

# C 语 言 实 例 解 析 精 粹

曹衍龙 林瑞仲 徐 慧 编著

人 民 邮 电 出 版 社

## 图书在版编目（CIP）数据

C 语言实例解析精粹 / 曹衍龙, 林瑞仲, 徐慧编著. —北京: 人民邮电出版社, 2005.3  
ISBN 7-115-13183-X

I . C... II . ①曹... ②林... ③徐... III . C 语言—程序设计 IV . TP312

中国版本图书馆 CIP 数据核字 (2005) 第 017833 号

## 内容提要

本书共分 8 篇, 分别为基础篇、数据结构篇、数值计算与趣味数学篇、图形篇、系统篇、常见试题解答篇、游戏篇和综合实例篇, 汇集了近 200 个实例, 基本涵盖了目前 C 语言编程的各个方面。

书中以具体的实例为线索, 特别注重对例题的分析、对知识点的归纳、对求解方法的引申, 同时程序代码中融会了 C 语言的各种编程技巧, 条理清晰, 以方便读者举一反三, 开发出符合特定要求的程序。本书的配套光盘中涵盖了书中所有实例的源代码, 以方便读者学习和查阅。

本书适合具有初步 C 语言基础的读者阅读, 可作为高校相关专业的辅导教材, 也可作为 C 语言使用者进行程序设计的实例参考手册。

## C 语言实例解析精粹

- 
- ◆ 编 著 曹衍龙 林瑞仲 徐 慧  
责任编辑 汤 倩
  - ◆ 人民邮电出版社出版发行 北京市崇文区夕照寺街 14 号  
邮编 100061 电子函件 315@ptpress.com.cn  
网址 <http://www.ptpress.com.cn>  
读者热线 010-67132692  
北京密云春雷印刷厂印刷  
新华书店总店北京发行所经销
  - ◆ 开本: 787×1092 1/16  
印张: 25  
字数: 706 千字 2005 年 3 月第 1 版  
印数: 1 - 5 000 册 2005 年 3 月北京第 1 次印刷

ISBN 7-115-13183-X/TP • 4513

---

定价: 44.00 元 (附光盘)

本书如有印装质量问题, 请与本社联系 电话: (010) 67129223

# 前言

当今的计算机软件设计中，无论开发技术如何发展，C 语言作为一种基本的程序语言，仍然是程序开发人员必须掌握的基本功。掌握了 C 语言，不但可以对结构化的编程有全面的了解，而且能够深入理解操作系统的运作方式、内存管理分配方式以及硬件编程控制方法等。

目前流行的许多开发工具，包括微软的 Visual C++ 和 Visual C++ .NET，Borland 公司的 C++ Builder 等开发工具都还遵循着标准 C 语言的基本语法。在很多嵌入式系统的软件设计中，甚至都采用 C 语言来进行开发。

## 为什么写本书

对于 C 语言的学习者来说，最重要的是具备程序设计的能力。初学者往往能读懂别人编写的代码，而自己编写程序时却无所适从，往往不清楚通过程序语言的控制结构如何将简单的计算步骤串联起来完成一项复杂的计算。提高程序设计能力的一个重要途径是学习别人编写的程序，从中掌握解决问题的核心方法和关键步骤，循序渐进，直至自己能够找出算法并编写程序。本书正是为了满足 C 语言学习者的这种需求而策划的，这也是书名“C 语言实例解析精粹”的由来。

## 本书特色

书中以近 200 个实例的编程求解为线索，突出了对例题的分析、对知识点的归纳、对求解方法的引申，力图通过对例题的详细分析，帮助学习者快速提高 C 语言的“程序设计能力”。

## 主要内容

**基础篇**——介绍了 C 语言编程的基础知识，包括第一个程序的建立，基本数据类型的转换，数组、函数、指针、文件、结构、联合的使用等。

**数据结构篇**——介绍了冒泡排序、堆排序、归并排序等各种排序算法，顺序表、双链表、二叉树、图的建立和操作等。

**数值计算和趣味数学篇**——数值计算部分包括多项式求值，线性方程求解以及矩阵运算等。趣味数学部分主要介绍了一些经典问题的求解，包括绘

制余弦曲线，计算高次方数的尾数，求解阿姆斯特朗数，歌德巴赫猜想，素数幻方，爱因斯坦的数学题，三色球问题等。

**图形篇**——介绍如何使用 Turbo C 提供的图形函数，绘制基本的直线、圆弧，设置屏幕颜色、线条类型、填充类型；绘制直线类图形、金刚石、飘带、肾形、心脏形、沙丘等图案；绘制正多边形、递归三角形、抛物样条曲线等；实现图形变换，VGA 编程，分形图，动画设计等。

**系统篇**——介绍如何通过 Turbo C 中的系统调用函数，编写屏幕窗口程序、获取系统各类信息程序以及硬件参数读取程序等。

**常见试题解答篇**——介绍一些常见的 C 语言考试试题的解答方法，比如水果拼盘问题、计算方差、统计符合特定条件的数、字符串倒置、部分排序、产品销售记录处理、求解三角方程、统计选票、数字移位等。

**游戏篇**——介绍了 DOS 环境下的 C 语言游戏编程，包括商人过河游戏、吃数游戏、解救人质游戏、打字训练游戏、双人竞走游戏、迷宫探险游戏、迷你撞球游戏、模拟扫雷游戏、推箱子游戏、五子棋游戏。

**综合实例篇**——介绍了综合 CAD 系统、功能强大的文本编辑器、图书管理系统和进销存管理系统。

书中所有实例均在 Turbo C 2.0 或 Borland C++ 3.1 编译器中调试通过。如果某些程序需要显示汉字，可以通过 Visual C++ 或 Visual C++ .NET 编译运行；如果在 DOS 下运行，需要先运行汉字环境，比如 UCDOS。具体的使用方法参见书中的“光盘使用说明”以及光盘中的相关文档。

