intr_num, union REGS * inregs, union REGS * outregs, struct SREGS * segregs)`

功能: 执行由参数 `intr_num` 指定的软中断。

过程: • 把联合变量 `inregs` 的内容拷贝到 CPU 寄存器中;

• 把结构成员 `segregs->ds` 和 `segregs->es` 的值分别拷贝到 DS 和 ES 寄存器;

• 执行 `intr_num` 指定的软中断;

• 中断返回时, 把 CPU 寄存器内容存放到联合变量 `outregs` 中。

返回: AX 的值

#### (2) 联合类型 REGS 和结构类型 SREGS

联合类型 REGS 和结构类型 SREGS 在标题文件 `dos.h` 中定义, 其内容分别如下。

```
union REGS{
    struct WORDREGS w;
    struct BYTEREGS b;
};

struct SREGS{
    unsigned int es;
    unsigned int cs;
    unsigned int ss;
    unsigned int ds;
};
```

其中, WORDREGS 和 BYTEREGS 为结构类型, 分别定义为:

```

struct WORDREGS
{
    unsigned int ax,bx,cx,dx,si,di,cflag;
};

struct BYTEREGS
{
    unsigned char al,ah,bl,bh,cl,ch,dl,dh;
};

```

(3)ROM BIOS ROM BIOS除了自诊断测试和自举装入主引导记录外,主要是提供设备一级的I/O驱动程序。它们都以软中断指令形式出现。如用户需要,可直接用形式为:INT n的中断指令调用。可供用户直接调用的软中断指令如表14.7所示。

表 14.7 用户可直接调用的软中断指令

中断号(n)	功 能
5h	屏幕拷贝
10h	视频 I/O
11h	设备配置
12h	内存容量
13h	磁盘 I/O
14h	串行口 I/O
15h	盒式磁带控制
16h	键盘 I/O
17h	打印机 I/O
18h	执行 ROM BASIC
19h	执行引导装入程序
1Ah	时间与日期

表14.7中的10h—1Ah为设备I/O驱动程序,现分别介绍如下:

①INT 10H,其功能设置、输入和输出参数如表14.8所示。

表 14.8 INT 10H 功能表

功 能 号	输 入 参 数	输 出 参 数
AH=0 (置屏幕方式)	AL=0 40×25 黑白 AL=1 40×25 彩色 AL=2 80×25 黑白 AL=3 80×25 彩色 AL=4 320×200 彩色图形 AL=5 320×200 黑白图形 AL=6 340×200 黑白图形	建立相应显示方式
AH=1 (置光标行)	CH:低5位为开始行 CL:低5位为结束行	按CX值设置光标类型
AH=2 (置光标位置)	DH:行 DL:列 BH:页	按输入值放置光标位置
AH=3 (读光标位置)	BH:显示页号	DH:行 DL:列 CX:方式

续表

功 能 号	输 入 参 数	输 出 参 数
AH=4 (读光笔位置)	AL=0~8	7AH=0:光笔未激发 AH=1:光笔已激发 DH:行 DL:列 CH:光栅行(0~199) BH:象素列(0~319 或 639)
AH=5(置活动显示页)		
AH=6 (向上滚页)	AL:滚动行数,0代表全屏幕 CH:滚动的左上角行 CL:滚动的左上角列 DH:滚动的右下角行 DL:滚动的右下角列 BH:空行属性	
AH=7 (向下滚页)		
AH=8 (读光标处字符)	BH:显示页	AL:读入字符 AH:属性
AH=9 (在光标处写字符及属性)	BH:显示页 BL:属性 CX:写字符次数 AL:字符	按输入值在当前光标位置显示字符
AH=0AH (在当前光标处写字符)	BH:显示页 CX:写字符次数 AL:字符	按输入值在当前光标处显示字符
AH=0BH (置调色板)	BH:调色板号 BL:颜色	按输入值设置调色板
AH=0CH (写象素)	DX:行号 CX:列号 AL:颜色值	在指定的位置显示象点
AH=0DH (读象素)	DX:行号 CX:列号	AL:读入的指定象点
AH=0EH (向屏幕写字符)	AL:字符 BL:前景色 BH:显示页	在当前页面显示字符
AH=0FH (读显示状态)		AL:当前状态 AH:屏幕列数 BH:当前页号

②INT 11H 其功能、输入和输出参数如表 14.9 所示。

表 14.9 INT 11H 功能表

功 能 号	输 入 参 数	输 出 参 数
		AX:系统设备配置状况 15、14 位:附加打印机数目 13:串行打印机(PCjr)(1:有) 12 位:游戏卡(1:有) 11-9 位:RS232 口数目 8 位:DMA(0:有) 7、6 位:软盘驱动器数(0:1 个) 5、4 位:显示方式 (01:40 列 10:80 列彩色 11:80 列单色) 3、2 位:系统板 RAM (11:64K) 1 位:未用 0 位:软盘自举

③INT 12H 其功能、输入和输出参数如表 14.10 所示。

表 14.10 INT 12H 功能表

功 能 号	输 入 参 数	输 出 参 数
		AX,存储容量(K)

④INT 13H 用于软盘 I/O 中断,其功能、输入和输出参数如表 14.11 所示。

表 14.11 INT 13H 功能表

功 能 号	输 入 参 数	输 出 参 数
AH=0 (复位磁盘)		
AH=1 (读磁盘状态)		AL:状态
AH=2 (读扇区)	DL:驱动器号(0-3) DH:磁头号(0-1) CH:磁道号(0-39) CL:扇区号(1-8)	将指定的扇区内容读入缓冲区 AH:磁盘操作状态 CF=0:成功 CF=1:错误 AL:读的扇区个数
AH=3 (写扇区)	AL:扇区个数(≤8) ES,BX:缓冲区首址	将缓冲区内容写入扇区 (寄存器同上)
AH=4 (校验扇区)		校验指定的扇区 (寄存器同上)
AH=5 (格式化)	DL,DH,CH 同上 ES,BX:指向该磁道上指定地址 字段的集合 C=磁道号 H=磁头号 R=扇区号 N=字节数/区 (00=128,01=256, 10=512,11=1024)	同 AH=3

⑤INT 13H 用于硬盘 I/O 中断,其功能、输入和输出参数如表 14.12 所示。



表 14.12 INT 13H 功能表

功 能 号	输 入 参 数	输 出 参 数
AH=00 (复位硬盘)	DL:驱动器号 (80-87H 为硬盘)	磁盘按功能规定操作 CF=0:操作成功 CF=1:操作失败
AH=01 (读磁盘操作状态)	DH:磁头号(0-7) CH:圆柱面(0-1024) CL:扇区号(1-17)	AH:当前操作状态 出现 11H,有一数据错被 ECC 纠正
AH=02 (读指定扇区)	AL:扇区个数(1-80H) ES:BX:指向读/写缓冲区	AL:错误的长度 DL:驱动器数 DH:磁头号最大可用值 CH:圆柱面最大可用值 CL:扇区号最大可用值
AH=03 (写指定扇区)		
AH=04 (校验指定扇区)		
AH=05 (格式化)		
AH=06 (同上,且设坏扇区标志)		
AH=07 (从指定磁道开始格式化)		
AH=08 (返回当前驱动器参数)		
AH=09 (驱动器对性能初始化)		
AH=0A (长读扇区)	(512+4) 4 字 节 为 ECC	
AH=0B (长写扇区)		
AH=0C (查找磁道)		
AH=0D (变更磁盘复位)		
AH=0E (读扇区缓冲器)		
AH=0F (写扇区缓冲器)		
AH=10 (测试驱动器准备情况)		
AH=11 (磁盘再定位)		
AH=12 (控制器 RAM 诊断)		
AH=13 (驱动器诊断)		
AH=14 (控制器内部诊断)		

⑥INT 14H 其功能、输入和输出参数如表 14.13 所示。

表 14.13 INT 14H 功能表

功 能 号	输 入 参 数	输 出 参 数
AH=0 (初始化通信口)	AL:初始化参数 7,6,5 位:波特率 4,3 位:奇偶选择 2 位:停止位 1,0 位:字长	AH:线路状态
AH=1 (发送字符)	AL:发送字符	AH:返回状态 (7 位=1:未能发送)

续表

功能号	输入参数	输出参数
AH=2 (接收字符)		AL:接收的字符 AH:返回状态 (7位=1:超时)
AH=3 (返回状态)		AH:线路状态 AL:调制解调器状态

⑦INT 16H 其功能、输入和输出参数如表 14.14 所示。

表 14.14 INT 16H 功能表

功能号	输入参数	输出参数
AH=0 (读键符)		AL:键入的字符 AH:该字符的扫描码
AH=1 (判键符)		ZF=1:无码可读 ZF=0:有码可读 (可读字符在 AX 中)
AH=2 (返回状态)		AL:当前换挡状态

⑧INT 17H 其功能、输入和输出参数如表 14.15 所示。

表 14.15 INT 17H 功能表

功能号	输入参数	输出参数
AH=0 (打印字符)	AL:待打印字符	AH=1:超时,字符不打印
AH=1 (初始化打印机)		AH:打印机状态
AH=2 (读状态)		AH:打印机状态

⑨INT 1AH 其功能、输入和输出参数如表 14.16 所示。

表 14.16 INT 1AH 功能表

功能号	输入参数	输出参数
AH=0 (读当前时钟)		CX:计数值高位 DX:计数值低位 AL=0:未滿 24 小时 AL≠0:超过 24 小时
AH=1 (设置当前时钟)	CX:计数值高位 DX:计数值低位	按输入值设置时钟

#### (4)键扫描码

在使用 INT 16H 时,要涉及到键的扫描码值。为方便读者,这里给出,如表 14.17 和 14.18 所示。

表 14.17 PC 机键盘扫描码

键号	扫描码	基本档	上 档	带 Ctrl	带 Alt
1	01	ESC	ESC	ESC	禁止
2	02	1	!	禁止	可扩展
3	03	2	@	Nul	可扩展
4	04	3	#	禁止	可扩展
5	05	4	\$	禁止	可扩展
6	06	5	%	禁止	可扩展
7	07	6	^	RS(30)	可扩展
8	08	7	&	禁止	可扩展
9	09	8	*	禁止	可扩展
10	0A	9	(	禁止	可扩展
11	0B	0	)	禁止	可扩展
12	0C	-	_	US(31)	可扩展
13	0D	=	+	禁止	可扩展
14	0E	Back space	Back space	Del(127)	禁止
15	0F	Tab	可扩展	禁止	禁止
16	10	q	Q	DC1(17)	可扩展
17	11	w	W	ETB(23)	可扩展
18	12	e	E	ENQ(15)	可扩展
19	13	r	R	DC2(18)	可扩展
20	14	t	T	DC4(20)	可扩展
21	15	y	Y	EM(25)	可扩展
22	16	u	U	NAK(21)	可扩展
23	17	i	I	HT(9)	可扩展
24	18	o	O	SI(15)	可扩展
25	19	p	P	DLE(16)	可扩展
26	1A	[	{	ESC(27)	禁止
27	1B	]	}	GS(29)	禁止
28	1C	Enter	Enter	LF(10)	禁止
29	1D	Ctrl	禁止	禁止	禁止
30	1E	a	A	SOH(1)	可扩展
31	1F	s	S	DC3(19)	可扩展
32	20	d	D	EOT(4)	可扩展
33	21	f	F	ACK(6)	可扩展
34	22	g	G	BEL(7)	可扩展
35	23	h	H	BS(8)	可扩展
36	24	j	J	LF(10)	可扩展
37	25	k	K	VT(11)	可扩展
38	26	l	L	FF(12)	可扩展
39	27	;	:	禁止	禁止
40	28	'	"	禁止	禁止
41	29	`	~	禁止	禁止
42	2A	Left shift	禁止	禁止	禁止
43	2B	\		FS(28)	禁止
44	2C	z	Z	SUB(26)	可扩展
45	2D	x	X	CAN(24)	可扩展
46	2E	c	C	ETS(3)	可扩展
47	2F	v	V	SYN(22)	可扩展
48	30	b	B	STX(2)	可扩展
49	31	n	N	SO(14)	可扩展
50	32	m	M	CR(13)	可扩展
51	33	,	<	禁止	禁止
52	34	.	>	禁止	禁止
53	35	/	?	禁止	禁止
54	36	Right shift	禁止	禁止	禁止
55	37	*	Print Screen	可扩展	禁止
56	38	Alt	禁止	禁止	禁止
57	39	Spacebar	Spacebar	Spacebar	Spacebar
58	3A	Caps lock	禁止	禁止	禁止
59	3B	F1	可扩展	可扩展	可扩展

续表

键号	扫描码	基本档	上 档	带 Ctrl	带 Alt
60	3C	F2	可扩展	可扩展	可扩展
61	3D	F3	可扩展	可扩展	可扩展
62	3E	F4	可扩展	可扩展	可扩展
63	3F	F5	可扩展	可扩展	可扩展
64	40	F6	可扩展	可扩展	可扩展
65	41	F7	可扩展	可扩展	可扩展
66	42	F8	可扩展	可扩展	可扩展
67	43	F9	可扩展	可扩展	可扩展
68	44	F10	可扩展	可扩展	可扩展
69	45	Num lock	禁止	Pause	禁止
70	46	Scroll lock	禁止	Break	禁止
71	47	Home	NA	Clear Screen	禁止
72	48	↑	NA	禁止	禁止
73	49	PgUp	NA	Top of Text	禁止
74	4A	Numpad—	NA	禁止	禁止
75	4B	←	NA	可扩展	禁止
76	4C	Numpad5	NA	禁止	禁止
77	4D	→	NA	可扩展	禁止
78	4E	Numpad+	NA	禁止	禁止
79	4F	End	NA	可扩展	禁止
80	50	↓	NA	禁止	禁止
81	51	PgDn	NA	可扩展	禁止
82	52	Ins	NA	禁止	禁止
83	53	Del	NA	Reset	Reset

表 14.18 AT 机 84 键盘扫描码

键号	扫描码	基本档	上 档
1	29	~	~
2	02	1	1
3	03	2	@
4	04	3	#
5	05	4	\$
6	06	5	%
7	07	6	^
8	08	7	&
9	09	8	*
10	0A	9	(
11	0B	0	)
12	0C	_	-
13	0D	=	+
14	2B	\	
15	0E	Back space	Back space
16	0F	Tab	Back tab
17	10	q	Q
18	11	w	W
19	12	e	E
20	13	r	R
21	14	t	T
22	15	y	Y
23	16	u	U
24	17	i	I
25	18	o	O
26	19	p	P
27	1A	[	{
28	1B	]	}
30	1D	Ctrl	禁止
31	1E	a	A

续表

键号	扫描码	基本档	上 档
32	1F	s	S
33	20	d	D
34	21	f	F
35	22	g	G
36	23	h	H
37	24	j	J
38	25	k	K
39	26	l	L
40	27	;	;
41	28	'	'
43	1C	Enter	Enter
44	2A	禁止	禁止
46	2C	z	Z
47	2D	x	X
48	2E	c	C
49	2F	v	V
50	30	b	B
51	31	n	N
52	32	m	M
53	33	,	<
54	34	.	>
55	35	/	?
57	36	Right shift	禁止
58	38	Alt	禁止
61	39	spacebar	
64	3A	Caps lock	禁止
65	3C	F2	
66	3E	F4	
67	40	F6	
68	42	F8	
69	44	F10	
70	3B	F1	
71	3D	F3	
72	3F	F5	
73	41	F7	
74	43	F9	
75	E0,52	Insert	
90	01	ESC	ESC
91	47	Keypad 7	Home
92	4B	Keypad 4	←
93	4F	Keypad 1	End
95	45	Num lock	禁止
96	48	Keypad 8	↑
97	4C	Keypad 5	禁止
98	50	Keypad 2	↓
99	52	Keypad 0	Ins
100	46	Scroll lock	禁止
101	49	Keypad 9	Page Up
102	4D	Keypad 6	→
103	51	Keypad 3	Page Down
104	53	Keypad.	Delete
105	54	Sys Reg	
106		Keypad *	Prtn
107	4A	Keypad -	
108	4E	Keypad +	

(5)应用实例 这里,通过实例来说明函数 `int86()` 的使用。

〔练习 14.9〕 这是最简单的编辑程序,能在屏幕终端上移动光栅,能输入字符。源程序如下。

C:\ws>type b:exp14\_9.c

```

#include "dos.h"
#include "conio.h"
#define row 24
#define col 79
void cls();
void cursor();
void goto_xy();
int get_key();
main()
{ union REGS r;
  union scan{
    int c;
    char ch[2];
  }sc;
  int x,y;
  int i;
  cls();
  x=0;
  y=0;
  goto_xy(x,y);
  cursor();
  r.h.ah=0;
  r.h.al=3;
  int86(0x10,&r,&r);
  for(;;)
  { sc.c=get_key();
    if(sc.ch[0]==27)/* ESC 退出 */
      exit(1);
    if(sc.ch[0])/* ASC 码字符显示 */
    { r.h.ah=9;
      r.h.al=sc.ch[0];
      r.h.bl=7;
      r.h.bh=0;
      r.x.cx=1;
      int86(0x10,&r,&r);
      y++;
      if(y==79)
      { y=0;
        x++;
        if(x==24)
          x=0;
      }
    }
    if(sc.ch[1])

```

```

    { switch(sc.ch[1])
      {
        case 72: /* 上移 */
          if(x>0) x--;
          break;
        case 80: /* 下移 */
          if(x<row)
            x++;
          break;
        case 75: /* 左移 */
          if(y>0) y--;
          break;
        case 77: /* 右移 */
          if(y<col) y++;
          break;
      }
      goto_xy(x,y);
      cursor();
    }
  }
}

void cls() /* 清屏 */
{ union REGS r;
  r.h.ah=6;
  r.h.al=0;
  r.h.ch=0;
  r.h.cl=0;
  r.h.dh=24;
  r.h.dl=79;
  r.h.bh=7;
  int86(0x10,&r,&r);
}

void cursor() /* 设光标类型 */
{ union REGS r;
  r.h.ah=1;
  r.h.ch=2;
  r.h.cl=1;
  int86(0x10,&r,&r);
}

void goto_xy(x,y) /* 光标定位 */
int x,y;
{ union REGS r;
  r.h.ah=2; /* 置光标位置 */
  r.h.dh=x; /* 列坐标 */

```

```

r.h.dl=y; /*行坐标*/
r.h.bh=0; /*显示页*/
int86(0x10,&x,&x);
}
int get_key()/*获得键码*/
{ union REGS r;
  r.h.ah=0;
  return int86(0x16,&x,&x);
}

```

该程序的 cls() 函数中出现的寄存器 bh 的值, 是 BIOS 中断 INT 10H 所需要的参数值, 用来指明加到窗口底部的空行显示属性。该值可称作属性字节, 其含义如下:

- 对于彩色/图形适配器, 各位的含义如图 14.3 所示。

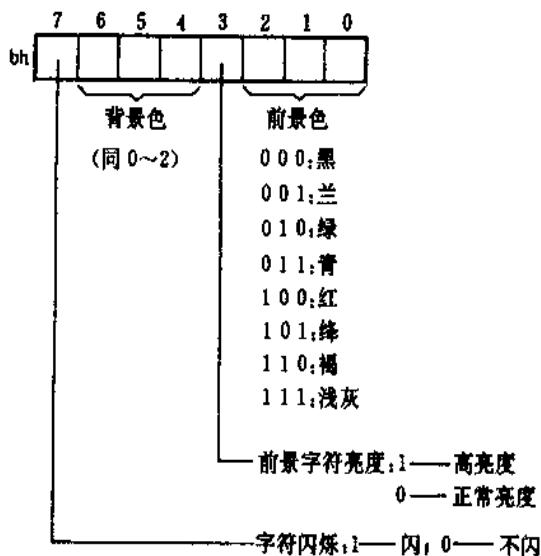


图 14.3 彩色/图形适配器的 bh 各位的含义

- 对于单色适配器, 各位的含义如图 14.4 所示。

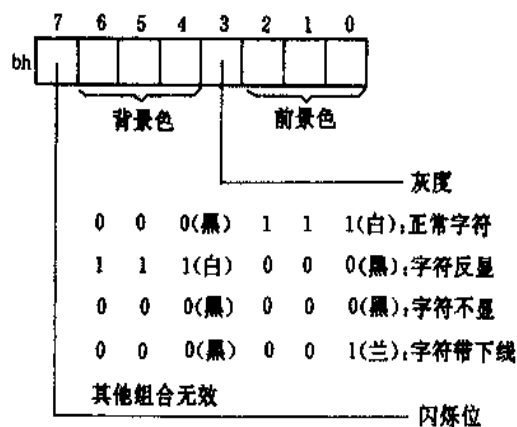


图 14.4 单色适配器的 bh 各位的含义



【练习 14.10】 该程序的功能是把所找到的文件进行分屏显示。

C:\USER\YZM\C>typeexpl4-10.c

/\* 功能:分屏显示所要找的文件 \*/

```
#include <stdio.h>
#include <dos.h>
#include <conio.h>
#include <process.h>
#include <dir.h>
FILE * fp;
char ch, * filename;
int row, i, j, r, c, p;
struct fblk * f;
union REGS in, out;
main(argc, argv)
int argc;
char * argv[];
{
    j=0; i=0; r=0; c=0; p=0;
    printf("给出最大行号、列号");
    scanf("%d%d", &r, &c);
    if(argc != 2){ printf("\n 使用格式错误\n");
        exit(1);
    }
    filename=argv[1]; /* 要找的文件名 */
    j=findfirst(filename, f, FA_RDONLY); /* 搜寻文件 */
    if(j != 0){ printf("未找到!");
        exit(1);
    }
    else{
        do
        {
            clrscr();
            gotoxy(1, 1);
            printf("\n 文件名: %s 尺寸: %ld\n", f->ff_name, f->ff_fsize);
            fp=fopen(f->ff_name, "r"); /* 打开找到的文件 */
            row=0;
            while((ch=fgetc(fp)) != EOF) /* 向屏幕输出文件内容 */
            { in.h.ah=3;
                in.h.bh=0;
                int86(0x10, &in, &out); /* 读光标位置 */
                if(out.h.dh != row)
                { if(out.h.dh <= r)
                    { row=out.h.dh;
                        gotoxy(1, row+1);
```

```

    }
    else { gotoxy(c/3,r+3);    /* 换屏 */
        printf(".....第%d 屏.....",++p);
        if(getch()==27) exit(1);
        clrscr(); {
            gotoxy(1,1);
        }
    }
}

in.h. ah=14;in.h. al=ch;
in.h. bh=0;in.h. bl=7;
int86(0x10,&in,&out);    /* 向屏幕写字符 */
}

fclose(f->ff_name);    /* 关闭文件 */
}while((i=findnext(f))==0);/* 进行下一个同名文件的显示 */
}
}

```

#### 14.5 调用 DOS 系统功能访问相应硬件

这种方法是使用库函数 `bdos()`, 执行指定的 DOS 系统功能调用, 访问相应的硬件。

(1) 库函数 `bdos()` 和 `bdosptr()` 这两个函数的原型在文件 `dos.h` 中定义, 它们是 ANSI 标准的一员, 其功能是根据给定的功能号, 执行相应的 DOS 的 INT 21H 系统功能调用。具体情况如下。

用法: `bdos(int fn, unsigned dx, unsigned al);`

`bdosptr(int fn, void *dsdx, unsigned al);`

其中, `fn` 为功能调用号。

功能:

- 首先把 `dx` 和 `al` 的值放入 `DX` 和 `AL` 寄存器;
- 然后执行由 `fn` 指定的 INT 21H DOS 系统功能调用;
- 对于微型、小型和中型存储模式来说, 这两个函数的功能是相同的; 但在使用大型存储模式时, 需要把 20 位的指针参数传递给 DOS(DS:DX)。这时, 就要用 `bdosptr()` 函数来代替 `bdos` 函数。

返回: 返回 `AX` 寄存器的值, DOS 使用该值返回信息。

(2) 库函数 `intdos()` 和 `intdosx()` 这两个函数不是 ANSI 标准的成员, 定义在文件 `dos.h` 中。它们的具体情况如下。

用法: `intdos(union REGS *in_regs, union REGS *out_regs);`

`intdosx(union REGS *in_regs, union REGS *out_regs, struct SREGS *segregs);`

功能: 执行 `in_regs` 指向的联合变量所给定的 DOS 系统功能调用, 并把结果存入 `out_regs` 指向的联合变量中。

返回: 返回 `AX` 寄存器中供 DOS 返回信息用的值。返回时, 若进位标志被置位, 则说明出现错误。

适用: 用于需要 `DX` 和 `AL` 之外的寄存器参数, 或者要求返回 `AX` 之外的某一寄存器值的系统

功能调用。其中,函数 `intdosx()` 主要用于大型存储模式,它的参数 `segregs` 用来指定 DS 和 ES 寄存器。

(3)DOS 系统功能调用 DOS 在更高层次上,提供了与 BIOS 相同的功能,比起 BIOS 来,其优点有:使用更方便,接口简单,可移植性好;其缺点是,效率较低,功能没有 BIOS 丰富。正因为如此,在可能的情况下,要尽量使用 DOS 系统功能调用。这里,给出 DOS 系统功能调用摘要供参考,如表 14.19 所示。

表 14.19 DOS 系统功能调用及其支持版本摘要

功能号	功能名称	支持版本								分类
		1	1.1	2	2.1	3	3.1	3.2	3.3	
00H	终止程序	S	S	O	O	O	O	O	O	l
01H	读键盘并回显	S	S	S	S	S	S	S	S	a
02H	显示字符	S	S	S	S	S	S	S	S	b
03H	异步串行口输入	S	S	S	S	S	S	S	S	c
04H	异步串行口输出	S	S	S	S	S	S	S	S	c
05H	打印机字符输出	S	S	S	S	S	S	S	S	b
06H	直接控制台 I/O	S	S	S	S	S	S	S	S	a,b
07H	直接控制台输入(无回显)	S	S	S	S	S	S	S	S	a
08H	读键盘(无回显)	S	S	S	S	S	S	S	S	a
09H	显示字符串	S	S	S	S	S	S	S	S	b
0AH	输入字符串	S	S	S	S	S	S	S	S	a
0BH	检查键盘状态	S	S	S	S	S	S	S	S	a
0CH	清缓冲区,并读键盘	S	S	S	S	S	S	S	S	a
0DH	磁盘初始化	S	S	S	S	S	S	S	S	d
0EH	选择当前盘	S	S	S	S	S	S	S	S	d
0FH	打开文件	S	S	O	O	O	O	O	O	f
10H	关闭文件	S	S	O	O	O	O	O	O	f
11H	在当前目录中,查找第一匹配文件	S	S	O	O	O	O	O	O	f
12H	在当前目录中,查找下一个匹配文件	S	S	O	O	O	O	O	O	f
13H	删除文件	S	S	O	O	O	O	O	O	f
14H	顺序读记录	S	S	O	O	O	O	O	O	h
15H	顺序写记录	S	S	O	O	O	O	O	O	h
16H	建立文件	S	S	O	O	O	O	O	O	f
17H	文件易名	S	S	O	O	O	O	O	O	f
18H	保留	R	R	R	R	R	R	R	R	
19H	取当前盘号	S	S	O	O	O	O	O	O	d
1AH	置磁盘传送地址	S	S	S	S	S	S	S	S	d
1BH	取当前驱动器的 FAT 信息	S	S	O	O	O	O	O	O	d
1CH	取指定驱动器的 FAT 信息	S	S	O	O	O	O	O	O	d
1DH	保留	R	R	R	R	R	R	R	R	
1EH	保留	R	R	R	R	R	R	R	R	
1FH	保留	R	R	R	R	R	R	R	R	
20H	保留	R	R	R	R	R	R	R	R	
21H	随机读一个记录	S	S	R	R	R	R	R	R	h
22H	随机写一个记录	S	S	R	R	R	R	R	R	h
23H	取文件尺寸	S	S	R	R	R	R	R	R	f
24H	置随机记录字段	S	S	R	R	R	R	R	R	h
25H	置中断向量	S	S	S	S	S	S	S	S	m
26H	建立新程序段	S	S	S	S	S	S	S	S	k
27H	随机读若干记录	S	S	O	O	O	O	O	O	h
28H	随机写若干记录	S	S	O	O	O	O	O	O	h
29H	分析文件名	S	S	O	O	O	O	O	O	f
2AH	取日期	S	S	S	S	S	S	S	S	m
2BH	置日期	S	S	S	S	S	S	S	S	m
2CH	取时间	S	S	S	S	S	S	S	S	m
2DH	置时间	S	S	S	S	S	S	S	S	m
2EH	设置或关闭校验标志	S	S	S	S	S	S	S	S	d
2FH	取盘传送地址			S	S	S	S	S	S	d

续表

功能号	功能名	支持版本								分类
		1	1.1	2	2.1	3	3.1	3.2	3.3	
30H	取DOS版本			S	S	S	S	S	S	m
31H	终止进程并保持驻留			S	S	S	S	S	S	l
32H	保留	R	R	R	R	R	R	R	R	
33H	取或置 Ctrl-break			S	S	S	S	S	S	m
34H	保留	R	R	R	R	R	R	R	R	e
35H	取中断向量			S	S	S	S	S	S	m
36H	取盘自由空间			S	S	S	S	S	S	d
37H	保留	R	R	R	R	R	R	R	R	
38H.O	取国度信息			S	S	S	S	S	S	m
38H.X	置国度信息					S	S	S	S	m
39H	建立目录			S	S	S	S	S	S	e
3AH	删除目录			S	S	S	S	S	S	e
3BH	改变现行目录			S	S	S	S	S	S	e
3CH	建立文件			S	S	S	S	S	S	g
3DH	打开文件			S	S	S	S	S	S	g
3EH	关闭文件			S	S	S	S	S	S	g
3FH	读文件或设备			S	S	S	S	S	S	g
40H	写文件或设备			S	S	S	S	S	S	i
41H	删除文件			S	S	S	S	S	S	g
42H	移动读/写指针			S	S	S	S	S	S	i
43H	取或置文件属性			S	S	S	S	S	S	g
44H.0	取设备信息			S	S	S	S	S	S	i
44H.1	置设备信息			S	S	S	S	S	S	i
44H.2	读设备字符			S	S	S	S	S	S	i
44H.3	写设备字符			S	S	S	S	S	S	i
44H.4	读设备数据块			S	S	S	S	S	S	i
44H.5	写设备数据块			S	S	S	S	S	S	i
44H.6	取输入状态			S	S	S	S	S	S	i
44H.7	取输出状态			S	S	S	S	S	S	i
44H.8	检测设备是否有可移动介质					S	S	S	S	i
44H.9	测试是否定向数据模块						S	S	S	i
44H.A	测试设备是否定向句柄						S	S	S	i
44H.B	置共享重试次数					S	S	S	S	i
44H.C	字符设备的通用 I/O 控制							S	S	i
44H.D	块设备的通用 I/O 制							S	S	i
44H.E	取设备映象							S	S	i
44H.F	置设备映象							S	S	i
45H	复制文件句柄			S	S	S	S	S	S	g
46H	强行复制文件句柄		S	S	S	S	S	S	S	g
47H	取当前目录			S	S	S	S	S	S	e
48H	分配内存			S	S	S	S	S	S	k
49H	释放已分配内存			S	S	S	S	S	S	k
4AH	修改分配的内存块			S	S	S	S	S	S	k
4BH.0	装入并执行程序			S	S	S	S	S	S	k
4BH.3	装载覆盖			S	S	S	S	S	S	k
4CH	终止进程			S	S	S	S	S	S	l
4DH	取子进程的返回代码			S	S	S	S	S	S	k
4EH	搜索第一个匹配文件			S	S	S	S	S	S	g
4FH	搜索下一个匹配文件			S	S	S	S	S	S	g
50H	保留	R	R	R	R	R	R	R	R	
51H	保留	R	R	R	R	R	R	R	R	
52H	保留	R	R	R	R	R	R	R	R	
53H	保留	R	R	R	R	R	R	R	R	
54H	取校验开关状态			S	S	S	S	S	S	g
55H	保留	R	R	R	R	R	R	R	R	
56H	文件名			S	S	S	S	S	S	g
57H	取或置文件日期和时间			S	S	S	S	S	S	g
58H	取或置内存分配对策					S	S	S	S	j
59H	取扩充错误代码					S	S	S	S	m

续表

功能号	功能名称	支持版本								分类
		1	1.1	2	2.1	3	3.1	3.2	3.3	
5AH	建立一个临时文件					S	S	S	S	g
5BH	建立一个新文件					S	S	S	S	g
5CH,0	封闭文件					S	S	S	S	j
5CH,1	开锁文件					S	S	S	S	j
5DH	保留	R	R	R	R	R	R	R	R	j
5EH,0	取机器名						S	S	S	j
5EH,2	置打印机参数						S	S	S	j
5EH,3	取打印机参数						S	S	S	j
5FH,2	取重定向表入口						S	S	S	j
5FH,3	设备重定向						S	S	S	j
5FH,4	清除重定向						S	S	S	j
60H	保留	R	R	R	R	R	R	R	R	
61H	保留	R	R	R	R	R	R	R	R	
62H	取程序段前缀地址					S	S	S	S	m
63H	取扩展字符表地址				2.25					m
65H	取扩展国定信息								S	m
66H	取或置总的码页								S	m
67H	置句柄计数								S	m
68H	COM 执行文件								S	m

说明:

①表中出现的 S、O 和 R 分别代表:

S 表示支持

O 表示已过时或作废

R 表示 DOS 内部使用

②功能号带有两个值的,其中第 2 个值放在 AL 中;

③功能号为 63 的 DOS 系统功能调用仅适用于 2.25 版本;

④表中的分类项中的字母含义如下:

- a 从标准设备输入
- b 向标准设备输出
- c 串行端口 I/O
- d 磁盘级管理
- e 目录管理
- f 基于 FCB 的文件
- g 基于句柄的文件管
- h 基于 FCB 的文件的记录/字节级管理
- i 基于句柄的文件的记录/字节级管理
- j 网络管理
- k 内存和进程管理
- m 其它功能

⑤DOS4.0 和 DOS5.0 的系统功能调用与 DOS3.3 的基本相同。DOS4.0 只是扩大了 44H 的功能和支持 6CH 系统功能调用;DOS5.0 只是扩大了 44H 和 58H 的功能并支持 1FH、32H、33H 和 63H 系统功能调用。

(4)应用实例 这里,给出使用 `intdos()` 函数进行 DOS 系统功能调用的例子。

〔练习 14.11〕 该程序的功能是把 A 盘上的文件拷贝到 B 盘上。

C:\USER\YZM\C>type exp14\_11.c

/\* 该程序的功能:把 A 盘上的文件拷贝到 B 盘 \*/

```
#include <stdio.h>
#include <io.h>
#include <dos.h>
#include <sys\stat.h>
#include <fcntl.h>
#define A 0
#define B 1
#define C 2
char * name, * sname, * oname;
main()
{
    cls();
    gotoxy(10,20);
    printf("敲入原文件名:");
    scanf("%s",sname);
    gotoxy(10,20);
    printf("敲入目标文件名:");
    scanf("%s",oname);
    gotlxy(10,20);
    printf("正在拷贝,请稍等!\n");
    fcopy(sname,oname);
    gotoxy(10,20);
    printf("拷贝完毕!");
}
cls() /* 清屏函数 */
{
    union REGS rv;
    rv.x.ax=0x0600;
    rv.x.bx=0x1700;
    rv.x.cx=0x0000;
    rv.x.dx=0x184f;
    int86(0x10,&rv,&rv);
}
fcopy(aname,bname) /* 拷贝函数 */
char * aname, * bname;
{
    int fout,fin,n;
    char buf[3];
    union REGS rv;
    fout=open(aname,o_RDWR); /* 打开要拷贝的文件 */
    rv.x.ax=0x0e00;
```

```

rv.x.dx=B;
intdos(&rv,&rv);
fin=creat(bname,S_IWRITE);      /* 建立一个文件 */
do{ rv.x.ax=0x0e00;
    rv.x.dx=0x0000;
    intdos(&rv,&rv);              /* 读原文件 */
    n=read(fout,buf,1);
    rv.x.ax=0x0e00;
    rv.x.dx=B;
    intdos(&rv,&rv);
    write(fin,buf,n);            /* 写目标文件 */
}while(n !=0);
rv.x.ax=0x0e00;                  /* 关闭两个文件 */
rv.x.dx=B;
intdos(&rv,&rv);
close(fin);
rv.x.ax=0x0e00;
rv.x.dx=0x0000;
intdos(&rv,&rv);
close(fout);
}

```

## 第十五章 排序与检索算法及其实现

所谓算法是解决问题的方法或规则。算法可分为两大类,即数值计算的算法和非数值计算的算法。

本章讨论排序和检索这两种非数值计算的算法,并给出在 PC 机上,使用 C 语言,实际通过的程序。

### 15.1 排序的基本方法

所谓排序是指按指定关键字的值,或按升序,或按降序,重新调整数据元素的位置。下面介绍其基本算法。

(1)冒泡排序法 此法就是交换排序法。它是从数组的最后一个元素开始,两两相邻元素进行比较,直到把最小元素(按升序)或最大元素(按降序)放到最前面为止。例如,要把下列元素升序排序,则从后向前的冒泡排序过程为:

7 3 2 5 8 1 6  
1步结果:[1] 7 3 2 5 8 6  
2步结果:[1 2] 7 3 5 6 8  
3步结果:[1 2 3] 7 5 6 8  
4步结果:[1 2 3 5] 7 6 8  
5步结果:[1 2 3 5 6] 7 8  
6步结果:[1 2 3 5 6 7] 8  
7步结果:[1 2 3 5 6 7 8]

〔练习 15.1〕 该练习是用冒泡排序法对字符数组进行升序排序的程序,其中函数 bubble()为排序函数。

```
C>type a:exp15_1.c
#include <stdio.h>
main()
{ int n;
  char *i;
  i=(char *) malloc(256);
```



```

printf("请输入数据元素个数:元素序列\n");
scanf("%d;%s",&n,i);
bubble(i,n);
printf("%s\n",i);
}
bubble(char *item,int count)
{ int a,b;
  char t;
  for (a=1;a<count;a++)
    for(b=count-1;b>=a;b--)
      if(item[b-1]>item[b])
        { t=item[b-1];
          item[b-1]=item[b];
          item[b]=t;
        }
}

```

该程序执行结果如下所示:

C>a:exp15\_1

请输入数据元素个数:元素序列

7,7325816

1235678

C>

(2)选择排序法 该法是从要排序的数据元素中选出最小元素(按升序),或最大元素(按降序),与原来第一个元素交换位置,然后再从余下的  $n-1$  个元素中重复上述的选择与交换,这种重复操作直到最后两元素为止。例如,要把下列数据元素按升序排序,则选择排序的过程为:

元素序号:[0][1][2][3][4][5][6]

元素:     7   3   2   5   8   1   6

第1步:[1]3   2   5   8   7   6

第2步:[1   2]3   5   8   7   6

第3步:[1   2   3]5   8   7   6

第4步:[1   2   3   5]8   7   6

第5步:[1   2   3   5   6]7   8

第6步:[1   2   3   5   6   7]8

(练习 15.2) 该练习是用选择排序法对数组元素排序的程序。其中,函数 select()为选择排序函数。

A>type a:exp15\_2.c

main()

{ int n;

  char \*i;

  i=(char \*) malloc(256);

  printf("请输入数据元素个数:元素序列\n");

```

scanf("%d:%s",&n,i);
select(i,n);
printf("%s\n",i);
}
select(char * item,int count)
{ int a,b,c;
  char t;
  for(a=0;a<count-1;a++)
  { c=a;
    t=item[a];
    for(b=a+1;b<count;b++)
      if(item[b]<t)
      { c=b;
        t=item[b];
      }
    item[c]=item[a];
    item[a]=t;
  }
}

```

该程序执行结果如下所示:

A>exp15\_2

请输入数据元素个数:元素序列

7:8924651

1245689

A>

(3)插入排序法 该法是从1号元素(即第2个元素)开始,依次把后面的元素按大小插入到前面的元素中间。例如,要把下列数组按升序排序,则插入排序过程为:

元素序号:[0] [1] [2] [3] [4]

元 素: 5 3 4 1 2

第1步: [3 5] 4 1 2

第2步: [3 4 5] 1 2

第3步: [1 3 4 5] 2

第4步: [1 2 3 4 5]

〔练习 15.3〕 该练习是用插入排序法对数组元素进行排序的程序。其中,函数 insert()为插入排序的函数。

C>type a:exp15\_3.c

main()

{ int n;

char \* i;

i=(char \*)malloc(256);

printf("请输入数据元素个数:元素序列\n");

scanf("%d:%s",&n,i);

```

    insert(i,n);
    printf("%s\n",i);
}
insert(char * item,int count)
{ register int a,b;
  char t;
  for(a=1;a<count;a++)
  {
    t=item[a];
    b=a-1;
    while(b>=0&&item[b]>t)
    { item[b+1]=item[b];
      b--;
    }
    item[b+1]=t;
  }
}

```

该程序执行结果如下所示:

C>a:exp15\_3

请输入数据元素个数,元素序列

7,8967321

1236789

C>

## 15.2 改进型排序方法

(1)二分法插入排序 这是插入排序的改进型。这种排序法是对具有一定间隔的两数进行比较,且间隔逐步缩小,直到间隔为1止。这种方法不仅交换,而且要按一定间隔把被比较的元素放到适当的位置。可见,这种方法具有插入的性质。

〔练习 15.4〕 该练习是按改进型插入排序法对数组元素进行升序排序的程序。其中,函数 ins()为改进型插入排序函数。

```

C>type a:exp15_4.c
#include <stdio.h>
#include <string.h>
#define MAX 1000
main()
{
  int i,n;
  char s[MAX];
  printf("请输入数据元素序列:\n");
  gets(s);
  n=strlen(s);
  ins(s,n);
  printf("%s\n",s);
}

```

```

}
ins(char s[],int n)
{ register int g,i,j;
  char temp;
  for(g=n/2;g>0;g/=2)
    for(i=g;i<n;i++)
      for(j=i-g;j>=0;j-=g)
        if(s[j]>s[j+g])
          { temp=s[j];
            s[j]=s[j+g];
            s[j+g]=temp;
          }
}

```

该程序执行结果如下所示:

A>exp15\_4

请输入数据元素序列:

68763432uytewgffGXTRL';.

'.23346678;FGLRTXefgtuwy

A>

(2)快速排序法 该法是由交换排序法派生出来的。其方法是,从要排序的数据中,任意选择一个称为比较数的数据元素,然后从两头开始,与比较数进行比较,把要排序的数据分为两部分:大于或等于比较数的放在一边,小于的放在另一边。例如,对于如下数据序列,按升序排序:

3 5 4 1 6 2

比较数选4,则比较交换过程为:

```

      ↗         ↘
3  5  4  1  6  2
(3<4,位置不变)

      ↗         ↘
3  2  4  1  6  5
(6>4,位置不变)

3  2  1  4  6  5
(5>4,位置不变)

```

接下来,再对比较数两边的子集重复上述过程。显然,这种排序法很适宜用递归算法。

〔练习 15.5〕 该练习是用快速排序法对数据元素进行升序排序的程序。其中,函数 quick() 为快速排序函数。

```

C>type exp15_5.c
#include <stdio.h>
#include <string.h>
#define MAX 1000

```

```

main()
{
    int i,n;
    char s[MAX],c;
    printf("请输入数据元素序列:\n");
    gets(s);
    n=strlen(s);
    quick(s,0,n-1);
    for(i=0;i<n;i++)
        putchar(s[i]);
    putchar('\n');
}

quick(item,left,right)
char * item;
int left,right;
{
    register int i,j;
    char m,t;
    i=left; j=right;
    m=item[(left+right)/2];
    do{
        while(item[i]<m&& i<right)
            i++;
        while(m<item[j]&& j>left)
            j--;
        if(i<=j)
        {
            t=item[i];
            item[i]=item[j];
            item[j]=t;
            i++;j--;
        }
    }while(i<=j);
    if(left<j) quick(item,left,j);/* 对左子集排序 */
    if(i<right) quick(item,i,right);/* 对右子集排序 */
}

```

该程序执行结果如下所示:

C>exp15\_5

请输入数据元素序列:

%t5jhgThggUTlio

%5ITTUggghhijot

### 15.3 结构数据的排序

以上,我们仅介绍了对字符型数据元素的排序,这主要是为了说明算法.在实际应用中,要排序

的数据元素有可能是字符串、C语言结构之类的结构数据类型。而快速排序法又是算法中最好最通用的排序方法。因此,在这里,我们就使用该方法,举例说明字符串、结构数据的排序问题。

(1)数据元素为字符串的数据的排序 简便可行的办法是不改变各字符串的实际物理位置,只调整指向各字符串的指针顺序。这就需要使用指向要排序的各字符串的指针数组。

〔练习 15.6〕 该练习就是使用快速排序法按升序对字符串进行排序的程序。

```
C>type exp15_6.c
#include (stdio.h)
#define MAX 10
char *p[]={
    "ENTE",
    "NEWB",
    "POIN",
    "FIRS",
    "COPY",
    "TYPE",
    "FORM",
    "LINK",
    "MASM",
    "TASM"
};

main()
{
    int i;
    qs(p,0,MAX-1);
    for(i=0;i<MAX;i++)
        printf("%s\n",p[i]);
}

qs(item,left,right)
char *item[];
int left,right;
{
    register int i,j;
    char *m,*t;
    i=left;
    j=right;
    m=item[(left+right)/2];
    do{
        while(strcmp(item[i],m)<0&&i<right)
            i++;
        while(strcmp(item[j],m)>0&&j>left)
            j--;
        if(i<=j){
```

```

        t=item[i];
        item[i]=item[j];
        item[j]=t;
        i++;
        j--;
    }
}while(i<=j);
if(left<j) qs(item,left,j);
if(i<right) qs(item,i,right);
}

```

该程序执行结果如下所示:

C>exp15\_6

```

COPY
ENTE
FIRS
FORM
LINK
MASM
NEWB
POIN
TASM
TYPE

```

C>

(2)数据元素为结构的数据的排序 从前面的介绍,大家已经知道,在C语言中,字符串是结构的特殊情况。即,字符串是同类型的基本数据类型的数据的序列,而结构则是由不同类型的基本数据类型的数据组成的。可见,如果不考虑这两种数据的组成成份,则它们的一个元素都是基本数据类型数据的序列。因此,字符串的排序方法同样适用于结构,只是在结构的排序中应使用结构指针或结构数组。

〔练习 15.7〕 该练习要排序的数据元素是结构,功能是按学员的成绩排出名次。

C>type a;exp15\_7.c

```

#include "stdio.h"
#include "stdlib.h"
#include "string.h"
struct student
{ char name[9];
  char city[7];
  char score[2];
}sl;
unsigned t;
void quick_disk(),swap_all_fields();
main()
{ FILE *fp;

```

```

int i,num;
t=sizeof(sl);
printf("请输入实际记录个数:");
scanf("%d",&num);
if((fp=fopen("sl.sc","rb+"))==0)
{ printf("不能打开文件,sl.sc\n");
  exit(0);
}
quick_disk(fp,0,num-1);
fclose(fp);
printf("排序完毕!");
}

void quick_disk(FILE *fp,int left,int right)
{ int i,j;
  char m[3],*get_score();
  i=left;j=right;
  strcpy(m,get_score(fp,(i+j)/2));
  do{
    while(strcmp(get_score(fp,i),m)<0&&i<right) i++;
    while(strcmp(get_score(fp,j),m)>0&&j>left) j--;
    if(i<=j)
    { swap_all_fields(fp,i,j);
      i++;j--;
    }
  }while(i<=j);
  if(left<j) quick_disk(fp,left,j);
  if(i<right) quick_disk(fp,i,right);
}

void swap_all_fields(FILE *fp,int i,int j)
{ char a[sizeof(sl)],b[sizeof(sl)];
  fseek(fp,(t+2)*i,0);
  fread(a,t,1,fp);
  fseek(fp,(t+2)*j,0);
  fread(b,t,1,fp);
  fseek(fp,(t+2)*j,0);
  fwrite(a,t,1,fp);
  fseek(fp,(t+2)*i,0);
  fwrite(b,t,1,fp);
}

char *get_score(FILE *fp,int r)
{ struct student *p;
  p=&sl;
  fseek(fp,r*(t+2),0);
  fread(p,t,1,fp);

```



```
return(sl.score);
```

该程序执行情况如下所示:

```
C>type sl.sc
```

卢晓光 天津市 67

马玉林 北京市 78

刘海峰 河北省 90

张坚强 上海市 60

王韦平 甘肃省 50

李志琴 南京市 80

```
C>exp15_7
```

请输入实际记录个数:6

排序完毕!

```
C>type sl.sc
```

王韦平 甘肃省 50

张坚强 上海市 60

卢晓光 天津市 67

马玉林 北京市 78

李志琴 南京市 80

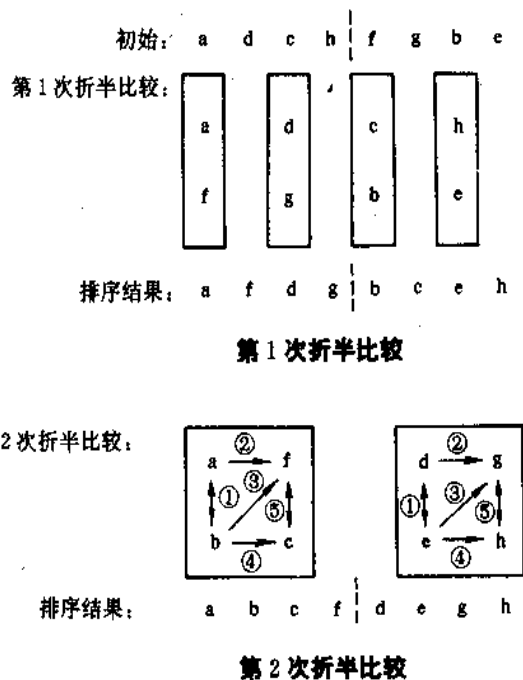
刘海峰 河北省 90

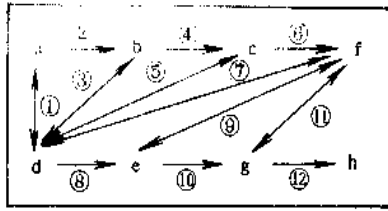
#### 15.4 归并排序法

这是对文件中的数据进行排序的方法,该法是由交换法演变来的。该法的适用条件是:

- 需要三个文件;
- 数据元素的个数为  $2^n$  个。

这里以文本文件为例来说明其排序方法。设文件的内容为 adchfgbe,则排序过程如图 15.1 所示:





最终排序结果: a b c d e f g h

(C)第3次折半比较

图 15.1 归并排序过程

显然,对于数据元素个数为  $2^n$  的文件,其折半比较次数为  $n$  次,而每次相互比较的元素的个数为  $2^1, 2^2, 2^3, \dots$ 。

〔练习 15.8〕 该练习是使用命令行:

exp15\_8 文件名

即可对内容为  $2^n$  个字母的文件进行排序的程序。

```
C>type exp15_8.c
#include "stdio.h"
void merge();
main(argc,argv)
int argc;
char *argv[];
{ FILE *fp1,*fp2,*fp3;
  int lenth=0;
  if(argc!=2)
    printf("please add can");
  if((fp1=fopen(argv[1],"r+"))==0)
  { printf("不能打开文件:%s\n",argv[1]);
    exit(1);
  }
  else if((fp2=fopen("sort1","w+"))==0)
  { printf("不能打开文件 2.\n");
    exit(1);
  }
  else if((fp3=fopen("sort2","w+"))==0)
  { printf("不能打开文件 3.\n");
    exit(1);
  }
  else{ while(getc(fp1)!=EOF)
    length++;
    rewind(fp1);
    merge(fp1,fp2,fp3,length);
    fclose(fp1);
    fclose(fp2);
    fclose(fp3);
```

```

    }
}
/* 归并排序函数 */
void merge(fp1,fp2,fp3,count)
FILE *fp1, *fp2, *fp3;
int count;
{
    register int n,t,q,k,j;
    char x,y;
    for(n=1;n<count;n=n*2)
    /* n 为比较次数 */
    { for(t=0;t<count/2;t++)
        putc(getc(fp1),fp2);
        for(;t<count;t++)
            putc(getc(fp1),fp3);
        exit(0);
    /* 前半放在 fp2,后半放在 fp3 */
    reset(fp1,fp2,fp3);
    for(q=0;q<count/2;q+=n)
    { x=getc(fp2);
        y=getc(fp3);
        for(j=k=0;;)
        /* 小者放 fp1,同时取小者行的下个元素再比较 */
        { if(x<y)
            { putc(x,fp1);
                j++;
                if(j<n) x=getc(fp2);
                else break;
            }
            else
            { putc(y,fp1);
                k++;
                if(k<n) y=getc(fp3);
                else break;
            }
        }
    }
    /* 比较后的大者放 fp1 */
    if(j<n)
    { putc(x,fp1);
        j++;
    }
    if(k<n)
    { putc(y,fp1);
        k++;
    }
}

```

```

    }
    /* 把比较后的剩余元素放 fp1 */
    for(i;j<n;j++)
        putc(getc(fp2),fp1);
    for(k<n;k++)
        putc(getc(fp3),fp1);
    }
    reset(fp1,fp2,fp3);
}
}
/* 恢复文件位置指示器到文件头 */
reset(fp1,fp2,fp3)
FILE *fp1, *fp2, *fp3,
{ rewind(fp1);
  rewind(fp2);
  rewind(fp3);
}

```

该程序对含有 2<sup>n</sup> 个数据元素的文件的排序情况如下所示:

```

C>type d1.d
asdf
C>exp15-8 d1.d
C>type d1.d
adfa
C>type d2.d
qwerpoi
C>exp15-8 d2.d
C>type d2.d
eiopqruw
C>type d3.d
zxcvdfhdkjhuytr
C>exp15-8 d3.d
C>type d3.d
cddfhjhklrtuvwxyz
C>

```

1:

## 15.5 检索

检索是信息处理中最常见的基本技术。所谓检索就是从众多的数据元素中找出符合某一条件的数据元素。这里所说的条件称作检索条件。检索到所要找的数据元素称作检索成功,否则称为检索失败。下面介绍常用的检索方法。

(1)顺序检索 所谓顺序检索就是用待查的关键字值与线性表中的各数据元素逐个顺序比较查找的方法。该法又叫线性检索。

〔练习 15.9〕 该练习是从给定通讯录中按姓名查找指定人员的程序。

```
C>type exp15-9.c
```

```

#include "stdio.h"
#include "string.h"
struct student
{ char name[9];
  char city[7];
  char score[2];
};
unsigned int t,i;
long l;
main()
{
  int flag==0;
  char * get_name();
  int i,record_num;
  char kv[9],fname[9];
  FILE * fp;
  l==sizeof(s);
  printf("请输入文件名:");
  scanf("%s",fname);
  printf("请输入记录个数:");
  scanf("%d",&record_num);
  if((fp=fopen(fname,"r+"))==0)
  { printf("文件,%s 没有打开! \n",fname);
    exit(1);
  }
  printf("请输入姓名:");
  scanf("%s",kv);
  for(i=strlen(kv);i<9;i++)
  kv[i]=' ';
  kv[i]='\0';
  for(t=0;t<record_num;++t)
  if(! strcmp(kv,get_name(fp,t)))
  { fseek(fp,(l+2)*t,0);
    for(i=0;i<l;i++)
    printf("%c",getc(fp));
    printf("\n");
    flag=1;
  }
  if(flag==0)
    printf("查无此人!! \n");
}
char * get_name(fp,r)
FILE * fp;
unsigned int r;

```

```

/ struct student *p;
p=&sl;
fseek(fp,r*(1+2),0);
fread(p,9,1,fp);
return sl.name;
}

```

C>type sl.sc

王伟平 甘肃省 50

张坚强 上海市 60

卢晓光 天津市 67

马玉林 北京市 78

李志琴 南京市 80

刘海峰 河北省 90

C>exp 15-9

请输入文件名:sl.sc

请输入记录个数:6

请输入姓名:马玉林

马玉林 北京市 78

C>exp 15-9

请输入文件名:sl.sc

请输入记录个数:6

请输入姓名:mayulin

查无此人!!