本书在编写过程中，参考了《The C Programming Language》等书籍和 Internet 上的相关资源，在此对相关的作者和机构深表谢意。

## **技术支持**

本书主要由曹衍龙、林瑞仲编写，参加写作的人员还有吴越、徐慧、续瑞瑞、张静、程立、吴阳等。在写作过程中，我们力求精益求精，但难免存在一些不足之处，恳请读者批评指正。如果您在使用本书时遇到问题，可以发 E-mail 至 [zjulinruizhong@yahoo.com.cn](mailto:zjulinruizhong@yahoo.com.cn) 和 [tangqian@ptpress.com.cn](mailto:tangqian@ptpress.com.cn) 与我们联系。

编 者  
2005 年 3 月

# 目 录

## 第一部分 基础篇

实例 1 第一个 C 程序.....	2	实例 21 通过指针比较整数大小.....	44
实例 2 求整数之积.....	6	实例 22 指向数组的指针.....	48
实例 3 比较实数大小 .....	8	实例 23 寻找指定元素的指针 .....	50
实例 4 字符的输出.....	10	实例 24 寻找相同元素的指针 .....	52
实例 5 显示变量所占字节数 .....	11	实例 25 阿拉伯数字转换为罗马数字 .....	53
实例 6 自增/自减运算 .....	13	实例 26 字符替换 .....	56
实例 7 数列求和.....	14	实例 27 从键盘读入实数.....	57
实例 8 乘法口诀表.....	17	实例 28 字符行排版 .....	59
实例 9 猜数字游戏.....	19	实例 29 字符排列 .....	60
实例 10 模拟 ATM (自动柜员机) 界面.....	22	实例 30 判断字符串是否回文 .....	62
实例 11 用一维数组统计学生成绩 .....	24	实例 31 通讯录的输入输出 .....	63
实例 12 用二维数组实现矩阵转置.....	26	实例 32 扑克牌的结构表示 .....	68
实例 13 求解二维数组的最大/最小元素 .....	29	实例 33 用“结构”统计学生成绩 .....	69
实例 14 利用数组求前 n 个质数 .....	31	实例 34 报数游戏 .....	72
实例 15 编制万年历.....	33	实例 35 模拟社会关系 .....	73
实例 16 对数组元素排序 .....	36	实例 36 统计文件的字符数 .....	74
实例 17 任意进制数的转换 .....	37	实例 37 同时显示两个文件的内容 .....	80
实例 18 判断回文数.....	39	实例 38 简单的文本编辑器 .....	81
实例 19 求数组前 n 元素之和.....	41	实例 39 文件的字数统计程序 .....	82
实例 20 求解钢材切割的最佳订单.....	42	实例 40 学生成绩管理程序 .....	85

## 第二部分 数据结构篇

实例 41 插入排序.....	96	实例 42 希尔排序 .....	100
-----------------	----	------------------	-----

实例 43 冒泡排序	102	实例 52 二叉树遍历	130
实例 44 快速排序	105	实例 53 浮点数转换为字符串	132
实例 45 选择排序	109	实例 54 汉诺塔问题	133
实例 46 堆排序	111	实例 55 哈夫曼编码	135
实例 47 归并排序	115	实例 56 图的深度优先遍历	138
实例 48 基数排序	119	实例 57 图的广度优先遍历	139
实例 49 顺序表插入和删除	123	实例 58 求解最优交通路径	141
实例 50 链表操作	126	实例 59 八皇后问题	143
实例 51 双链表	129	实例 60 骑士巡游	145

### 第三部分 数值计算与趣味数学篇

实例 61 绘制余弦曲线和直线的迭加	150	实例 80 求 $\pi$ 的近似值	173
实例 62 计算高次方数的尾数	151	实例 81 奇数平方的有趣性质	175
实例 63 打鱼还是晒网	151	实例 82 角谷猜想	176
实例 64 怎样存钱以获取最大利息	154	实例 83 四方定理	177
实例 65 阿姆斯特朗数	155	实例 84 卡布列克常数	178
实例 66 亲密数	156	实例 85 尼科彻斯定理	179
实例 67 自守数	157	实例 86 扑克牌自动发牌	180
实例 68 具有 $abcd=(ab+cd)^2$ 性质的数	158	实例 87 常胜将军	181
实例 69 验证歌德巴赫猜想	159	实例 88 搬山游戏	182
实例 70 素数幻方	161	实例 89 兔子产子（菲波那契数列）	183
实例 71 百钱百鸡问题	163	实例 90 数字移动	184
实例 72 爱因斯坦的数学题	164	实例 91 多项式乘法	186
实例 73 三色球问题	165	实例 92 产生随机数	189
实例 74 马克思手稿中的数学题	166	实例 93 堆栈四则运算	190
实例 75 配对新郎和新娘	167	实例 94 递归整数四则运算	196
实例 76 约瑟夫问题	168	实例 95 复平面作图	199
实例 77 邮票组合	169	实例 96 绘制彩色抛物线	200
实例 78 分糖果	170	实例 97 绘制正态分布曲线	203
实例 79 波瓦松的分酒趣题	172	实例 98 求解非线性方程	206

实例 99 实矩阵乘法运算	209
实例 100 求解线性方程	211
实例 101 $n$ 阶方阵求逆	215
实例 102 复矩阵乘法	219
实例 103 求定积分	220
实例 104 求满足特异条件的数列	221
实例 105 超长正整数的加法	222

## 第四部分 图形篇

实例 106 绘制直线	226
实例 107 绘制圆	230
实例 108 绘制圆弧	231
实例 109 绘制椭圆	232
实例 110 设置背景色和前景色	233
实例 111 设置线条类型	235
实例 112 设置填充类型和填充颜色	237
实例 113 图形文本的输出	238
实例 114 金刚石图案	240
实例 115 飘带图案	241
实例 116 圆环图案	242
实例 117 肾形图案	243
实例 118 心脏形图案	244
实例 119 渔网图案	245
实例 120 沙丘图案	246
实例 121 设置图形方式下的文本类型	246
实例 122 绘制正多边形	248
实例 123 正六边形螺旋图案	249
实例 124 正方形螺旋拼块图案	251
实例 125 图形法绘制圆	252
实例 126 递归法绘制三角形图案	254
实例 127 图形法绘制椭圆	255
实例 128 抛物样条曲线	257
实例 129 Mandelbrot 分形图案	259
实例 130 绘制布朗运动曲线	261
实例 131 艺术清屏	262
实例 132 矩形区域的颜色填充	263
实例 133 VGA256 色模式编程	265
实例 134 绘制蓝天图案	266
实例 135 屏幕检测程序	267
实例 136 运动的小车动画	268
实例 137 动态显示位图	269
实例 138 利用图形页实现动画	270
实例 139 图形时钟	271
实例 140 音乐动画	274

## 第五部分 系统篇

实例 141 读取 DOS 系统中的国家信息	278
实例 142 修改环境变量	279
实例 143 显示系统文件表	280
实例 144 显示目录内容	282
实例 145 读取磁盘文件	284
实例 146 删除目录树	286
实例 147 定义文本模式	287
实例 148 设计立体窗口	290

实例 149 彩色弹出菜单	292
实例 150 读取 CMOS 信息	293
实例 151 获取 BIOS 设备列表	294
实例 152 锁住硬盘	295
实例 153 备份/恢复硬盘分区表	297
实例 154 设计口令程序	298
实例 155 程序自我保护	300

## 第六部分 常见试题解答篇

实例 156 水果拼盘	304
实例 157 小孩吃梨	305
实例 158 删除字符串中的特定字符	306
实例 159 求解符号方程	307
实例 160 计算方差	308
实例 161 求取符合特定要求的素数	309
实例 162 统计符合特定条件的数	310
实例 163 字符串倒置	312
实例 164 部分排序	314
实例 165 产品销售记录处理	316
实例 166 特定要求的字符编码	318
实例 167 求解三角方程	320
实例 168 新完全平方数	321
实例 169 三重回文数	323
实例 170 奇数方差	324
实例 171 统计选票	326
实例 172 同时整除	328
实例 173 字符左右排序	329
实例 174 符号算式求解	331
实例 175 数字移位	333
实例 176 统计最高成绩	334

## 第七部分 游戏篇

实例 177 商人过河游戏	338
实例 178 吃数游戏	340
实例 179 解救人质游戏	341
实例 180 打字训练游戏	344
实例 181 双人竞走游戏	346
实例 182 迷宫探险游戏	349
实例 183 迷你撞球游戏	351
实例 184 模拟扫雷游戏	353
实例 185 推箱子游戏	357
实例 186 五子棋游戏	359

## 第八部分 综合实例篇

实例 187 综合 CAD 系统	362
实例 188 功能强大的文本编辑器	368
实例 189 图书管理系统	381
实例 190 进销存管理系统	385

# 01

## 第一部分 基 础 篇

### 精彩导读

- 第一个 C 程序
- 数列求和
- 模拟 ATM 界面
- 编制万年历
- 任意进制数的转换
- 文件的字数统计

# 实例 1 第一个 C 程序

## 实例说明

本实例将建立第一个 C 程序，通过使用 Borland 公司的 Turbo C 2.0 软件新建一个 C 程序，在该程序中添加代码，使其能够输出“Hello World!”的问候语。本实例重点介绍如何创建、编译、调试和运行 C 程序。

程序运行效果如图 1-1 所示。

C 语言程序的编译器有很多，如 Turbo C 2.0、Turbo C 3.0、Borland C++ 3.1、Visual C++ 6.0 和 Visual C++.NET 等开发环境都可以编译运行 C 程序。其中，最常用的是 Turbo C 2.0（简称 TC）。本书配套光盘上的【Turbo C 2.0】目录下有该编译器的自解压文件 tc.exe。要在硬盘上安装 TC 编译器，可按如下步骤进行：

- 【1】将配套光盘【Turbo C 2.0】目录下的 tc.exe 文件拷贝到硬盘上，比如 C 盘根目录下。
- 【2】双击硬盘上的 tc.exe 文件，出现如图 1-2 所示的自解压文件向导。
- 【3】采用默认的安装目录【C:】，单击【安装】按钮，完成安装。



图 1-1 第一个 C 程序运行效果



图 1-2 Turbo C 2.0 的安装向导

## 实例解析

安装好 TC 编译器之后，就可以按照如下步骤创建第一个 C 程序。

- 【1】双击 TC 安装目录（比如【C:\TC】目录）下的 tc.exe 程序，打开 TC 开发环境。
- 【2】此时 TC 编译器会自动创建一个 NONAME.C 的 C 程序，如图 1-3 所示。
- 【3】在该文件中添加【程序代码】中所示的内容。
- 【4】使用【Alt+F】组合键激活 File 菜单，通过上下光标键选择“Save”命令，或用快捷键“F2”打开如图 1-4 所示的保存文件对话框，输入保存路径和文件名，按回车键，完成保存。