(2)二分法检索 这是仅适合于对已排好序的数据进行检索的方法。该法是先由中间元素与关键字比较,若相等,则找到;若关键字的值大于中间元素的值,则用右半部分的中间元素再与关键字比较;否则,用左半部分的中间元素与关键字比较。如此重复,直到找到为止。检索过程可按图 15.2 所示进行。图中①②③为测试次数。

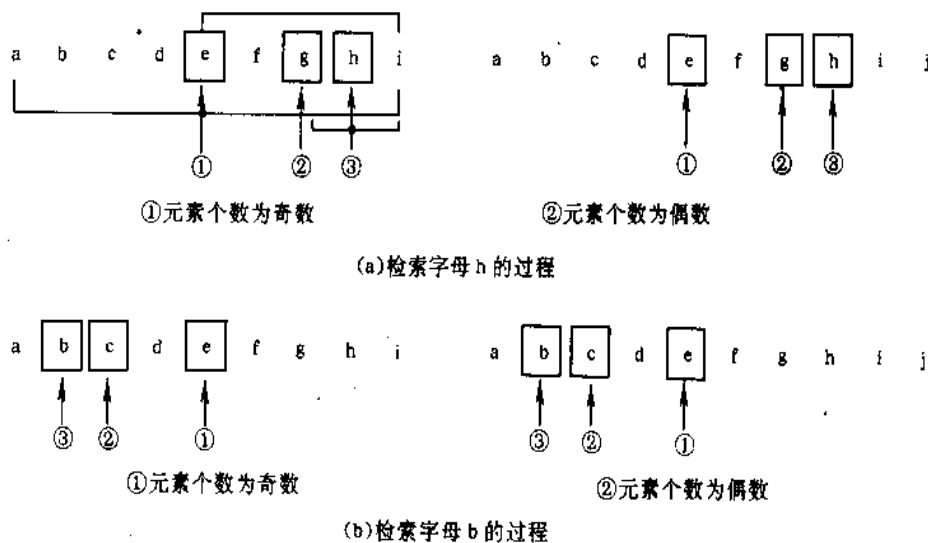


图 15.2 二分法检索过程

〔练习 15.10〕 该练习是用二分法对字符型数据元素进行检索的过程。其中 bs() 函数为检索函数。

```
A>type exp15_10.c
#define MAX 80
main()
{ char *it,key1,bs(),x;
  int count;
  it=(char *)malloc(MAX);
  printf("请输入数据:\n");
  scanf("%s",it);
  printf("请输入要查找的数据元素:\n");
  scanf("%c",&key1);
  count=strlen(it);
  x=bs(it,count,key1);
  if(x==-1)
    printf("not found");
  else
    printf("找到数据元素:%c",x);
}

char bs(it,count,key)
char *it;
int count;
char key;
{ int low,high,mid;
  low=0;high=count-1;
  while(low<=high)
  {
    mid=(low+high)/2;
    if(key<it[mid])
      high=mid-1;
    else if(key>it[mid])
      low=mid+1;
    else
      return it[mid];
  }
  return -1;
}
```

该程序执行情况如下:

```
A>exp 15-10
请输入数据:
abcdef
请输入要查找的数据元素:
a
找到数据元素:a
```

```
A>exp15_10
请输入数据:
abcdef
请输入要查找的数据元素:
f
找到数据元素:f
A>exp15_10
请输入数据:
abcdef
请输入要查找的数据元素:
c
找到数据元素:c
A>
```



## 第十六章 数据结构及其实现

数据结构是研究逻辑上如何由基本数据元素构造出复合数据,以及物理上如何实现的一门计算机学科。数据结构在计算机科学与工程上得到广泛应用,如表 16.1 所示。

表 16.1 数据结构的应用

应用领域	数据结构
编译系统	栈、散列表、语法树
操作系统	循环队列、存储管理表、目录树
数据库管理系统	散列表、链表、索引树
人工智能领域	广义表、集合、搜索树、向量图

C 语言号称系统描述语言。它成功地描述了 UNIX 操作系统和 FoxBASE 数据库管理系统。这足以证明,其处理数据结构的能力是相当强的。本章重点讨论常用数据结构及其实现的 C 程序。

### 16.1 队列

(1) 队列的结构及其访问方式 队列是数据的一种线性结构。其物理空间可用动态分配函数 malloc() 在内存的自由区域(堆)内设置。它是靠两个指针,即头指针和尾指针,进行数据元素的加入和移出的,如图 16.1 所示。

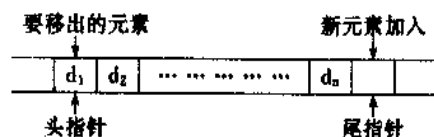


图 16.1 队列的结构与操作

队列的访问方式严格遵循先进先出(FIFO)原则,不允许随机访问,如图 16.2 所示。其中, Rp 为头指针, Sp 为尾指针。

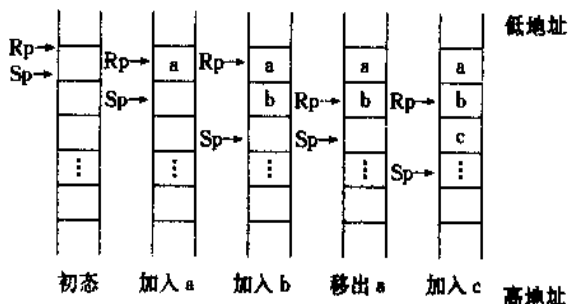


图 16.2 队列操作过程

(2) 队列设置和操作的程序设计 这里给出以字符串为元素的加入元素操作函数和移出元素

操作函数。

①加入元素操作函数 qstore()

```
void qstore(q)
char *q;
{ if(sp==M)
    { printf("空间满\n")
      return;
    }
  p[sp++] = q;
}
```

②移出元素操作函数 qretrieve()

```
char *qretrieve()
{ if(rp==sp)
    { printf("已无元素\n");
      return NULL;
    }
  rp++;
  return p[rp-1];
}
```

【练习 16.1】 该练习是用队列所设计的一个备忘录程序。该程序有录入、存储、装入内存、列表、删除、返回诸功能，分别由函数 enter()、save()、load()、list()、delete()、exit()来实现。源程序及其执行结果如下所示。可以说，该程序有实用价值。注意，save()函数中出现的'C'作为每条备忘录的结尾标志，而'\0'是整个备忘录的结尾标志。

```
C>type exp16_1.c
#include "stdio.h"
#include "stdlib.h"
#include "string.h"
#define M 10
char *p[M], *qretrieve();
int sp, rp;
void enter(), qstore(), list(), delete();
void save(), load();
main()
{ char c, s[80];
  register int t;
  for (t=0; t<M; t++)
    p[t] = '\0';
  sp=rp=0;
  for(;;)
  { switch(menu_select())
    { case 1: enter();
      break;
      case 2: save();
```

```

        break;
    case 3:load();
        break;
    case 4:list();
        break;
    case 5:delete();
        break;
    case 0:exit(0);
}
}
}
menu_select()
{ char s[80];
  int c;
  printf("1----备忘录录入  \n");
  printf("2----备忘录存盘  \n");
  printf("3----打开备忘录  \n");
  printf("4----备忘录列表  \n");
  printf("5----删完成事件  \n");
  printf("0----返回      \n");
  do{
    printf("\n 请输入选择号:");
    gets(s);
    c=atoi(s);
  }while(c<0||c>7);
  return c;
}
void enter()
{ char s[256], *p;
  do{ printf("请输入第%d 件事:",sp+1);
    gets(s);
    if(*s==0)break; /* 未录入,结束 */
    p=malloc(strlen(s)+1);
    if(! p)
      { printf("空间不够!!");
        return;
      }
    strcpy(p,s);
    p[strlen(s)]='\0';
    if(*s) qstore(p);
  }while(*s);
}
void save()
{

```

```

FILE *fp;
int i;
if((fp=fopen("bwl.txt","w+"))==NULL)
{
    printf("文件打开错误!!");
    exit(0);
}
for(i=rp;i<sp;i++)
{
    fwrite(p[i],strlen(p[i]),1,fp);
    fwrite('C',1,1,fp);
}
fwrite('\0',1,1,fp);
fclose(fp);
}

void load()
{
    FILE *fp;
    int i,k;
    char j;
    if((fp=fopen("bwl.txt","r+"))==NULL)
    {
        printf("文件打开错误!!");
        exit(0);
    }
    for(i=0;i++)
    {
        p[i]=(char *)malloc(100);
        j=fgetc(fp);
        if(j=='\0')
            break;
        else
        {
            p[i][0]=j;
            k=1;
            while((j=fgetc(fp))!= 'C')
                p[i][k++]=j;
            p[i][k]='\0';
        }
    }
    rp=0;
    sp=i;
}

void list()

```

```

register int t;
for (t=rp;t<sp;t++)
    printf("第%d 件事为: %s\n",t+1,p[t]);
}

void delete()
{
    if(! (qretrieve()))
        return;
}

void qstore(q)
char *q;
{ if(sp==M)
    { printf("空间满!! \n");
      return;
    }

    p[sp]=q;
    sp++;
}

char *qretrieve()
{ if(rp==sp)
    { printf("事件全部处理完毕!! \n");
      return NULL;
    }

    rp++;
    return p[rp-1];
}

```

A>exp16\_1

1----备忘录录入  
 2----备忘录存盘  
 3----打开备忘录  
 4----备忘录列表  
 5----删完成事件  
 0----返回

请输入选择号:3

1----备忘录录入  
 2----备忘录存盘  
 3----打开备忘录  
 4----备忘录列表  
 5----删完成事件  
 0----返回

请输入选择号:4

第1件事为:八点上课.

第2件事为:十点开会.

第3件事为:十二点会客.

第4件事为:两点看电影.

1----备忘录录入

2----备忘录存盘

3----打开备忘录

4----备忘录列表

5----删完成事件

0----返回

请输入选择号:1

请录入第5件事:五点参观.

请录入第6件事:

1----备忘录录入

2----备忘录存盘

3----打开备忘录

4----备忘录列表

5----删完成事件

0----返回

请输入选择号:5

1----备忘录录入

2----备忘录存盘

3----打开备忘录

4----备忘录列表

5----删完成事件

0----返回

请输入选择号:4

第1件事为:十点开会.

第2件事为:十二点会客.

第3件事为:两点看电影.

第4件事为:五点参观.

1----备忘录录入

2----备忘录存盘

3----打开备忘录

4----备忘录列表

5----删完成事件

0----返回

请输入选择号:2

1----备忘录录入

2----备忘录存盘

3----打开备忘录

4----备忘录列表

5----删完成事件

0----返回

请输入选择号:3

- 1----备忘录录入
- 2----备忘录存盘
- 3----打开备忘录
- 4----备忘录列表
- 5----删完成事件
- 0----返回

请输入选择号:4

- 第2件事为:十点开会.
- 第3件事为:十二点会客.
- 第4件事为:两点看电影.
- 第5件事为:五点参观.

- 1----备忘录录入
- 2----备忘录存盘
- 3----打开备忘录
- 4----备忘录列表
- 5----删完成事件
- 0----返回

请输入选择号:0

## 16.2 堆栈

### (1)堆栈的结构及其访问方式

堆栈也是数据的一种线性结构。其物理空间可由数组提供,也可用动态分配函数 `malloc()` 在内存的自由区域(堆)内设置。其访问形式有两种:一是按后进先出(LIFO)的原则进行;二是间接寻址,随机访问。这里,只介绍第一种访问方式,第2种访问方式已在12章介绍过。访问操作有两种:即压栈和退栈,其操作过程如图16.3所示。

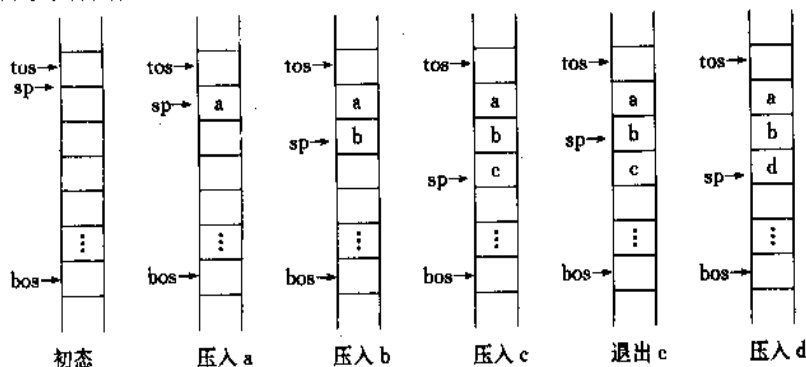


图 16.3 LIFO 方式的压栈与退栈

### (2)堆栈程序设计

①由数组提供栈区的堆栈操作程序,如练习16.2所示。

#### (练习 16.2)

```
01 type a:exp16_2.c
02 #define M 10
03 int stack[M];
```

```

int tos=0; /* 栈顶指针 */
void push(i) /* 压栈函数 */
char i;
{ if(tos>=M)
{ printf("栈区满!");
return;
}
tos++;
stack[tos]=i;
}
int pop() /* 退栈函数 */
{ if(tos<0)
{ printf("栈区空!");
return;
}
tos--;
return(stack[tos]);
}
main()
{ int i;
char x;
for(x='A';x<='A'+10;x++)
push(x);
for(i=1;i<=M;i++)
printf("%c ",stack[i]);
printf("\n");
tos++;
for(i=1;i<=M;i++)
{ x=pop();
printf("%c ",x);
}
printf("\n");
}

```

C>exp16-2

```

A B C D E F G H I J
J I H G F E D C B A

```

②由动态存储分配函数 malloc() 在内存自由区域设置栈区的程序,如练习 16.3 所示。

### 〔练习 16.3〕

C>type a,exp16-3.c

```

#define M 10
int *p; /* 指向栈区 */
int *tos; /* 指向栈顶 */
int *bos; /* 指向栈底 */
void push();

```



```

main()
{ int a,i;
  char s[80];
  a=0;
  p=(int *)malloc(M*sizeof(int));
  if(! p)
  { printf("分配失败! \n");
    exit(1);
  }
  tos=p;
  bos=p+M+1;
  for(i=0;i<M;i++)
  { push(i);
    printf("%d\n", *p);
  }
  p++;
  for(i=0;i<=M;i++)
    a+=pop();
  printf("\n%d\n",a);
}

void push(i) /* 压栈函数 */
int i;
{ if(p>bos)
  { printf("栈区溢出!");
    return;
  }
  p++;
  *p=i;
}

pop() /* 退栈函数 */
{ if(p<tos)
  { printf("栈区空!");
    return 0;
  }
  p--;
  return *p;
}

```

该程序执行结果如下:

C>exp16\_3

0  
1  
2  
3  
4

5  
6  
7  
8  
9

45  
C>

### 16.3 单链表

(1)什么是单链表 所谓单链表是指每一个数据元素都有一个链指针用来指向另一个数据元素的数据结构。在这样的数据结构中,每一个数据元素都由两部分组成:一部分为信息域,另一部分是链指针域。这样的数据元素可用C的结构数据来设计。以通讯录为例,用单链表来设计,通讯录的每条记录便是一个数据元素,其对应的结构数据(struct 类型)如下所示:

```
struct address{  
    char name[9];  
    char address[7];  
    char zip[7];  
    struct address * next;  
};info;
```

为了在第一个元素前面加入新元素或删除第一个元素时,头指针的值不变化,一般还要在第一个数据元素的前面加一个所谓的头元素,而最末一个数据元素没有后继元素,故其链指针域可设为空域。这样,单链表的结构便如图 16.4 所示。

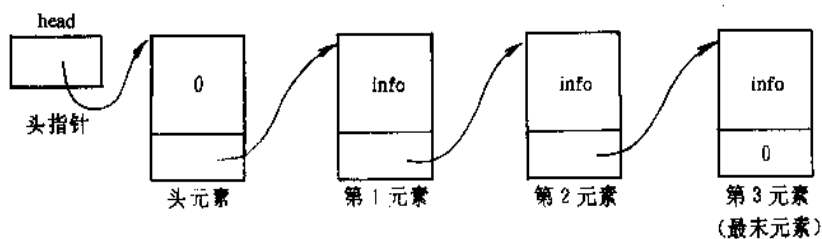
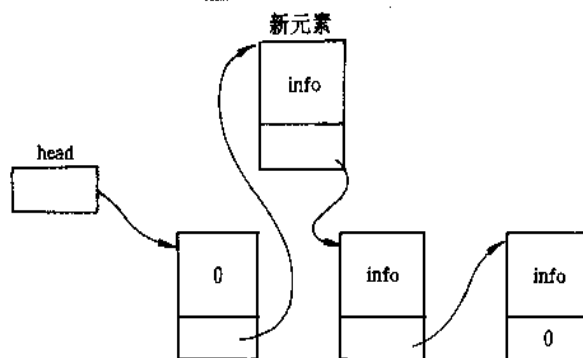


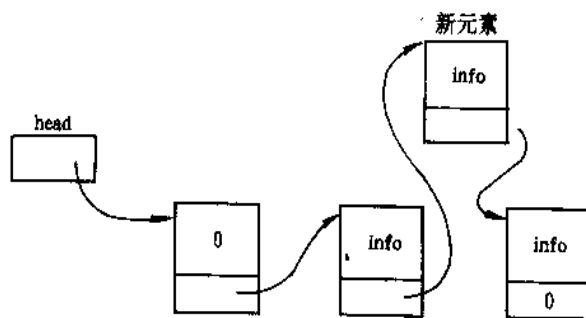
图 16.4 单链表数据结构

链表的特点是,不用移动数据元素的物理位置,便可进行数据元素的删除、插入,操作方便,故在管理信息系统中广为应用。

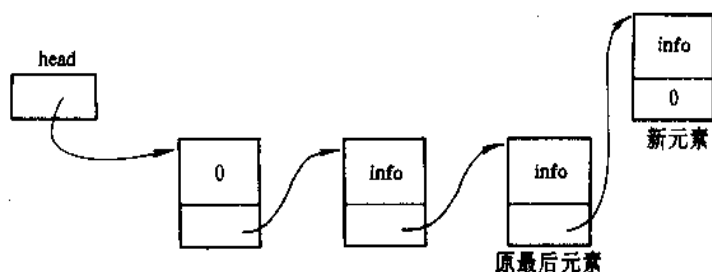
(2)单链表的构造方法 有三种,如图 16.5 所示。



(a)新元素作为第一元素



(b)新元素插在链表中间



(c)新元素作为最末元素

图 16.5 单链表结构的构造方法

按照图 16.5(c)所示的方法构造单链表的函数取名为 `slstore()`,其函数体如下所示:

```
void slstore(i)
struct address * i;
{
    if(last=NULL)
        last=i;
    else
        last->next=i;
        i->next=NULL;
        last=i;
}
```