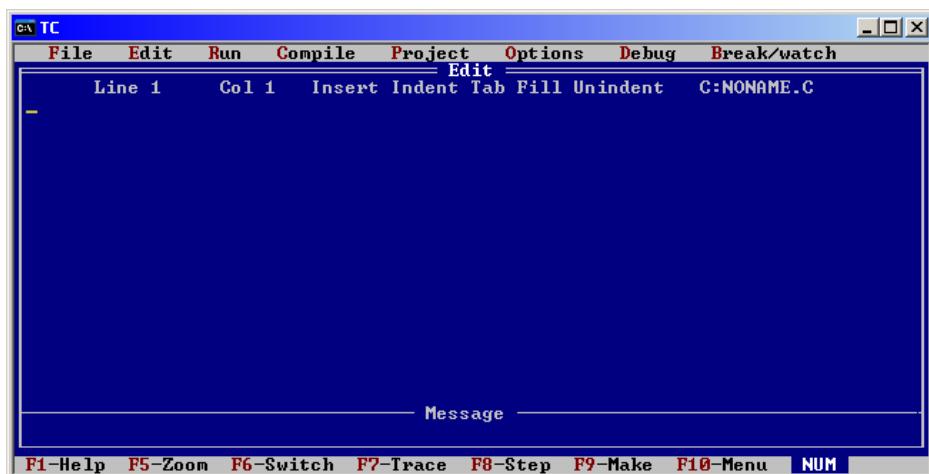


图 1-3 TC 开发环境



图 1-4 保存程序文件

【5】执行菜单命令“Compile | Compile to OBJ”（见图 1-5），可将程序编译为 OBJ 对象文件，此时编译器将对程序的语法语义进行检查，若有错误或警告，将给出提示。如图 1-6 所示是编译成功的例子，图 1-7 所示是将 printf 语句后面的用于表示语句结束的分号 “;” 去掉后出现的错误信息。

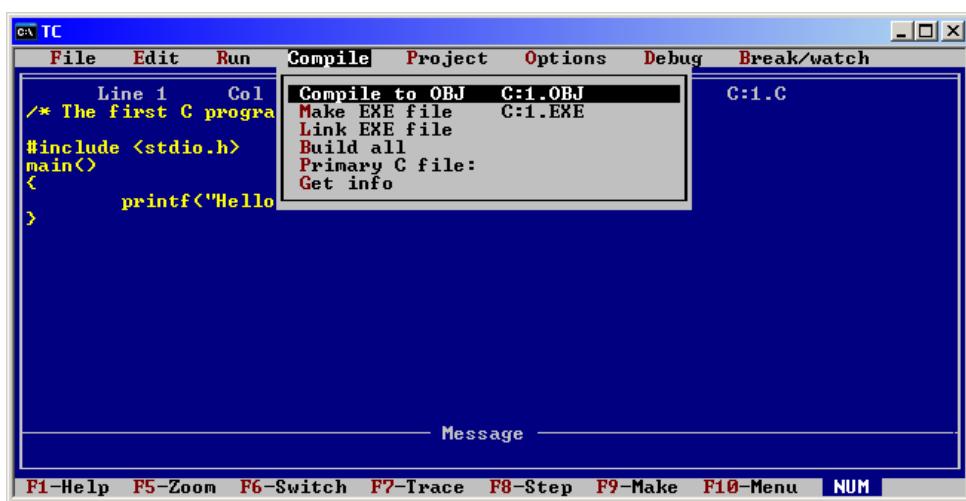


图 1-5 编译程序

The screenshot shows the emTC IDE interface. The code editor window contains a C program:

```

Line 1 Col 1 Insert Indent Tab Fill Unindent C:1.C
/* The first C programme */

#include <stdio.h>
main()
{
    printf("He

```

A modal dialog box titled "Compiling" displays the following information:

```

Main file: 1.C
Compiling: EDITOR → 1.C

Total      File
Lines compiled: 221   221
Warnings: 0       0
Errors: 0        0

Available memory: 245K
Success : Press any key

```

The status bar at the bottom shows keyboard shortcuts: F1-Help, F5-Zoom, F6-Switch, F7-Trace, F8-Step, F9-Make, F10-Menu, NUM.

图 1-6 编译成功的信息

The screenshot shows the emTC IDE interface. The code editor window contains a C program:

```

Error: Statement missing ; in function main
/* The first C programme */

#include <stdio.h>
main()
{
    printf("Hello World!");
}

```

A modal dialog box titled "Message" displays the following error message:

```

Compiling C:\TC\1.C:
•Error C:\TC\1.C 7: Statement missing ; in function main

```

The status bar at the bottom shows keyboard shortcuts: F1-Help, F5-Zoom, F6-Switch, F7-Trace, F8-Step, F9-Make, F10-Menu, NUM.

图 1-7 编译时报错的信息

【6】执行菜单命令“Compile | Make EXE File”，生成可执行文件，如图 1-8 所示。

The screenshot shows the emTC IDE interface. The code editor window contains a C program:

```

Line 6 Col 1
/* The first C programme */

#include <stdio.h>
main()
{
    printf("Hello
}

```

The "Compile" menu is open, showing the following options:

- Compile to OBJ C:1.OBJ
- Make EXE file C:1.EXE**
- Link EXE file
- Build all
- Primary C file:
- Get info

The status bar at the bottom shows keyboard shortcuts: F1-Help, F5-Zoom, F6-Switch, F7-Trace, F8-Step, F9-Make, F10-Menu, NUM.

图 1-8 生成可执行文件

【7】执行菜单命令“Run | Run”，或按快捷键【Ctrl + F9】，运行程序，如图 1-9 所示。程序运行结果见图 1-1。



图 1-9 运行程序

**【8】**要查看程序运行结果，可以执行菜单命令“File | OS Shell”切换到命令行下查看结果，如图 1-10 所示。在命令行下键入“exit”，可以返回 TC 开发环境。



图 1-10 查看运行结果

## 程序代码

### 【程序 1】 第一个 C 程序

```
/* The first C programme */
#include <stdio.h>           /* 包含标准输入输出头文件 */
main()                      /* 主函数 */
{
    printf("Hello World!\n"); /* 打印输出信息 */
}
```

## 归纳注释

#include 语句用于包含头文件 stdio.h，即定义标准输入输出函数的头文件。

每一个 C 程序必须有且只有一个 main() 主函数，这样程序运行时才可以找到入口。

本程序通过 printf 函数向标准的显示终端打印输出字符“Hello World!”。该程序仅打印这些字符，因此光标此时还停在同一行。通常程序打印输出一行后，需要回车换行，这里可以通过加入控制符 “\n” 来实现，即 printf("Hello World!\n");。