其中, `i` 为指向新元素的结构指针, `last` 为指向原链表最后一个元素的结构指针。

(3)删除数据元素的方法 亦有三种,如图 16.6 所示。

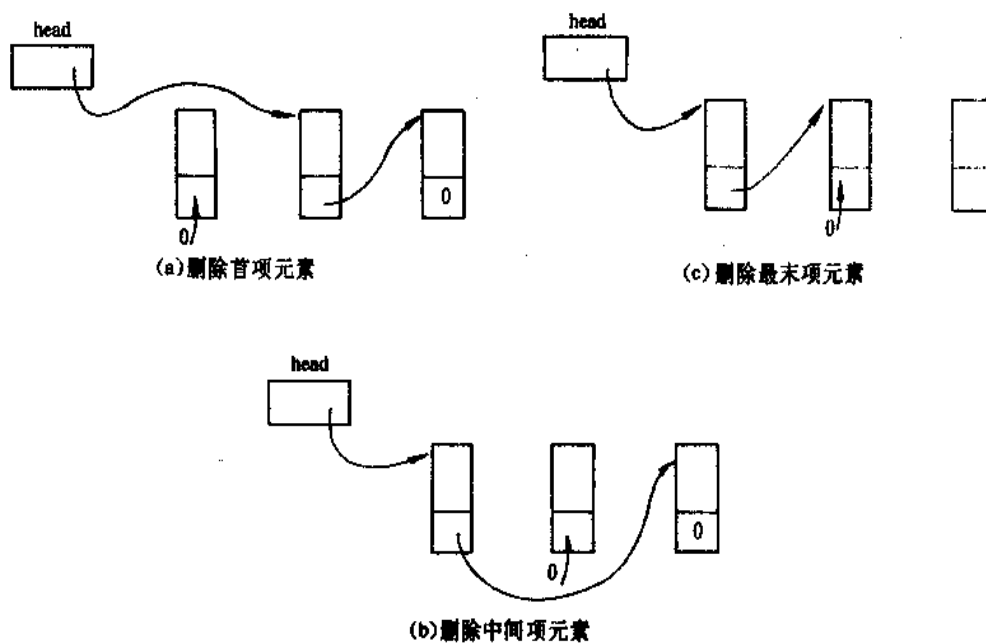


图 16.6 从单链表中删除数据元素的方法

删除一个数据元素的函数取名为 `sdelete()`, 其参数 `p`、`i`、`top` 分别是指向要删去的数据元素的前一项数据元素、要删去的数据元素和链表的第一项数据元素的结构指针, 其函数体如下所示:

```
struct address * sdelete(p,i,top)
struct address * p;
struct address * i;
struct address * top;
{ if(p)
    p->next=i->next;
  else
    top=i->next;
  return(top);
}
```

(4) 录入功能的实现 录入功能由函数 `enter()` 实现, 其函数体如下所示:

```
struct address * enter()
{ void inputs();
  struct address * a;
  a=(struct address *) malloc(sizeof)(info));
```

```

if(! a)
{ printf("\n 内存不够! \n");
  exit(0);
}
inputs("\t 姓名:", a->name, 8);
if(! *a->name)
  return(NULL);
inputs("\t 地址:", a->address, 6);
inputs("\t 邮编:", a->zip, 6);
return(a);
}

```

其中函数 inputs() 为提示用户按规定长度输入字段的函数, 其函数体如下所示:

```

void inputs(a,b,c)
char * a;
char * b;
int c;
{ char s[80];
  printf("%s", a);
  do{ gets(s);
    }while(strlen(s)>c);
  strcpy(b,s);
  b[c]='\0';
}

```

(5) 列表函数 该函数的函数体如下所示:

```

void display(top)
struct address * top;
{ while(top)
  { printf("%-9s %-7s %-7s\n", top->name, top->address, top->zip);
    top=top->next;
  }
}

```

(6) 检索函数 该函数的函数体如下所示:

```

struct address * search(top,n)
struct address * top;
char * n;
{ while(top)
  { if(! strcmp(n, top->name))
    return(top);
    top=top->next;
  }
  return(NULL);
}

```

(7)插入函数 该函数取名为 sl\_store(i,top),其功能是把结构指针 i 所指向的记录,按姓名字段的编码值大小,添加到结构指针 top 所指定的通讯录的适当位置。其函数体如下所示:

```
struct address * sl_store(i,top)
struct address * i;
struct address * top;
{ struct address * old, * start1;
  start1=top;
  if(last==NULL)
    { i->next=NULL;
      last=i;
      return(i);
    }
  old=NULL;
  while(top)
    { if(strcmp(top->name,i->name)<0)
        { old=top;
          top=top->next;
        }
      else
        { if(old)
            { old->next=i;
              i->next=top;
              return(start1);
            }
          i->next=top;
          return(i);
        }
    }
  last->next=i;
  i->next=NULL;
  last=i;
  last=i;
  return(start1);
}
```

(8)菜单函数 该函数取名为 menu(),其功能是在屏幕终端上显示出菜单,供用户选择。该函数体如下所示:

```
int menu()
{ int c;
  char s[30];
  printf("\n");
  printf("#1,-----录入\n");
  printf("#2,-----添加\n");
```

```

printf("3:-----显示\n");
printf("4:-----查询\n");
printf("5:-----删除\n");
printf("6:-----返回\n");
do{ printf("请选择:");
    gets(s);
    c=atoi(s);
}while(c<0 || c>6);
return(c);
}

```

(9)应用举例 用主函数 main()把这些函数连接起来,即调用它们,就可以实现通讯录的录入、添加、显示、查询、删除等功能。

下面,举例说明由这些函数所组成的通讯录程序的执行情况。

A>exp16\_4

1:-----录入

2:-----添加

3:-----显示

4:-----查询

5:-----删除

6:-----返回

请选择:1

姓名:高大全

地址:泰安市

邮编:321000

姓名:贾方

地址:北京市

邮编:100000

姓名:王宝

地址:上海市

邮编:200100

姓名:

1:-----录入

2:-----添加

3:-----显示

4:-----查询

5:-----删除

6:-----返回

请选择:3

高大全 泰安市 321000

贾方 北京市 100000

王宝 上海市 200100

1:-----录入

- 2:-----添加
- 3:-----显示
- 4:-----查询
- 5:-----删除
- 6:-----返回

请选择:2

姓名:张飞  
地址:广州市  
邮编:140000

- 1:-----录入
- 2:-----添加
- 3:-----显示
- 4:-----查询
- 5:-----删除
- 6:-----返回

请选择:3

高大全 泰安市 321000  
贾方 北京市 100000  
王宝 上海市 200100  
张飞 广州市 140000

- 1:-----录入
- 2:-----添加
- 3:-----显示
- 4:-----查询
- 5:-----删除
- 6:-----返回

请选择:4

请输入姓名:安长正  
无此记录!!

- 1:-----录入
- 2:-----添加
- 3:-----显示
- 4:-----查询
- 5:-----删除
- 6:-----返回

请选择:4

请输入姓名:高大全  
高大全 泰安市 321000

- 1:-----录入
- 2:-----添加
- 3:-----显示
- 4:-----查询



5:-----删除

6:-----返回

请选择:5

请输入姓名:王宝

1:-----录入

2:-----添加

3:-----显示

4:-----查询

5:-----删除

6:-----返回

请选择:3

高大全 泰安市 321000

贾方 北京市 100000

张飞 广州市 140000

1:-----录入

2:-----添加

3:-----显示

4:-----查询

5:-----删除

6:-----返回

请选择:6

A>

#### 16.4 双链表

(1)什么是双链表 双链表是指由数据元素加上指向前项和指向后项两个指针组成的数据结构,如图 16.7 所示。可见,双链表有逆向操作功能;且一条链损坏后,可用另一条链修复该链表。

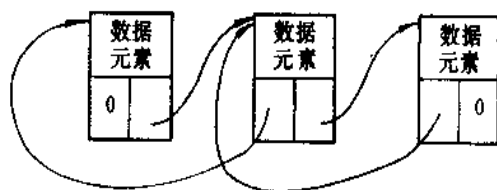


图 16.7 双链表数据结构

C 语言的结构数据亦可作双链表的一个数据元素。例如,为建立通信录而设置如下结构类型及其变量。

```
struct address{  
    char name[9];  
    char city[7];  
    char zip[6];  
    struct address * next;  
    struct address * prior;  
} list_entry;
```

我们知道, `address` 为该结构数据的类型名, 它是由三个字符型数组, 两个指向其本身的结构指针组成的复合数据。其中三个字符型数组是构成一条记录的三个字段; 指针 `next` 是指向下一条记录的指针; 指针 `prior` 是指向前一条记录的指针。 `list_entry` 为结构变量名, 其内容就是一条实在的记录。

(2) 双链表的构成方法 有三种, 如图 16.8 所示。

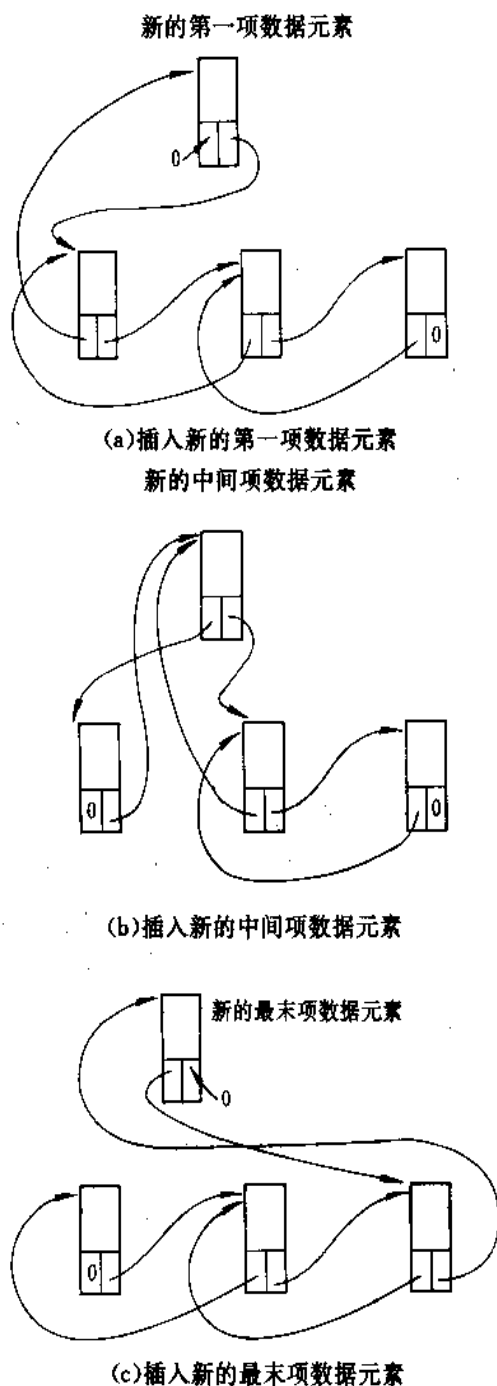


图 16.8 双链表数据结构的构造方法

(3) 双链表删除数据元素的方法 亦有三种, 如图 16.9 所示。

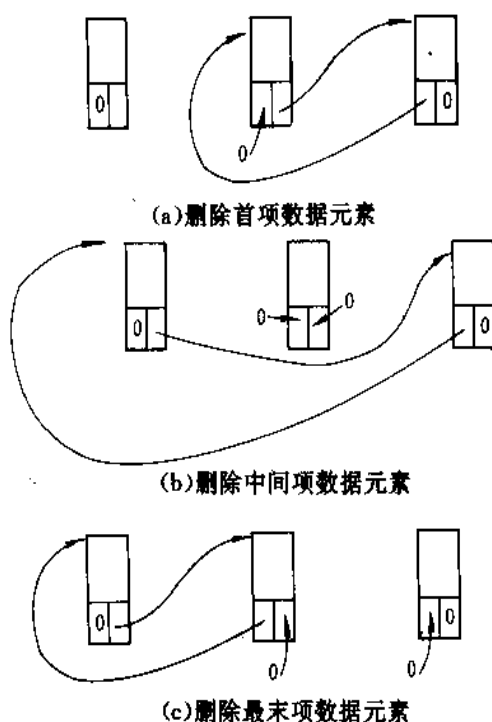


图 16.9 双链表删除数据元素的方法

(4)应用举例 使用双链表建立通讯录的程序就是一个实用的例子。要想使该程序具有录入、删除、列表、检索、存盘、加载,以及返回到操作系统等功能,就要建立相应的函数。下面给出这些函数。

①录入函数 该函数取名为 enter(),其功能是从键盘输入一条记录,并按顺序存放到链表中。函数体如下:

```
void enter()
{
    int i;
    char *left();
    struct address *info, *dls_store();
    for(i=0;i<M;i++){
        info=(struct address *)malloc(sizeof(list_entry));
        if(! info)
        {
            printf("记录超量!!!");
            return;
        }
        inputs("姓名:",info->name,9);
        if(! info->name[0])
            break;
        inputs("地址:",info->city,7);
        inputs("邮编:",info->zip,6);
        start=dls_store(info,start);
    }
}
```

其中,函数 inputs()和 dls\_store()分别是输入一条记录和存放一条记录的函数,它们的函数体分别是:

• 记录输入函数 inputs()

```
inputs(prompt,s,count)
char *prompt;
char *s;
int count;
{
    int j;
    char p[200];
    do{
        printf("%s",prompt);
        gets(p);
        if(strlen(p)>count) printf("\n 输入超长,请重输\n");
        }while (strlen(p)>count);
    if(strlen(p)<count&&strlen(p)>0)
    {
        for(j=strlen(p);j<count;j++)
            p[j]=' ';
        p[j]='\0';
    }
    strcpy(s,p);
}
```

• 记录存放函数 dls\_store()

```
struct address *dls_store(i,top)
struct address *i;
struct address *top;
{
    struct address *old,*p;
    if(last==NULL)
    {
        i->next=NULL;
        i->prior=NULL;
        last=i;
        return(i);
    }
    p=top;
    old=NULL;
    while(p){
        if(strcmp(p->name,i->name)<0)
        {
            old=p;
            p=p->next;
        }
    }
}
```

```

    }
    else{
        if(p->prior)
        {
            p->prior->next=i;
            i->next=p;
            i->prior=p->prior;
            p->prior=i;
            return(top);
        }
        i->next=p;
        i->prior=NULL;
        p->prior=i;
        return(i);
    }
}
old->next=i;
i->next=NULL;
i->prior=old;
last=i;
return(start);
}

```

②检索函数 取名为 search(),其功能是根据输入的姓名,查找本条记录,函数体如下;

```

void search()
{
    int i;
    char name[10];
    struct address *info, *find();
    printf("输入姓名:");
    gets(name);
    for(i=strlen(name);i<9;i++)
        name[i]=' ';
    name[i]='\0';
    if(info=find(name))
        display(info);
}

```

其中,函数 find()的功能是按姓名字段查找要找的记录,函数体如下:

```

struct address *find(char *name)
{
    struct address *info;
    info=start;
    while(info)
    {
        if(! strcmp(left(9,name),left(9,info->name)))

```

```

    return(info);
    info=info->next;
}
printf("查无此人! \n");
return(NULL);
}

```

其中,函数 left()的功能是从第 2 个参数所指定的字符串中取前几个字符,所取的个数由第 1 参数给出。函数体如下:

```

char * left(int a,char * b)
{
    char * c;
    int i;
    if(a>strlen(b))
    {
        printf("input error \n");
        exit(0);
    }
    c=malloc(a+1);
    for(i=0;i<a;i++)
    c[i]=b[i];
    c[i]='\0';
    return c;
}

```

函数 display()的功能是显示一条记录,函数体如下:

```

void display(info)
struct address * info;
{
    chat * left();
    printf("%s ",left(9,info->name));
    printf("%s ",left(7,info->city));
    printf("%s\n",left(6,info->zip));
}

```

③删除函数 取名为 delete(),其功能是删除一条记录,函数体如下:

```

void delete()
{
    int i;
    struct address * info,* find();
    char s[80];
    printf("输入姓名:");
    gets(s);
    for(i=strlen(s);i<9;i++)
        s[i]=' ';
    s[i]='\0';
}

```

```

info=find(s);
if(info)
{
    if(start==info)
    {
        start=info->next;
        if(start) start->prior=NULL;
        else last=NULL;
    }
    else
    {
        info->prior->next=info->next;
        if(info!=last)
            info->next->prior=info->prior;
        else
            last=info->prior;
    }
    free(info);
}
}

```

④列表函数 取名为 list(),其功能是把整个通讯录显示出来,函数体如下:

```

list()
{
    register int t;
    struct address *info;
    info=start;
    while(info)
    {
        display(info);
        info=info->next;
    }
    printf("\n\n");
}

```

⑤存盘函数 取名为 save(),其功能是把通讯录存到磁盘文件 txll 中,函数体如下:

```

void save()
{
    register int t;
    struct address *info;
    FILE *fp;
    if((fp=fopen("txll","wb"))==NULL)
    {
        printf("文件不能打开!!");
        exit(1);
    }
}

```

```

printf("\n");
info=start;
while(info)
{
    fwrite(info,sizeof(struct address),1,fp);
    info=info->next;
}
fclose(fp);
}

```

⑥加载函数 所谓加载就是把文件从磁盘中取出,放到内存中,该函数取名为load(),函数体如下:

```

void load()
{
    register int t;
    struct address * info, * temp=NULL;
    FILE * fp;
    if((fp=fopen("txt1","rb"))==NULL)
    {
        printf("文件不能打开! \n");
        exit(1);
    }
    while(start)
    {
        info=start->next;
        free(info);
        start=info;
    }
    printf("\n");
    start=(struct address *)malloc(sizeof(struct address));
    if(! start)
    {
        printf("\n");
        return;
    }
    info=start;
    while(! feof(fp))
    {
        if(! fread(info,sizeof(struct address),1,fp))
            break;
        info->next=(struct address *)malloc(sizeof(struct address));
        if(! info->next)
        {
            printf("\n");
            return;
        }
    }
}

```



```

    }
    info->prior=temp;
    temp=info;
    info=info->next;
}
temp->next=NULL;
last=temp;
start->prior=NULL;
fclose(fp);
}