TC 编译器使用的其他问题，可以通过运行 TC 安装目录下的 Readme.exe 帮助文件来了解。

基本上所有的功能都可以通过其菜单实现。相信读者只要多尝试，多使用这些菜单命令，很快就可以掌握 TC 开发环境。

值得注意的是，TC 开发环境中，无法显示中文，而在 Visual C++ 6.0 及 Visual C++.NET 等编译器中支持中文，且运行时也能够显示。读者若需要应用中文，可采用这些开发环境。比如上面的 printf 语句改为“printf(“世界，您好！”);”，则显示在命令行下即为“世界，您好！”。本书的实例程序中的代码尽量不使用中文，以便与 TC 编译器兼容，而其后的注释则采用中文，这样虽然在 TC 编译器中无法看到中文（看到的是乱码），但用记事本等其他文本阅读工具打开，则可以看到中文，以便增加程序的可读性。

## 实例 2 求整数之积

### 实例说明

从键盘输入两个整数，输出它们的积。通过本实例，读者可以理解从键盘读取输入的数据以及输出整型变量等方法。程序运行结果如图 2-1 所示。



图 2-1 实例 2 程序运行结果

### 实例解析

C 语言提供的数据结构是以数据类型形式出现的。C 语言的数据类型可分为基本类型、构造类型、指针类型 3 大类。其中基本类型包括整型（变量声明关键字为 int），字符型（变量声明关键字为 char），实型（即浮点型，分为单精度浮点型 float 和双精度浮点型 double float），枚举类型（变量声明关键字为 enum）。构造类型包括数组、结构体和共用体 3 种。

整型、字符型和实型都有常量和变量之分。比如 12、0、-3 为整型常量，4.6、-1.23 为实型常量，'a'、'D' 为字符常量。常量一般从其字面形式即可判别，也可以用一个标识符代表一个常量，比如用预定义#define PRICE 30，这样可提高程序的可读性和可维护性。

值可以改变的量称为变量。变量名只能由字母、数字和下划线组成，且第一个字符必须为字母或下划线。整型变量可分为基本型（用 int 表示）、短整型（用 short int 或 short 表示）、长整型（用 long int 或 long 表示）和无符号型（包括无符号整型、无符号短整型、无符号长整型，分别用 unsigned int、unsigned short 和 unsigned long 表示）。

各种整型变量在不同的计算机机型上存放的内存字节数不同，典型的 IBM PC 中所占的位（bit）和数的范围如表 2-1 所示。整型变量的定义为：

```
int a,b; /* 定义变量 a, b 为整型 */  
unsigned short c,d; /* 定义变量 c, d 为无符号短整型 */
```

```
long e,f; /* 定义变量 e, f 为长整型 */
```

表 2-1 整型变量所占位数和数的范围

数据类型	所占位数	数的范围	
int	16	-32768~32768	即 $-2^{15} \sim (2^{15}-1)$
Short [int]	16	-32768~32768	即 $-2^{15} \sim (2^{15}-1)$
Long [int]	32	-2147483648~2147483647	即 $-2^{31} \sim (2^{31}-1)$
unsigned [int]	16	0~65535	即 $0 \sim (2^{16}-1)$
unsigned short	16	0~65535	即 $0 \sim (2^{16}-1)$
unsigned long	32	0~4294967295	即 $0 \sim (2^{32}-1)$

在本例中，设两个整数分别为 x、y，它们的乘积为 m；程序首先调用 printf() 函数，提示用户输入数据，然后调用 scanf() 函数，输入变量 x 和 y 的值，接着求 x 与 y 的积 m，最后输出结果。上述过程用算法描述如下：

**【算法】** 读入两个整数，输出它们的积

```
{  
    提示用户输入数据;  
    输入变量 x 和 y 的值;  
    计算乘积;  
    输出乘积;  
}
```

## 程序代码

**【程序 2】** 求整数之积

```
/* Input two numbers, output the product */  
#include <stdio.h>  
main()  
{  
    int x,y,m; /* 定义整型变量 x, y, m */  
    printf("Please input x and y\n"); /* 输出提示信息 */  
    scanf("%d%d",&x,&y); /* 读入两个乘数，赋给 x, y 变量 */  
    m=x*y; /* 计算两个乘数的积，赋给变量 m */  
    printf("%d * %d = %d\n",x,y,m); /* 输出结果 */  
}
```

## 归纳注释

本实例程序实现的是两个整数的简单乘积，同样的，也可以通过修改，实现简单的整数四则运算。整型变量包括整型、短整型、长整型和无符号整型，在 printf() 和 scanf() 函数中的格式说明都可以是 %d。

printf() 输出函数（关键字中的 f 就是表示 format，格式化的意思）用于输出变量的值时，调用格式为“printf(格式控制, 输出表列);”，“格式控制”是用双引号括起来的字符串，包括格式字符串和普通字符。格式字符串是以 % 开头的，在 % 后跟各种格式字符，以说明输出数据的类型、形式、长度、小数位数等。如“%d”表示按十进制整型输出，“%ld”表示按十进制长整型输出等。

格式字符串。在 Turbo C 中，格式字符串的一般形式为：[标志][输出最小宽度][.精度][长度]类型。其中方括号[]中的项为可选项。各项的意义如下：①类型字符用以表示输出数据的类型，d