```

⑦菜单函数 该函数取名为 menu\_select(),其功能是显示菜单,供用户选择,函数体如下:

```

menu_select()
{
    char s[80];
    int c;
    printf("\n[记录操作菜单]\n");
    printf("    1. 录入\n");
    printf("    2. 删除\n");
    printf("    3. 列表\n");
    printf("    4. 检索\n");
    printf("    5. 存盘\n");
    printf("    6. 加载\n");
    printf("    7. 返回\n");
    do{
        printf("\n 请输入菜单命令:");
        gets(s);
        c=atoi(s);
    }while(c<0||c>7);
    return(c);
}

```

⑧执行情况 编写主函数 main(),用主函数调用这些函数,就可以实现上述各种功能。

设由上述函数所组成的通讯录程序,其文件名为 exp16\_5.c,存放记录的磁盘文件 txll 的内容是:

```
C>type txll
```

```
刘海发 天津市 300150
```

```
马玉林 泰安市 210350
```

则该程序的执行情况如下所示:

```
C>exp16_5
```

```
[记录操作菜单]
```

```
1. 录入
```

```
2. 删除
```

```
3. 列表
```

```
4. 检索
```

5. 存盘

6. 加载

7. 返回

请输入菜单命令:6

[记录操作菜单]

1. 录入

2. 删除

3. 列表

4. 检索

5. 存盘

6. 加载

7. 返回

请输入菜单命令:3

刘海发 天津市 300150

马玉林 泰安市 210350

[记录操作菜单]

1. 录入

2. 删除

3. 列表

4. 检索

5. 存盘

6. 加载

7. 返回

请输入菜单命令:1

姓名:安百发

地址:北京市

邮编:100010

[记录操作菜单]

1. 录入

2. 删除

3. 列表

4. 检索

5. 存盘

6. 加载

7. 返回

请输入菜单命令:3

安百发 北京市 100010

刘海发 天津市 300150

马玉林 泰安市 210350

[记录操作菜单]

1. 录入

2. 删除

3. 列表

4. 检索

- 5. 存盘
- 6. 加载
- 7. 返回

请输入菜单命令:4

输入姓名:马玉林

马玉林 泰安市 210350

[记录操作菜单]

- 1. 录入
- 2. 删除
- 3. 列表
- 4. 检索
- 5. 存盘
- 6. 加载
- 7. 返回

请输入菜单命令:2

输入姓名:马玉林

[记录操作菜单]

- 1. 录入
- 2. 删除
- 3. 列表
- 4. 检索
- 5. 存盘
- 6. 加载
- 7. 返回

请输入菜单命令:3

安百发 北京市 100010

刘海发 天津市 300150

[记录操作菜单]

- 1. 录入
- 2. 删除
- 3. 列表
- 4. 检索
- 5. 存盘
- 6. 加载
- 7. 返回

请输入菜单命令:7

C>

## 16.5 二叉树

(1)二叉树及其有关术语 所谓二叉树是指在逻辑上每个数据元素都包含有一个指向左子树指针和右子树指针的一种数据结构,如图 16.10 所示。

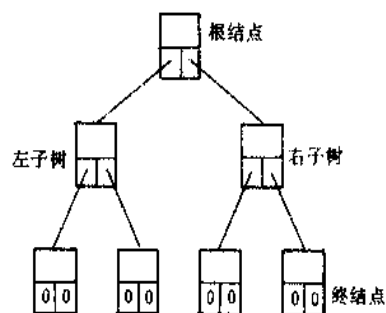


图 16.10 二叉树数据结构

在二叉树中,每个数据元素被看作是一个结点。其中首项元素叫做根结点。每个数据元素都含有两个指针,一个指向左子树,一个指向右子树。即每个结点带两个子树。不带子树的结点叫终点。显然,二叉树的每个结点亦可用 C 语言的结构数据来表示,例如:

```
struct address{
    char name[9];
    char stree[20];
    char zip[7];
    struct address * left;
    struct address * right;
}list_entry;
```

(2)二叉树的访问方法 二叉树的排序与对它的访问方式有关。对树的每个结点的访问称作遍历(tree traversal)。有三种访问方法,即:

- ①中序(inorder)法,其访问次序是:左子树→根→右子树;
- ②前序(preorder)法,其访问次序是:根→左子树→右子树;
- ③后序(postorder)法,其访问次序是:左子树→右子树→根。

譬如,对于如下二叉树,三种访问方法的结果分别为:

中序遍历:dbaefcg

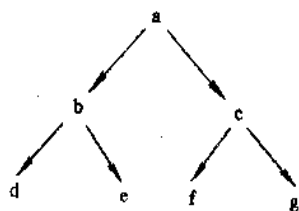
前序遍历:abdecfg

后序遍历:debfgca

(3)应用举例 这里,仍以通讯录为例。通讯录使用二叉树数据结构表示时,其各种操作的相应函数如下。

①录入函数 取名为 enter(),其功能是从键盘上输入一条记录,然后按中序遍历把它存放到二叉树中,即建立二叉树,函数体如下:

```
void enter()
{
    struct address * info, * stree();
    for(;;)
    {
        clrscr();
        info=(struct address *)malloc(sizeof(list_entry));
```



```

if(! info)
{
    printf("\n\n\n 内存不够");
    return;
}
jls=jls+1;
sz[jls]=info;
info->left=NULL;
info->right=NULL;
inputs("名字:",info->name,9);
if(! info->name[0])
{
    --jls;
    break;
}
inputs("地址:",info->street,20);
inputs("邮编:",info->zip,7);
if(root==NULL) root=stree(root,root,info);
else stree(root,root,info);
}
}

```

其中,inputs()和stree()分别是键盘输入记录的函数和按中序遍历建立二叉树的函数。它们的函数体分别如下所示。

#### • inputs()函数

```

inputs(prompt,s,count)
char *prompt,*s;
int count;
{
    char p[50];
    strcpy(s,"");
    do{
        printf(prompt);
        gets(p);
        if(strlen(p)>count) printf("超限\n");
        }while(strlen(p)>count);
    if(strcmp(p,"")!=0) strcpy(s,p);
}

```

#### • stree()函数

```

struct address *stree(top,r,i) /* 按中序遍历排序建立二叉树 */
struct address *i,*r;
struct address *top;
{
    if(! r)
    { if(! top) return(i);

```

```

    if(strcmp(i->name,top->name)<=0) top->left=i;
    else top->right=i;
    return(i);
}
else
{ if(strcmp(i->name,r->name)<=0) stree(r,r->left,i);
  else stree(r,r->right,i);
}
}

```

其中,sz 为类型为 address 的结构指针数组,其定义如下:

```

struct address{
    char name[9];
    char street[20];
    char zip[7];
    struct address * left;
    struct address * right;
}list_entry;
struct address * root=NULL;
struct address * sz[100];

```

②存盘函数 取名为 save(),其功能是把通讯录存入磁盘文件,函数体如下:

```

save ()
{
    int i;
    FILE * fp;
    clrscr();
    if((fp=fopen("read","wb"))==NULL)
    { printf("不能打开文件\n");
      exit(1);
    }
    i=1;
    while(i<=jls)
    {
        if(sz[i]!=NULL)
            fwrite(sz[i],sizeof(struct address),1,fp);
        ++i;
    }
    fclose(fp);
}

```

③加载函数 取名为 load(),其功能是从磁盘文件 read 中读取一条记录,放到结构指针 info 所指定的内存空间里。函数体如下:

```

void load()
{

```

```

sprucp address *info,*spree();
file *fp;
clrscr();
if((fp=fopen("read","rb"))==NULL)
{ printf("不能打开文件\n\n");
  exit(1);
}
while(! feof(fp))
{ info=(struct address *)malloc(sizeof(list_entry));
  if(! info)
  { printf("out of memory\n");
    return;
  }
  if(fread(info,sizeof(struct address),1,fp)!=1)
  {
    free(info);
    break;
  }
  ++jls;
  sz[jls]=info;
  info->left=NULL;
  info->right=NULL;
  if(root==NULL) root=stree(root,root,info);
  else stree(root,root,info);
}
fclose(fp);
}

```

④列表函数 取名为 list(),其功能是打印整个通讯录。函数体如下:

```

list(b)
struct address *b;
{
  if(! b) return;
  list(b->left);
  if(flag)
  {
    flag=0;
    printf("=====\n");
    printf(" | 姓名 | 住址 | 邮编 | \n");
  }
  else
  printf("%14s| %18s| %7s| \n",b->name,b->street,b->zip);
  printf(" |-----|-----|-----| \n");
  list(b->right);
}

```

⑤删除函数 取名 delete(),其功能是删除一条记录。函数体如下:

```

delete()
{ struct address * info, * find();
  int i;
  char s[80];
  clrscr();
  printf("\n\n 名字:");
  gets(s);
  puts(s);
  info=find(s);
  if(info)
  { for(i=1;i<=jls; ++i)
    if(sz[i]==info)
    {
      free(sz[i]);
      sz[i]=NULL;
      save();
      jls=0;
      root=NULL;
      load();
      return;
    }
  }
  else printf("\n Can't delete \n");
}

```

其中,find()为记录查找函数,其功能是查找给定名字字段的记录。函数体如下:

```

struct address * find(name)
char * name;
{ struct address * info;
  info=root;
  while(info)
  { if(! strcmp(name,info->name)) return(info);
    if(strcmp(name,info->name)<=0)info= info->left;
    else info=info->right;
  }
  return(NULL);
}

```

⑥检索函数 取名为 seach(),其功能是根据所给出的名字,查找本条记录,并打印出来。函数体如下:

```

void search()
{
  char name[10];
  struct address * info, * find();
  clrscr();

```



```

printf("查询的姓名:");
gets(name);
puts(name);
info=find(name);
if(! info) printf("未发现\n");
else
{
    printf("===== \n");
    printf(" | 姓名 | 住址 | 邮编 | \n");
    printf("%14s|%18s|%7s\n",info->name,info->street,info->zip);
    printf("===== \n");
}
}

```

⑦菜单函数 取名为 menu\_select(),其功能是在屏幕终端上显示出功能菜单,供用户选择。

函数体如下:

```

menu_select()
{
    char s[80];
    int c;
    clrscr();
    printf("\n\n\n-----主菜单-----\n\n");
    printf(" * * * * * \n");
    printf(" * * * * * \n");
    printf(" *          1. 录入          *          2. 删除          * \n");
    printf(" * * * * * \n");
    printf(" *          3. 显示          *          4. 查询          * \n");
    printf(" * * * * * \n");
    printf(" *          5. 存储          *          6. 加载          * \n");
    printf(" * * * * * \n");
    printf(" *          7. 退出          * * * * * \n");
    printf(" * * * * * \n");
    do
    { printf("\n\n\n 选择操作:");
      gets(s);
      c=atoi(s);
    } while(c<0||c>7);
    puts(s);
    return(c);
}

```

⑧执行情况 用主函数 main()把以上函数连接起来,即调用它们,就可以实现上述各种功能,如下所示:

A>exp16-6

主菜单

```
*****
*                                     *
*      1. 录入      *      2. 删除      *
*                                     *
*      3. 显示      *      4. 查询      *
*                                     *
*      5. 存储      *      6. 加载      *
*                                     *
*      7. 退出      *                                     *
*****
```

选择操作:1

1

名字:刘立

地址:天津大学 2 宿舍

邮编:300153

名字:马宝

地址:红桥区东京路 25 号

邮编:300012

名字:王佩东

地址:河西区马场道 2 号

邮编:300427

名字:王维

地址:河北区曲府道 1 号

邮编:300189

名字:杨科

地址:和平区密云路 14 号

邮编:300178

名字:赵雪梁

地址:南开区三马路 33 号

邮编:300213

名字:赵一铭

地址:东北角和平里 4 号

邮编:300157

-----主菜单-----

```
*****
*                                     *
*          1. 录入          *          2. 删除          *
*                                     *
*          3. 显示          *          4. 查询          *
*                                     *
*          5. 存储          *          6. 加载          *
*                                     *
*          7. 退出          *                                     *
*****
```

选择操作:3

3

姓名,	住址	邮编
刘立	天津大学 2 宿舍	300153
马宝	红桥区东京路 25 号	300012
王佩东	河西区马场道 2 号	300427
王维	河北区曲府道 1 号	300189
杨科	和平区密云路 14 号	300178
赵雪梁	南开区三马路 33 号	300213
赵一铭	东北角和平里 4 号	300157

-----主菜单-----

```
*****
*                                     *
*          1. 录入          *          2. 删除          *
*                                     *
*          3. 显示          *          4. 查询          *
*                                     *
*          5. 存储          *          6. 加载          *
*                                     *
*          7. 退出          *                                     *
*****
```

选择操作:2

2

名字:马宝

马宝

-----主菜单-----

```
*****
*                                     *
*      1. 录入      *      2. 删除      *
*                                     *
*      3. 显示      *      4. 查询      *
*                                     *
*      5. 存储      *      6. 加载      *
*                                     *
*      7. 退出      *                                     *
*****
```

选择操作:3

3

姓名	住址	邮编
刘立	天津大学 2 宿舍	300153
王佩东	河西区马场道 2 号	300427
王维	河北区曲府道 1 号	300189
杨科	和平区密云路 14 号	300178
赵雪梁	南开区三马路 33 号	300213
赵一铭	东北角和平里 4 号	300157

-----主菜单-----

```
*****
*                                     *
*      1. 录入      *      2. 删除      *
*                                     *
*      3. 显示      *      4. 查询      *
*                                     *
*      5. 存储      *      6. 加载      *
*                                     *
*      7. 退出      *                                     *
*****
```

选择操作:4

4

查询的姓名:刘立

刘立

姓名	住址	邮编
刘立	天津大学 2 宿舍	300153

选择操作:7

7

A>

# 第十七章 C 与 FoxBAXE 的接口程序设计

## 17.1 接口的概念

(1)什么是接口 所谓接口,是指设备与设备之间用来传送数据的电路,或是程序与程序之间用来传送数据的软件。

(2)接口的种类 根据如上所述,接口分为硬件接口和软件接口两种。下面分别作一简单介绍。

①硬件接口 它是连接设备的桥梁,一般是指连接 CPU 与外设的硬件电路,如图 17.1 所示。



图 17.1 硬件接口

由于在计算机系统中,各设备是通过总线连接的,故硬件接口与 CPU 和外设的关系,便如图 17.2 所示;

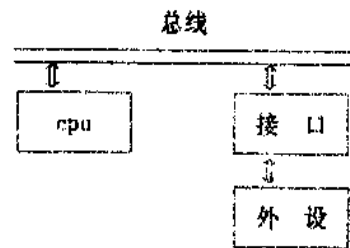


图 17.2 计算机系统中的接口

外设种类繁多,有机械的、电的、光的;信号类型不一;速度相差也甚远,有秒级的、毫秒级的、微秒级的,等。故硬件接口设计是个比较复杂的问题。

②软件接口 它不是硬件电路,而是用来连接两个程序的一个程序。两个程序通过它进行数据传递,如图 17.3 所示。

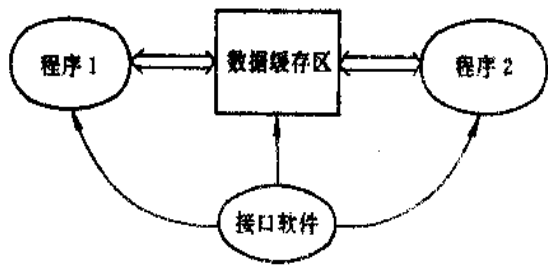


图 17.3 软件接口示意图

(3)软件接口的分类 常用的分法有:

①根据所用的数据缓冲区的不同,分为:

- 使用内存单元作数据缓存区的接口软件。注意,所用的存储单元应为不被程序或数据所覆盖,如选在中断向量表中系统保留部分。

- 使用文本文件作数据缓存区的接口软件。文本文件有传递数据量大、文件格式自由、访问方便、系统依赖性小等特点,故在软件接口中被广为使用。

- 使用系统建立的数据库文件作数据缓存区的接口软件。由于数据库文件建立比较容易,数据量较大,数据规范,所以使用它作数据缓存区,也是一种理想的选择。

②根据被连接的程序,又可分为:

- 应用系统间接口:一般是指用同一种语言所编写的两个程序的接口软件。

- 语言系统间接口:是指用来连接用不同语言所编写的两个程序的接口软件。从本讲的题目来看,这种接口正是我们所要探讨的问题,其中包括如下几个问题:

- 数据缓存区的设置与结构;

- 语言 A(程序 1)读写数据缓存区的方法;

- 语言 B(程序 2)读写数据缓存区的方法;

- 语言 A(程序 1)调用语言 B(程序 2)的方法;

- 语言 B(程序 2)调用语言 A(程序 1)的方法;

等。

(3)本章的目的 C 语言有强有力的计算、绘图功能,而 FoxBASE 有很强的数据处理能力,如能实现 C 与 FoxBASE 的连接,这无疑能生产出更加完美的计算机信息系统。本讲将探讨这一问题的解决方法及其实现。

由于 C 与 FoxBASE 之间数据传送量较大,C 与 FoxBASE 的接口要在文件一级上实现。这就需要了解 C 文件和 FoxBASE 文件。关于 C 文件已在第十章介绍过,这里从 FoxBASE 文件谈起。

## 17.2 FoxBASE 的文件类型

FoxBASE 有 10 种不同格式的磁盘文件,即:

(1)数据库文件 其扩展名为 .DBF,其内容为数据库的结构说明信息和数据。数据是以记录形式存放的,每个记录由若干个字段组成。FoxBASE 的数据库文件最多能存储 10 亿个记录,每个记录最多允许有 128 个字段。记录的字节数可达 4000 个。

进入 FoxBASE 系统后,该文件由 CREATE 命令建立,由 USE 命令调用。

(2)备注文件 其扩展名为 .DBT。它是数据库文件的辅助文件,用来存储备注(MEMO)字段的内容。一个数据库文件中的所有备注字段都被存到同一个 .DBT 文件中。每个数据库记录最多可包含 128 个备注字段。备注字段的数据类型与字符型字段相同,其大小可达 4096 个字节。每个备注字段在数据库文件中占 10 个字节,作为指针,用来指向备注文件中相应信息的位置。

(3)索引文件 其扩展名为 .IDX。它是由关键字和相应的数据库记录号所组成的文件。当数据库文件和索引文件一起使用时,数据库将按其关键字顺序而显示。

索引文件由 INDEX 命令建立。

(4)命令文件 其扩展名为 .PRG。它是由 FoxBASE 命令语句构成的程序,是 ASCII 文件,可由 MODIFY COMMAND 命令建立,也可由其它编辑软件编辑。