——以十进制形式输出带符号整数(正数不输出符号); o——以八进制形式输出无符号整数(不输出前缀 O); x——以十六进制形式输出无符号整数(不输出前缀 OX); u——以十进制形式输出无符号整数; f——以小数形式输出单、双精度实数; e——以指数形式输出单、双精度实数; g——以%f%e 中较短的输出宽度输出单、双精度实数; c——输出单个字符; s——输出字符串。②标志字符为-、+、#、空格 4 种，——结果左对齐，右边填空格；+——输出符号（正号或负号），输出值为正时冠以空格，为负时冠以负号；#——对 c、s、d、u 类无影响；对 o 类，在输出时加前缀 o；对 x 类，在输出时加前缀 0x；对 e、g、f 类当结果有小数时才给出小数点。③输出最小宽度：用十进制整数来表示输出的最少位数。若实际位数多于定义的宽度，则按实际位数输出，若实际位数少于定义的宽度则补以空格或 0。④精度格式符以“.”开头，后跟十进制整数。如果输出数字，则表示小数的位数；如果输出的是字符，则表示输出字符的个数；若实际位数大于所定义的精度数，则截去超过的部分。⑤长度格式符为 h、l 两种，h 表示按短整型量输出，l 表示按长整型量输出。

scanf()为输入函数，即按用户指定的格式从键盘上把数据输入到指定的变量之中。scanf 函数的一般形式为“scanf(“格式控制字符串”，地址表列);”其中，格式控制字符串的作用与 printf 函数相同，但不能显示普通字符串，也就是不能显示提示字符串。地址表列中给出各变量的地址。地址是由地址运算符“&”后跟变量名组成的。例如，“&a,&b”分别表示变量 a 和变量 b 的地址。这个地址就是编译器在内存中给 a 和 b 变量分配的地址。

## 实例 3 比较实数大小

### 实例说明

从键盘输入两个实数，输出它们中比较大的数。通过本实例，可以学习如何从键盘读取输入的实数，如何实现实数的大小比较，以及实型变量的基本概念等。程序运行结果如图 3-1 所示。

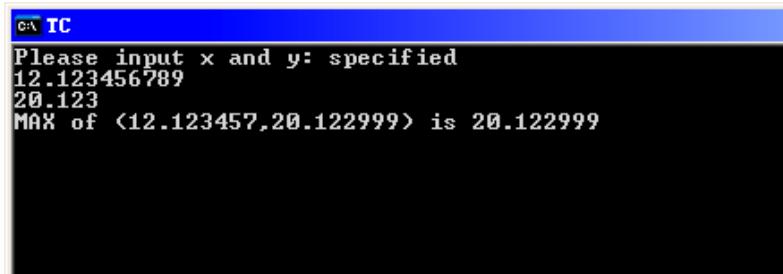


图 3-1 实例 3 程序运行结果

### 实例解析

实型变量分为单精度（float 型）和双精度（double 型）两类。对于每一个实型变量在使用前都应定义。例如：

```
float x,y ;           /* 指定 x、y 为单精度实数 */
double z ;             /* 指定 z 为双精度实数 */
```

在一般系统中，一个 float 型数据在内存中占用 4 个字节（32 位），一个 double 型数据占 8 个字节。单精度实数提供 7 位有效数字，双精度实数提供 15~16 位有效数字，数值的范围随机器系

统而异。在 IBM PC 中，单精度实数的数值范围约为  $10^{-38} \sim 10^{38}$ ，双精度实数范围约为  $10^{-308} \sim 10^{308}$ 。

本实例先定义 3 个单精度浮点型变量，用于存放 x, y 和它们中的大数 c。接着，提示用户已经输入的信息，通过条件运算符来比较大小，得到 c，最后输出结果。

条件运算符有 3 个操作对象，称三目（元）运算符，它是 C 语言中惟一的一个三目运算符。条件表达式的一般形式为：

表达式 1 ? 表达式 2 : 表达式 3

### 条件表达式的说明

(1) 条件表达式的执行顺序：先求解表达式 1，若为真（非 0）则求解表达式 2，并将表达式 2 的值作为条件表达式的值，若表达式 1 为假（0），则求解表达式 3，并将表达式 3 的值作为条件表达式的值。

max=(a>b)?a:b;

执行结果就是将条件表达式的值赋给 max，即将 a, b 中比较大的值赋给 max。

(2) 条件运算符优先于赋值运算符，因此上面的赋值表达式的求解过程是先求解条件表达式，再将它的值赋给 max。

条件表达式的优先级比关系运算符和算术运算符都低，因此，

max=(a>b)?a:b;

可以写成

max=a>b?a:b;

又比如

a>b?a:b+1;

相当于 a>b?a:(b+1)，而不等价于(a>b?a:b)+1。

(3) 条件运算符的结合方向为“自右向左”。比如

a>b?a:c>d?c:d;

相当于 a>b?a:(c>d?c:d)，如果 a=1, b=2, c=3, d=4，则条件表达式的值等于 4。

(4) 条件表达式中，表达式 1 的类型可以与表达式 2 和表达式 3 的类型不同。如

x?'a':'b';

x 是整型变量，若 x=0，则条件表达式的值为 b。表达式 2 和表达式 3 的类型也可以不同，此时条件表达式的值的类型为二者中较高的类型。比如

x>y?1:1.5;

如果 x 小于等于 y，则条件表达式的值为 1.5，若 x 大于 y，则值为 1，由于 1.5 为实型，比整型高，因此将 1 转换为实型 1.0。



## 程序代码

### 【程序 3】 比较实数大小

```
/* 输入两个浮点数，输出其中较大的数 */
#include <stdio.h>
main()
{
    float x,y,c; /* 变量定义 */
    printf("Please input x and y:\n"); /* 提示用户输入数据 */
    scanf("%f%f",&x,&y);
    c=x>y?x:y; /* 计算 c=max(x,y) */
    printf("MAX of (%f,%f) is %f",x,y,c); /* 输出 c */
}
```



## 归纳注释

在本例中，由于 `x`, `y` 变量定义为 `float` 型，在输出定义没有说明小数点后面有几位时，系统的默认值为 6 位小数，因此，图 3-1 的输出结果为精确到 6 位。

由于实型变量的小数位具有默认值，而且系统会取近似值，因此在进行精确度要求较高的运算时，必须指定小数位数。

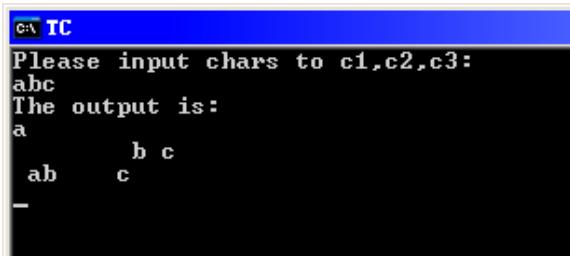
条件表达式在计算最大值最小值时比较方便，还可以计算 3 个数的最值，比如：  
`max(x,y,z)=(x>y?x:y)>z?(x>y?x:y):z;`

# 实例 4 字符的输出



## 实例说明

从键盘输入字符，再输出它们。通过本实例，可以学习如何从键盘读取输入的字符，如何实现字符的输出，以及字符串、字符常量、转义字符的基本概念等。程序运行结果如图 4-1 所示。



```
C:\> TC
Please input chars to c1,c2,c3:
abc
The output is:
a
      b c
ab    c
-
```

图 4-1 实例 4 程序运行结果



## 实例解析

字符变量的定义是“`char c1,c2;`”，字符变量的赋值可以用 `c1='a'` 来赋值，也可以从键盘输入，其格式说明符为“`%c`”。

字符常量是用单引号括起来的一个字符。例如'`'a'`、'`'b'`、'`'=`'、'`+'`、'`'?`'都是合法字符常量。在 C 语言中，字符常量有以下特点。

- (1) 字符常量只能用单引号括起来，不能用双引号或其他括号。
- (2) 字符常量只能是单个字符，不能是字符串。
- (3) 字符可以是字符集中任意字符。但数字被定义为字符型之后就不能参与数值运算。如'`'5'`和 `5` 是不同的。`'5'`是字符常量，不能参与运算。

字符串常量是由一对双引号括起的字符序列。例如"`CHINA`"、"`C program`"、"`$12.5`"等都是合法的字符串常量。字符串常量和字符常量是不同的量，它们之间主要有以下区别。

- (1) 字符常量由单引号括起来，字符串常量由双引号括起来。
- (2) 字符常量只能是单个字符，字符串常量则可以含一个或多个字符。
- (3) 可以把一个字符常量赋予一个字符变量，但不能把一个字符串常量赋予一个字符变量。
- (4) 字符常量占一个字节的内存空间。字符串常量占的内存字节数等于字符串中字节数加 1。增加的一个字节中存放字符"`\0`"(ASCII 码为 0)。这是字符串结束的标志。例如，字符串"`C program`"

在内存中所占的字节为 C program\0。字符常量'a'和字符串常量"a"虽然都只有一个字符，但在内存中的情况是不同的。'a'在内存中占一个字节，可表示为 a。"a"在内存中占两个字节，可表示为 a\0。

转义字符是一种特殊的字符常量。转义字符以反斜线"\\"开头，后跟一个或几个字符。转义字符具有特定的含义，不同于字符原有的意义，故称“转义”字符。例如，在前面例题 printf() 函数的格式串中用到的 “\n” 就是一个转义字符，其意义是“回车换行”。转义字符主要用来表示那些用一般字符不便于表示的控制代码。常用的转义字符及其含义如表 4-1 所示。

表 4-1 常用转义字符

转义字符	转义字符的意义	转义字符	转义字符的意义
\n	回车换行	\\	反斜线符"\\"
\t	横向跳到下一制表位置	'	单引号符
\v	竖向跳格	\a	鸣铃
\b	退格	\ddd	1~3 位八进制数所代表的字符
\r	回车	\xhh	1~2 位十六进制数所代表的字符
\f	走纸换页		

广义地讲，C 语言字符集中的任何一个字符均可用转义字符来表示，\ddd 和\xhh 正是为此而提出的。ddd 和 xhh 分别为八进制和十六进制的 ASCII 代码。如\101 表示字母"A"，\102 表示字母"B"，\134 表示反斜线，\XOA 表示换行等。

本例先定义 3 个字符变量，用于保存输入的字符，接着输出字符串、转义字符和字符变量。

## 程序代码

### 【程序 4】 字符的输出

```
/* 输入字符，输出通过转义字符控制的字符 */
#include <stdio.h>
main()
{
    char c1,c2,c3;
    printf("Please input chars to c1,c2,c3:\n");
    scanf ("%c%c%c",&c1,&c2,&c3);
    printf ("%s\n%c\n\t%c %c\n %c%c\t\b%c\n", "The output is:",c1,c2,c3,c1,c2,c3);
}
```

## 归纳注释

程序中，首先输出一个字符串，接着是“\n” 回车换行；在第一列输出 c1 值之后就是“\n”；接着又是“\t”，于是跳到下一制表位置（设制表位置间隔为 8），再输出 c2 值；空一格再输出 c3 后又是“\n”，因此再回车换行；再空一格之后又输出 c1 值；接着输出 c2 的值，跳到下一制表位置（与上一行的 c2 对齐），但下一转义字符“\b”又退回一格，故紧挨着 c2 再输出 c3 值。

## 实例 5 显示变量所占字节数

## 实例说明

通过程序显示 int、long、float、double 等变量在计算机系统中所占的位数。程序运行结果如图 5-1 所示。

```
The bytes of the variables are:  
int:2 bytes  
char:1 byte  
short:2 bytes  
long:4 bytes  
float:4 bytes  
double:8 bytes  
long double:10 bytes
```

图 5-1 实例 5 程序运行结果

## 实例解析

前面已经说明整型变量在内存中所占的字节数（1字节=8位）。不同的编译器，变量所占的字节数也不同。在 TC 编译器中，一般整型 int 变量占 2 字节，char 变量占 1 字节，short 整型占 2 字节，long 整型占 4 字节，float 型占 4 字节，double 型占 8 字节，long double 型占 8、10 或 12 字节（根据不同的系统）。