(5)格式文件 其扩展名为 .FMT。它是确定屏幕输出格式的文件,用于数据输入和打印输出。

该文件由 MODIFY COMAND 命令或其它编辑软件编辑,使用 SET FORMAT TO 命令来定义。

(6)标签文件 其扩展名为 .LBL。它包含打印标签时所需要的信息。

该文件由 CREATE LABEL 命令建立,由 LABEL 命令输出。

(7)变量文件 其扩展名为 .MEM。它是存放内存变量的文件。该文件最多可存 256 个内存变量,且总字节数限制在 6000 个。

该文件可由 SAVE 命令建立,并由 RESTORE 命令读入内存。

(8)报表文件 其扩展名为 .FRM,是用 REPORT 命令由当前数据库文件产生的文件,其内容包括制订报表时所需要的全部信息。

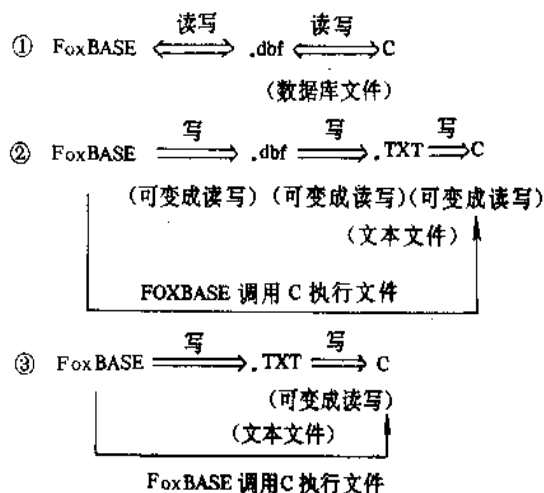
报表格式可由 CREATE REPORT 或 MODIFY REPORT 建立。

(9)文本文件 其扩展名为 .TXT。它是 ASCII 文件,不仅包含可打印的 ASCII 码字符,也可扩充到汉字机内码形式。使用该文件可与其他语言程序进行数据传送。

该文件可用 COPY 命令建立,可用 APPEND FROM 命令写回到数据库文件中。

(10)目标文件 其扩展名为 .FOX 或 .FMX。它是由命令文件或格式文件经编译后所形成的文件。它既可在 FoxBASE 下运行,又可在 FoxBASE 的运行环境下执行。

利用 FoxBASE 数据库文件和文本文件,本章将介绍 C 与 FoxBASE 接口的三种设计方案,如下所示:



### 17.3 FoxBASE 数据库文件结构

为了说明方便,我们建立一个很简单的数据库文件 PP.DBF,其内容如下:

Record #	A1	A2
1	4444	444444
2	6666	666666
3	8888	888888

该文件有三条记录,每条记录都只有 A1 和 A2 这么两种类型的字段。A1 和 A2 的类型和大小如下所示:

Structure for database : C:\FOX\PP.DBF

Number of data records: 3  
 Date of last update : 06/15/92  
 Field Field Name Type Width Dec  
 1 A1 Character 4  
 2 A2 Character 6  
 \* \* Total \* \* 11

该数据库文件整个结构如下所示:

Displacement	Hex codes	ASCII Value
0000(0000)	03 5C 06 0F 03 00 00 00 61 00 0B 00 00 00 00 00	\ a
0016(0010)	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
0032(0020)	41 31 00 00 00 00 00 00 00 00 00 43 01 00 00 00	A1 C
0048(0030)	04 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
0064(0040)	41 32 00 00 00 00 00 00 00 00 00 43 05 00 00 00	A2 C
0080(0050)	06 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	
0096(0060)	0D 20 34 34 34 34 34 34 34 34 34 20 36 36 36 36	4444444444 666
0112(0070)	36 36 36 36 36 36 36 36 20 38 38 38 38 38 38 38	66666666 88888888
0128(0080)	38 28 1A 20 20 20 20 20 50 52 47 20 00 00 00 00	88 PRG
0144(0090)	00 00 00 00 00 00 00 F4 93 64 19 A3 1A 33 05 00 00	dv 3
0160(00A0)	50 50 20 20 20 20 20 20 42 41 4B 20 00 00 00 00	PP BAK
0176(00B0)	00 00 00 00 00 00 84 73 CA 18 E9 10 47 00 00 00 00	s A > G
0192(00C0)	4B 4B 4B 20 20 20 20 20 42 41 4B 20 00 00 00 00	KKK BAK
0208(00D0)	00 00 00 00 00 00 13 81 CA 18 09 11 E6 00 00 00 00	A <
0224(00E0)	4B 4B 4B 20 20 20 20 20 54 58 54 2 00 00 00 00	KKK TXT
0240(00F0)	00 00 00 00 00 00 45 81 CA 18 00 00 00 00 00 00 00	E

经过对该文件结构的分析,我们可以看到数据库文件由如下三部分组成:

(1)数据库文件结构说明占 32 个字节,分配如表 17.1 所示。

表 17.1 数据库文件结构说明

位置	长度	内容
0	1 字节	最低两位为版本号
1-3	3 字节	建库年月日
4-7	4 字节	记录个数
8-9	2 字节	结构说明和字段说明共占字节数+1
10-11	2 字节	记录字节个数
12-31	20 字节	保留

(2)字段说明 每个字段的说明都占 32 个字节,其分配如表 17.2 所示。

表 17.2 字段的说明

位置	长度	内容
0-10	11 字节	字段名 ASCII 码
11	1 字节	字段类型
12-15	4 字节	字段在记录中的位置
16	1 字节	字段长度的 2 进制数
17	1 字节	小数点后位数的 2 进制
18-19	2 字节	保留
20	1 字节	工作区标志 ID
21-31	11 字节	保留



(3)记录内容 该部分的字节数由记录个数和每条记录的长度决定。注意:

①每个记录开始的第一个字节是删除标志。为 20H,表示该记录未删除;为 2AH,表示该记录已删除。

②最后一条记录后有文件结尾标志:1AH。

③第 1 条记录前有说明部分的结尾标志:0DH。

#### 17.4 C 如何读取数据库文件的说明信息

数据库文件的说明信息,包括其结构说明和字段说明,对于实现 C 直接操作 FoxBASE 数据库文件相当重要。因此,要把这些信息读到 C 文件,以备使用。那么,如何读取呢?无论从读这些信息来看,还是从使用这些信息来看,最方便的方法是建立与它们对应的两个结构数据,如下所示:

```
typedef struct {
    char ver;
    char date[3];
    unsigned long record_num;
    unsigned int stru_byte_num;
    unsigned int record_byte_num;
    char unse[20]
}DBF1;

typedef struct {
    unsigned char field_name[11]
    char field_type;
    char unse1[4];
    char field_len;
    char decimal;
    char unse2[2];
    char work_area;
    char unse3[11];
}DBF2;
```

这里是用类型定义关键字 typedef 定义的。其中,DBF1 是含有 6 个成员的结构数据类型,其 6 个成员分别对应着结构说明的 6 个数据。DBF2 是含有 8 个成员的结构数据类型,其 8 个成员分别与字段说明的 8 个数据相对应。这样,我们可以定义这样的两个结构数据变量,再用 fread()函数从数据库文件中把结构说明信息和字段说明信息分别读取到这两个变量中,如下所示:

```
DBF1 dbs;
DBF2 dbrec [FIELD_NUM_MAX];
fp=fopen(fname,"r+b");
fread(&dbs,32,1,fp);
fread(dbrec,32,(dbs.stru_byte_num-33)/32,fp);
```

其中,iname 为指向具体数据库文件的文件指针;(stru\_byte\_num-33)/32 为记录的字段数。

结构说明信息和字段说明信息分别读取到对应的结构变量后,再用到这些信息时就可以直接使用相应结构变量中的有关成员。

#### 17.5 记录的定位、录入和修改

(1)记录的定位 要存取记录内容,可用 fseek()函数定位,确定好文件指示器的位置。例如,要

把文件位置指示器定位在 num 条记录前、后,可分别使用如下 fseek() 函数调用。

```
fseek(fp,dbs.stru_byte_num+(num-1)*dbs.record_byte_num,0);
```

```
fseek(fp,num*dbs.record_byte_num+dbs.stru_byte_num,0);
```

其中,fp 是文件指针,用来指定具体的数据库文件。

(2)记录的录入 在 FoxBASE 数据库文件中,每条记录都是由若干个字段所组成。因此,记录的录入可按如下思路进行:

①设置一个记录指针,这里为 cc;

②再设置一个存放各字段的数组,这里为 ccc;

③然后使用 gets() 函数逐次把各字段内容输入到数组 ccc。注意,若输入的内容长度小于字段长度,则用空格填满。

④使用 strcat() 函数,依次把各字段追加到 cc 所指定的空间。这样,cc 所指定的内容便是我们想录入的一条记录。

该功能函数为 key\_read(), 其函数体如下所示:

```
void key_read()
{
    int i,j;
    char ccc[255];
    cc[0]='\0';
    printf("\n");
    gets(ccc);
    for(i=0;i<=(dbs.stru_byte_num-33)/32;i++)
    {
        printf("字段名:%s, 类型:%c, 长度:%d, 小数点:%d\n",dbrec[i].field_name,
            dbrec[i].field_type,dbrec[i].field_len,dbrec[i].decimal);
        if(gets(ccc)==NULL)
        {
            printf("输入内容错误!!");
            exit(0);
        }
        if(strlen(ccc)>dbrec[i].field_len)
        {
            printf("输入内容太长,请再输一遍!! \n");
            --i;
        }
        else if(strlen(ccc)<dbrec[i].field_len)
        {
            for(j=strlen(ccc);j<dbrec[i].field_len;j++)
                ccc[j]=' ';
            ccc[j]='\0';
            strcat(cc,ccc);
        }
        else

```

```

        strcat(cc,ccc);
    }
}

```

(3)记录的修改 记录的修改包括记录的删除、插入、置换等功能。这里,以这三种功能为例,介绍它们的实现方法,抛砖引玉,以期举一反三。

①记录的置换方法 置换某条记录,可按如下步骤进行:

- 用函数 malloc()在堆自由区域分配一块大小为记录字节数+1的空间,用来存放新记录。多余的一字节,用来存放字符串的结尾符。

- 用函数 strcpy()把新记录存入上述开辟的自由空间。
- 用函数 fseek()把文件位置指示器定位在要置换的记录的开始位置。
- 用函数 fwrite()把新记录写到原记录的位置。

实现该置换功能的函数取名为 f\_replace(),其函数体如下所示:

```

void f_replace()
{
    int i;
    char aa=' ',*t;
    if(num>dbb.record_num)
    {
        printf("记录号太大! \n");
        exit(0);
    }
    t=malloc(dbb.record_byte_num-1+1);
    strcpy(t,cc);
    fseek(fp,dbb.stru_byte_num+(num-1)*dbb.record_byte_num+1,0);
    fwrite(t,strlen(t),1,fp);
    for(i=strlen(cc);i<dbb.record_byte_num-1;i++)
        fwrite(&aa,1,1,fp);
    free(t);
}

```

②删除记录的方法 删除记录可使用覆盖技术,基本思路是:

- 把要删除的记录后面的全部记录读取到事先设置好的自由空间里去;
- 然后把文件位置指示器调整到要删除的记录的开始位置;
- 之后把存放在自由空间的要删除的记录后面的全部记录从要删除的记录的开始位置写入原数据库文件。这样,即删除要删的记录。

- 最后修改结构说明中的记录个数,即完成记录删除任务。

实现该功能的函数取名为 f\_delete(),其函数体如下所示:

```

void f_delete()
{
    char *t;
    if(num>dbb.record_num)
    { printf("无此记录! \n");

```

```

    exit(0);
}

t = malloc((dbs.record_num--num) * dbs.record_byte_num + 1);
fseek(fp, num * dbs.record_byte_num + dbs.stru_byte_num, 0);
fread(t, dbs.record_byte_num * (dbs.record_num--num) + 1, 1, fp);
fseek(fp, (num-1) * dbs.record_byte_num + dbs.stru_byte_num, 0);
fwrite(t, dbs.record_byte_num * (dbs.record_num--num) + 1, 1, fp);
free(t);
fseek(fp, 4, 0);
putw(--dbs.recorb_num, fp);
}

```

③插入记录的方法 在 FoxBASE 数据库文件中,插入一条记录的步骤可按如下思路进行:

- 用函数 malloc()在堆自由区域分配一块大小为插入位置后面的全部记录所占字节数+1 的自由空间;
- 用函数 fread()将插入位置后面的全部记录读到上述自由空间;
- 用函数 fwrite()在插入位置写进要插入的记录
- 用函数 fseek()把文件位置指示器下调到下一条记录的开始位置;
- 用函数 fwrite()把读到上述自由空间的内容写到新加入的记录后。这样,即在指定记录后加入一条新记录;
- 修改结构说明中的记录个数,即+1。

实现该功能的函数取名为 f\_insert(),其函数体如下所示:

```

void f_insert()
{
    char *t, tt = ' ';
    if(num >= dbs.record_num)
    {
        printf("无此记录! \n");
        exit(0);
    }
    t = malloc((dbs.record_num--num) * dbs.record_byte_num + 1);
    if(num != dbs.record_num)
    {
        fseek(fp, num * dbs.record_byte_num + dbs.stru_byte_num, 0);
        fread(t, dbs.record_byte_num * (dbs.record_num--num) + 1, 1, fp);
        fseek(fp, num * dbs.record_byte_num + dbs.stru_byte_num + 1, 0);
        fwrite(cc, dbs.record_byte_num, 1, fp);
        fseek(fp, dbs.record_byte_num * (num+1) + dbs.stru_byte_num, 0);
        fwrite(t, dbs.record_byte_num * (dbs.record_num--num) + 1, 1, fp);
    }
    else
    {
        fseek(fp, num * dbs.record_byte_num + dbs.stru_byte_num + 1 - 1, 0);
        fwrite(&tt, 1, 1, fp);
        fwrite(cc, dbs.record_byte_num, 1, fp);
    }
}

```

```

    fseek(fp,dbs.record_byte_num * (num+1)+dbs.stru_byte_num,0);
    putw(0x1a,fp);
}
free(t);
fseek(fp,4,0);
putw(++dbs.record_num,fp);
}

```

〔练习 17.1〕 该练习调用以上所介绍的函数,实现了 C 与 FoxBASE 数据库文件的通用接口,具有显示、置换、插入和删除记录等功能。为节省篇幅,这里只给出数据库结构及内容、主函数 main()、菜单函数 menu()和执行情况,如下所示:

(1)原 FoxBASE 数据库文件 MY2.DBF 的结构与内容:

.disp stru

数据库结构: C:\FOX\MY2.DBF

数据记录数: 5

最新更改日期: 04/14/93

字段	字段名	类型	宽度	小数
1	NAME1	字符	8	
2	NAME2	数值	5	2
3	NAME3	日期	8	
4	NAME4	逻辑	1	
5	NAME5	备注	10	
* * 总和 * *			33	

.disp all

记录号 #	NAME1	NAME2	NAME3	NAME4	NAME5
1	安来福	82.11	08/17/92	.F.	备注
2	兰淑萍	94.11	07/11/92	.T.	备注
3	司徒樱子	87.12	06/12/92	.F.	备注
4	吴玉方	78.33	08/01/91	.T.	备注
5	贾书萍	45.44	05/21/90	.T.	备注

(2)外部变量和主函数

C>type a,expl7-1.c

```
#include "graphics.h"
```

```
#include "my.h"
```

```
FILE *fp;
```

```
int num,k;
```

```
char *cc;
```

```
DBF1 dbs;
```

```
DBF2 dbrec[FIELD_NUM_MAX];
```

```
main(argc,argv)
```

```
int argc;
```

```
char *argv[];
```

```

int graphdriver=DETECT,graphmode;
void f_open();
void f_replace();
void f_delete();
void f_insert();
void f_list();
void menu();
void key_read();
int i,j;
char *ccc;
char dbf_name[12];
initgraph(&graphdriver,&graphmode,"");
if(argc ==2)
{ printf("用法:exp17-1 数据库文件名\n");
  exit(0);
}
f_open(argv[1]);
cc=malloc(dbs.record_byte_num+1);
for(;;)
{ menu();
  switch(k){
    case 1:
      printf("请输入记录号:");
      scanf("%d",&num);
      printf("输入记录内容:");
      key_read();
      f_replace();
      break;
    case 2:
      printf("请输入记录号:");
      scanf("%d",&num);
      f_delete();
      break;
    case 3:
      printf("将在几号记录后添加:");
      scanf("%d",&num);
      printf("请输入记录内容:");
      key_read();
      f_insert();
      break;
    case 4:
      break;
    case 5:

```

```

        closegraph();
        printf("\n\n 请输入 DOS 命令:");
        scanf("%s",dbf_name);
        system(dbf_name);
        initgraph(&graphdriver,&graphmode,"");
        break;
    case 6:
        f_list();
        break;
    case 0:
        fclose(fp);
        closegraph();
        exit(0);
        break;
    }
}
}

```

### (3) 菜单函数 menu()

```

void menu()
{
    char s[80];
    printf("1:----->置换一条记录\n");
    printf("2:----->删除一条记录\n");
    printf("3:----->插入一条记录\n");
    printf("4:----->扩展目录\n");
    printf("5:----->执行 DOS 命令\n");
    printf("6:----->列表\n");
    printf("0:----->返回\n");
    do{
        printf("请选择:");
        scanf("%s",s);
        k=atoi(s);
    }
    while(k<0||k>6);
}

```

### (4) 文件 my.h 的内容:

```

C>type my.h
#include <stdlib.h>
#include <stdio.h>
#include <alloc.h>
#include <string.h>
typedef struct{

```

```

char ver;
char date[3];
unsigned long record_num;
unsigned int stru_byte_num;
unsigned int record_byte_num;
char unse[20];
}DBF1;
typedef struct {
    unsigned char field_name[11];
    char field_type;
    char unse1[4];
    char field_len;
    char decimal;
    char unse2[2];
    char work_area;
    char unse3[11];
}DBF2;
#define FIELD_NUM_MAX 10

```

#### (5)执行情况:

A>exp17\_1 my2.dbf

当前记录个数:5

记录长度:33

- 1:----->置换一条记录
- 2:----->删除一条记录
- 3:----->插入一条记录
- 4:----->扩展目录
- 5:----->执行 DOS 命令
- 6:----->列表
- 0:----->返回

请选择:6

安来福	82.11	19920817	F	1
兰淑萍	94.11	19920711	T	2
司徒樱子	87.12	19920612	F	3
吴玉方	78.33	19910801	T	0
贾书萍	45.44	19900521	T	0

- 1:----->置换一条记录
- 2:----->删除一条记录
- 3:----->插入一条记录
- 4:----->扩展目录
- 5:----->执行 DOS 命令
- 6:----->列表
- 0:----->返回

请选择:1



请输入记录号:2

输入记录内容:

字段名:NAME1,类型:C,长度:8,小数点:0

郝考考

字段名:NAME2,类型:N,长度:5,小数点:2

ALDK

字段名:NAME3,类型:D,长度:8,小数点:0

19920809

字段名:NAME4,类型:L,长度:1,小数点:0

T

字段名:NAME5,类型:M,长度:10,小数点:0

1:----->置换一条记录

2:----->删除一条记录

3:----->插入一条记录

4:----->扩展目录

5:----->执行 DOS 命令

6:----->列表

0:----->返回

请选择:6

安来福 82.11 19920817 F 1

郝考考 ALDK 19920809 T

司徒樱子 87.12 19920612 F 3

吴玉方 78.33 19910801 T 0

贾书萍 45.44 19900521 T 0

1:----->置换一条记录

2:----->删除一条记录

3:----->插入一条记录

4:----->扩展目录

5:----->执行 DOS 命令

6:----->列表

0:----->返回

请选择:2

请输入记录号:4

1:----->置换一条记录

2:----->删除一条记录

3:----->插入一条记录

4:----->扩展目录

5:----->执行 DOS 命令

6:----->列表

0:----->返回

请选择:3

将在几号记录后添加:3

请输入记录内容:

字段名:NAME1,类型:C,长度:8,小数点:0

赠站站

字段名:NAME2,类型:N,长度:5,小数点:2

112.3

字段名:NAME3,类型:D,长度:8,小数点:0

19901209

字段名:NAME4,类型:L,长度:1,小数点:0

F

字段名:NAME5,类型:M,长度:10,小数点:0

3

1:----->置换一条记录

2:----->删除一条记录

3:----->插入一条记录

4:----->扩展目录

5:----->执行 DOS 命令

6:----->列表

0:----->返回

请选择:6

安来福 82.11 19920817 F 1

郝考考 ALDK 19920809 T

司徒樱子 87.12 19920612 F 3

熊晓站 112.3 19901209 F 3

陈华军 45.44 19905521 T 0

1:----->置换一条记录

2:----->删除一条记录

3:----->插入一条记录

4:----->扩展目录

5:----->执行 DOS 命令

6:----->列表

0:----->返回

请选择:5

(略)

## 17.6 利用由数据库文件建立的文本文件作为数据缓存区的 FoxBASE 与 C 的接口程序设计

(1)研究的目标 从本章的 3 到 5,介绍的是 C 与 FoxBASE 接口程序的第一设计方案。该接口程序实际上是 C 与 FoxBASE 的一个通用接口。该接口程序的特点是:

- 用 C 编写的;
- FoxBASE 的数据库文件直接作为数据缓存区;
- 在 C 系统下操作,可访问任何 FoxBASE 数据库文件。

在这一节里,我们研究 FoxBASE 与 C 接口程序设计的第二个方案。研究的目标是:接口程序用 FoxBASE 命令编写;在 FoxBASE 环境下操作,能调用使用 FoxBASE 数据库文件中数据的 C 程序,以此达到 FoxBASE 与 C 的连接。

(2)接口程序设计步骤 该方案设计步骤如下:

①建立数据库文件,为C程序提供数据。可使用如下命令建立。

```
create 数据库文件名
use 数据库文件名
append blank
```

这样,就建立了一个数据库文件,且在文件里放进了一条空记录。

②用 replace 命令给上述数据库文件某些字段赋值,供使用。

③用 copy 命令把上述数据库文件转换为数据项间为空格的文本文件。这种文件相当于C的流式文件,可直接为C使用。

④编辑使用上述.TXT文件的C程序(一个文件),并把它编译、链接成可执行文件。

⑤在 FoxBASE 下,直接使用 RUN(!)命令运行上述可执行文件。

〔练习 17.2〕 这是采用第二方案所设计的 FoxBASE 与 C 的接口程序。它可在 FoxBASE 下调用使用 FoxBASE 数据库文件中数据的 C 执行文件。有关文件如下:

(1)FoxBASE 命令文件

```
A>type fox. prg
```

```
clear
```

```
set talk off
```

```
set color off
```

```
set bell off
```

```
clear
```

```
k=" "
```

```
i=1
```

```
do while i=1
```

```
set device to screen
```

```
@1,25 say " " "
```

```
@2,25 say " 0-----返回系统 " "
```

```
@3,25 say " 1-----修改文件 " "
```

```
@4,25 say " 2-----查询内容 " "
```

```
@5,25 say " 3-----打印报表 " "
```

```
@6,25 say " 4-----绘制图表 " "
```

```
@7,25 say " 5-----数据统计 " "
```

```
@8,25 say " " "
```

```
@9,25 say " " "
```

```
wait " 输入命令序号(0--5)" to k
```

```
if k="5"
```

```
use gz
```

```
count for jbgz<90 to a1
```

```
count for jbgz>=90 .and. jbgz<100 to b1
```

```
count for jbgz>=100 .and. jbgz<150 to c1
```

```
count for jbgz>=150 to d1
```

```
use cad
```

```

append blank
replace a with a1,b with b1,c with c1,d with d1
copy to c:\tc\tyl fields a,b,c,d type delimited with blank
use
wait
run c:\tc\cf.exe
endif
if k="0"
use
return
endif
clear
enddo

```

## (2) FoxBASE 数据库文件

```

use a:gz.dbf
disp all

```

记录号 #	姓名	JBGZ
1	a	90.00
2	b	100.00
3	c	80.00
4	d	56.00
5	e	70.00
6	e	150.00
7	f	200.00
8	f	90.00
9	g	77.00
10	h	80.00
11	i	130.00
12	j	110.00
13	k	190.00

## (3) 作为数据缓存区的 FoxBASE 数据库文件

```

use a:cad.dbf
disp all

```

记录号 #	A	B	C	D
1	5.00	2.00	3.00	3.00
2	5.00	2.00	3.00	3.00

## (4) 由 cad.dbf 转换成的文本文件

```

A>type tyl.txt
5.00 2.00 3.00 3.00
5.00 2.00 3.00 3.00

```

## (5) 由 FoxBASE 命令文件调用的 C 源文件

```

C>type a:cf.c
#include "graphics.h"

```

```

#include "stdio. h"
#define YZ 50
main()
{
    FILE *fp;
    char bal[40], jbal[4][10];
    int i,j,k,a,b,c,d;
    if((fp=fopen("ty1. txt", "r"))==NULL)
    { printf("Cannot open file\n");
      exit(1);
    }
    fgets(bal, 40, fp);
    fclose(fp);
    for(i=0,j=0,k=0; bal[i]!='\n' || bal[i]==EOF || i<=40; ++i)
    {
        if(bal[i]==' ')
        {
            j++;
            k=0;
        }
        jbal[j][k++]=bal[i];
    }
    a=atoi(jbal[0]);
    b=atoi(jbal[1]);
    c=atoi(jbal[2]);
    d=atoi(jbal[3]);
    printf("\na=%d,b=%d,c=%d,d=%d", a,b,c,d);
    getch();
    sub1(a * YZ, b * YZ, c * YZ, d * YZ);
}

sub1(a,b,c,d)
int a,b,c,d;
{
    int driver, mode;
    char *a1="A", *b1="B", *c1="C", *d1="D";
    driver=HERCMONO;
    mode=HERCMONOH;
    initgraph(&driver, &mode, "");
    setbkcolor(LIGHTGRAY);
    settextjustify(CENTER_TEXT, CENTER_TEXT);
    settextstyle(TRIPLEX_FONT, HORIZ_DIR, USER_CHAR_SIZE);
    a=300-a;
    setfillstyle(SOLID_FILL, GREEN);
    bar(150, a, 170, 300);
}

```

```

outtextxy(160,a-20,a1);
b=300-b;
setfillstyle(SOLID_FILL,RED);
bar(250,b,270,300);
outtextxy(260,b-20,b1);
line(100,20,100,300);
line(100,300,550,300);
c=300-c;
setfillstyle(SOLID_FILL,BLUE);
bar(350,c,370,300);
outtextxy(360,c-20,c1);
d=300-d;
setfillstyle(SOLID_FILL,YELLOW);
bar(450,d,470,300);
outtextxy(460,d-20,d1);
getch();
restorecrtmode();
closegraph();
}

```

## 17.7 利用由 FoxBASE 直接建立的文本文件作为数据缓存区的 FoxBASE 与 C 的接口程序设计

17.6 节所介绍的接口程序有如下一些特点:

- 用 FoxBASE 命令编写的;
- 在 FoxBASE 下执行;
- 使用由 FoxBASE 数据库文件转换成的文本文件作为数据缓存区;
- FoxBASE 调用 C 执行文件。

可见,与按方案 1 所设计的接口程序完全不同。它不是通用接口程序,而是专用的。

本节介绍接口程序的第三种设计方案,其具体设计步骤如下:

①用如下命令建立磁盘文本文件:

set alternate to 文件名

②用屏幕输出命令(?),把 C 程序所需数据存放到上述文件;

③编辑、编译、链接 C 程序,使其最终成为可执行文件;

④用 Run(!)命令调用使用上述文本文件的 C 执行文件,完成 FoxBASE 与 C 的连接。

〔练习 17.3〕 采用第三种方案所设计的 FoxBASE 与 C 的接口程序。该程序在 FoxBASE 下可调用使用 FoxBASE 建立的文本文件中数据的 C 执行文件。其功能可按给定的数据绘制直方图、扇形图和折线图。程序如下:

①FoxBASE 主程序 该程序用 FoxBASE 编写,有二个功能:

• 对已有数据进行分析 事先建好一个名为 cbtjr 的数据库文件。它包括 XH 和 DATA 两个字段。DATA 字段中放置了一些绘图使用的统计结果数据。这些数据的标志是 XH 字段中左 4 位为“JSJG”(这里只提供一个调用模式,实际应用时,可根据实际情况,加以修改)。该程序可以直接从数据库中将这统计结果调出来,再调用使用这些数据的 C 程序,以绘制相应的图形。

• 对临时从键盘输入的数据进行分析 该程序也能对用户从键盘上临时输入的数据进行分

析,从而画出相应的图形。

```
C>type a:expl7-3.prg
set talk off
set color to w+/b+
clear
WAIT '对已有数据进行分析吗? [Y/N]:' TO EX4
DIME EX2(100)
IF EX4='Yy'
use cbtjk
LOCA ALL FOR LEFT(XH,4)='JSJG'
SKIP 2
EX1=1
DO WHILE LEFT(XH,4)='JSJG'
EX2(EX1)=DATA
EX1=EX1+1
SKIP 2
ENDDO
EX1=EX1-1
else
?
? '请输入任意个将要图形表示的数据:(若想结束,请直接回车):'
EX1=1
DO WHILE .T.
ACCEPT '请输入第'+LTRIM(RTRIM(STR(EX1)))+ '个数据:' TO EX2(EX1)
EX2(EX1)=' '+EX2(EX1)
IF RIGHT(EX2(EX1),1)=' '
EXIT
ENDIF
ENDDO
EX1=EX1-1
ENDIF
SET CONS OFF
SET ALTERNATE TO HJFCAN.TXT
SET ALTERNATE ON
? EX1
EX3=1
DO WHILE EX3<=EX1
? VAL(EX2(EX3))
EX3=EX3+1
ENDDO
SET ALTERNATE OFF
SET ALTERNATE TO
SET CONS ON
CLEAR
```

```

DO WHILE .T.
set color to /n+
@10,22 clear to 15,39
SET COLOR TO W+/G+,W+/R+
@9,20 say '
@10,20 PROM '直方图表示'
@11,20 PROM '扇形图表示'
@12,20 PROM '折线图表示'
@13,20 PROM '退出'
@14,20 say '
MENU TO EXXX
DO CASE
CASE EXXX=1
    1 HJFDRAW
CASE EXXX=2
    1 HJFY
CASE EXXX=3
    1 HJFX
CASE EXXX=4
    close all
    set color to w+/b+
    clear
    EXIT
ENDCASE
set color to w+/b+
clear
ENDDO

```

## ②画直方图的 C 程序

A>TYPE HJFDRAW.C

```

/* THIS IS A DRAW BAR3D PROGRAM,MADE OF HJF */
#include <graphics.h>
#include <stdio.h>
#define NUM 20
#define row(x) (x+100)
#define col(x) (320-x)
main()
{
    double atof();
    FILE *fp;
    int iii,w;
    int x,j,k1,k2;
    char *xx=">","*yy="^","*zz="0";
    char canc[15],k,jj[2];
    char xb[20][3]={"1","2","3","4","5","6","7","8","9","10","11","12","13","14","15","16",

```



```

        "17", "18", "19", "20");
double ii, can[NUM], max;
int num, i;
int graphdriver=HERCMONO, graphmode=HERCMONOH1;
fp=fopen("hjfcan.txt", "r");
k=fgetc(fp);
i=0;
while(fgets(canc, 15, fp) != NULL)
    can[i++] = atof(canc);
max=can[1];
for(i=1, i<can[0]; i++)
    if(max<can[i])
        max=can[i];
initgraph(&graphdriver, &graphmode, "");
setcolor(3);
line(row(0), col(0), row(0), col(330));
line(row(0), col(0), row(450), col(0));
settextjustify(CENTER_TEXT, CENTER_TEXT);
settextstyle(TRIPLEX_FONT, HORIZ_DIR, USER_CHAR_SIZE);
outtextxy(row(0), col(320), yy);
outtextxy(row(445), col(5), xx);
outtextxy(75, 310, zz);
j=360/can[0];
x=0;
for(k1=1, k1<=can[0]; k1++)
{
    k2=k1%11;
    if(k2==0)
        k2=11;
    setfillstyle(k2, k2);
    ii=can[k1]/max;
    iii=ii*280;
    bar3d(row(x), col(0), row(x+j), col(iii), 20, 1);
    outtextxy(row(x+j/2), 330, xb[k1-1]);
    outtextxy(80, col(280/can[0]*k1), xb[k1-1]);
    x+=j;
}
scanf("%c", &k);
closegraph();
}

```

### ③画扇形图的 C 程序

A>TYPE HJFY.C

```

/* THIS IS A DRAW PIESLICE PROGRAM, MADE OF HJF */
#include <graphics.h>

```

```

#include <stdio.h>
#include <math.h>
#define NUM 30
#define row(x) (x+100)
#define col(x) (310-x)
#define yxx 280
#define yxy 180
main()
{
    double atof();
    double hd=360/6.282,kx,ky;
    FILE *fp;
    char canc[15];
    char xb[30][3]={"1","2","3","4","5","6","7","8","9","10","11","12","13","14","15","16",
        "17","18","19","20","21","22","23","24","25","26","27","28","29","30"};
    double can[NUM],max,sum;
    int k2;
    int k1;
    char k;
    int num,i;
    int begin,begin1,end;
    int graphdriver=HERCMONO,graphmode=HERCMONOH1;
    fp=fopen("hjfcanc.txt","r"); /* in value */
    k=fgetc(fp);
    i=0;
    while(fgets(canc,15,fp)!=NULL)
        can[i++] = atof(canc);
    max=can[1]; /* find maximum values */
    for(i=1;i<=can[0];i++)
        if(max<can[i])
            max=can[i];
    sum=0.0; /* find sum about values */
    for(i=1;i<=can[0];i++)
        sum+=can[i];
    initgraph(&graphdriver,&graphmode,"");
    begin=0;end=can[1]/sum*360;
    settextjustify(CENTER_TEXT,CENTER_TEXT);
    settextstyle(TRIPLEX_FONT,HORIZ_DIR,USER_CHAR_SIZE);
    for(k1=1;k1<=can[0];k1++)
    {
        k2=k1%11;
        if(k2==0)
            k2=11;
        setfillstyle(k2,k2);
    }
}

```

```

begin1=(begin+end)/2;
pieslice(yxx,xyy,begin,end,150);
begin=end;
if(k1+1==can[0])
    end=360;
else
    end=end+can[k1+1]/sum*360;
ky=xyy-sin(begin1/hd)*150.0;
kx=yxx+150.0*cos(begin1/hd);
if(begin1<45||begin1>315)
    kx=kx+20;
else
    if(begin1>45&&begin1<135)
        ky=ky-5;
    else
        if(begin1>135&&begin1<225)
            kx=kx-15;
        else
            ky=ky+10;
    outtextxy(kx,ky,xb[k1-1]);
}
scanf("%c",&k);
closegraph();
}

```

#### ④画折线图的 C 程序

```

A>type hjfx.c /* THIS IS A DRAW MANY LINE PROGAM,MADE OF HJF */
#include <graphics.h>
#include <stdio.h>
#define NUM 200
#define row(x) (x+100)
#define col(x) (320-x)
main()
{
    double atof();
    FILE *fp;
    char *xx=">",*yy="^",*zz="0";
    char can[15],k,jj[2];
    char xb[20][3]={"1","2","3","4","5","6","7","8","9","10","11",
        "12","13","14","15","16","17","18","19","20"};
    double ii,can[NUM],max,j;
    int num,i;
    int x,y,k1,k2,iii;
    int graphdriver=HERCMONO,graphmode=HERCMONOHI;
    fp=fopen("hjfcan.txt","r");
}

```

```

k=fgetc(fp);
i=0;
while(fgets(canc,15,fp) !=NULL)
    can[i++] = atof(canc);
max=can[1];
for(i=1;i<=can[0];i++)
    if(max<can[i])
        max=can[i];
initgraph(&graphdriver,&graphmode,"");
setcolor(2);
line(row(0),col(0),row(0),col(330));
line(row(0),col(0),row(450),col(0));
settextjustify(CENTER_TEXT,CENTER_TEXT);
settextstyle(TRIPLEX_FONT,HORIZ_DIR,USER_CHAR_SIZE);
outtextxy(row(0),col(320),yy);
outtextxy(row(450),col(4),xx);
outtextxy(75,320,zz);
j=360/can[0];
x=0,y=0;
setcolor(3);
for(k1=1;k1<=can[0];k1++)
{
    k2=k1%11;
    if(k2==0)
        k2=11;
    ii=can[k1]/max;
    iii=ii * 280;
    line(row(x),col(y),row(x+j),col(iii));
    if(can[0]<20)
    {
        outtextxy(row(x+j),330,xb[k1-1]);
        outtextxy(80,col(280/can[0]*k1),xb[k1-1]);
    }
    x+=j,y=iii;
}
scanf("%c",&k);
closegraph();
}

```

## 参 考 文 献

1. 《IBM PC C 语言简明教程》  
天津科技出版社 李文兵 年璋潮编著  
1988 年 4 月
2. 《IBM PC C 语言例题习题库函数》  
清华大学出版社 李文兵 年璋潮编著  
1990 年 5 月
3. 《C 语言学习指导》  
清华大学出版社 李文兵编著  
1990 年 7 月
4. 《入門 Turbo C》  
(日) 敎学出版社 荒実著  
1988 年 7 月
5. 《PC C 语言教程》  
天津科技出版社 李文兵编著  
1994 年 4 月