程序先输出提示信息，接着依次输出各个类型变量所占的字节数。

## 程序代码

### 【程序 5】 显示变量所占的字节数

```
/* 输出不同类型所占的字节数 */  
#include <stdio.h>  
void main()  
{  
    printf("The bytes of the variables are:\n");  
    /*int 型在不同的 PC 机，不同的编译器中的字节数不一样，*/  
    /*一般来说在 TC2.0 编译器中字节数为 2，在 VC 编译器中字节数为 4 */  
    printf("int:%d bytes\n", sizeof(int));  
    /* char 型的字节数为 1 */  
    printf("char:%d byte\n", sizeof(char));  
    /* short 型的字节数为 2 */  
    printf("short:%d bytes\n", sizeof(short));  
    /* long 型的字节数为 4 */  
    printf("long:%d bytes\n", sizeof(long));  
    /* float 型的字节数为 4 */  
    printf("float:%d bytes\n", sizeof(float));  
    /* double 型的字节数为 8 */  
    printf("double:%d bytes\n", sizeof(double));  
    /* long double 型的字节数为 8 或 10 或 12 */  
    printf("long double:%d bytes\n", sizeof(long double));  
}
```

## 归纳注释

sizeof()是保留字，它的作用是求某类型或某变量类型的字节数，括号中可以是类型保留字或变量。比如，int a=1; sizeof(a)=2。

# 实例 6 自增/自减运算

## 实例说明

通过整型变量的自增运算结果来显示自增运算符的使用方法。程序运行结果如图 6-1 所示。

```
C:\ TC
a = 6, b = 6, c = 6
i, i++, i++ = 12,11,10
13
12
12
13
-12
-13
```

图 6-1 实例 6 程序运行结果

## 实例解析

自增 1 运算符记为 “`++`”，其功能是使变量的值自增 1。自减 1 运算符记为 “`--`”，其功能是使变量值自减 1。自增 1，自减 1 运算符均为单目运算，都具有右结合性。可有以下几种形式：

<code>++i</code>	i 自增 1 后再参与其他运算
<code>--i</code>	i 自减 1 后再参与其他运算
<code>i++</code>	i 参与运算后，值再自增 1
<code>i--</code>	i 参与运算后，值再自减 1

在使用上容易出错的是 `i++` 和 `i--`。特别是当它们出现在较复杂的表达式或语句中时，常常难于弄清。

本实例程序先定义变量 `a`、`b`、`c` 和 `i` 并初始化 `a` 和 `i`，接着赋值给 `b`、`c`，输出 `a`、`b`、`c`，最后输出 `i` 的各种自增自减运算结果。

## 程序代码

### 【程序 6】 自增/自减运算

```
/* 自增运算符的应用 */
#include <stdio.h>
main()
{
    int a=5,b,c,i=10;      /* 变量定义初始化 */
    b=a++;                  /* a 赋给 b 后 a 自增 1 */
    c=++b;                  /* b 自增 1，后赋给 c */

    printf("a = %d, b = %d, c = %d\n",a,b,c);    /* 输出 abc */
    printf("i, i++, i++ = %d,%d,%d\n",i,i++,i++); /* 输出 i, i++, i++ */
    printf("%d\n",++i);        /* i 自增 1，输出 i，此时 i=12+1 */
    printf("%d\n",--i);        /* i 自减 1，输出 i，i=13-1=12 */
    printf("%d\n",i++);        /* 输出 i=12, i 自增 1, i=13 */
    printf("%d\n",i--);        /* 输出 i=13, i 自减 1, i=12 */
```

```
printf("%d\n",-i++);      /* -(i++)即 i 取出, 加负号, 输出-12, i 取出前已自增 1, i=13*/
printf("%d\n",-i--);      /* i 取出, 加负号, 输出-13, i 取出前已再自减 1, i=12*/
}
```



## 归纳注释

自增自减运算符具有右结合性，在本实例程序中的第二个 printf 语句，运算顺序是自右向左，计算最右边的 i++，即输出 i=10，再对 i 自增 1，此时 i=11，再计算中间的 i++，即输出 i=11，i 自增 1，此时 i=12，最后输出 i=12。

自增运算符通常运用与 for, while 等循环语句中，对循环变量进行自增或自减。

# 实例 7 数列求和



## 实例说明

计算  $1+1+2+1+2+3+1+2+3+4+\dots+1+2+\dots+n$  的值。通过该实例，可以学习 if 条件判断语句和 for 循环语句的应用。程序运行结果如图 7-1 所示。

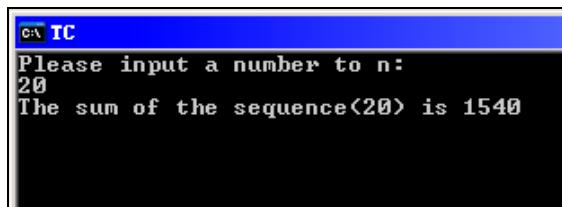


图 7-1 实例 7 程序运行结果



## 实例解析

### if 条件语句

if 语句有以下 3 种形式。

#### (1) if 语句

```
if(x>y) printf("%d",x);
```

其中，if 语句就是根据  $x>y$  这个表达式的值来决定是否要执行后面的 printf 语句，当表达式的值为真 ( $x>y$ ) 时，就执行，当表达式的值为假 ( $x \leq y$ ) 时，就跳过 printf 语句而直接执行后面的语句。

#### (2) if...else 语句

```
if(x>y) printf("%d",x);
else printf("%d",y);
```

如果  $x>y$  成立，则打印出 x，否则打印出 y。与第一种 if 语句的不同之处在于，不论表达式的值是真是假，if 语句都分别给它们安排了任务。也就是说无论 x, y 哪一个大，总能打印出一个数。

#### (3) if...elseif 语句

```
if(count>500)          price = 0.15;
else if(count>300)      price = 0.10;
else if (count>100)     price = 0.05;
```

```
else price = 0;
```

这段程序的意思是，如果 count 的值是在 500 以上，那么 price 等于 0.15；如果 count 的值在 300 到 500 之间（else if 可以理解为否则如果，即在不大于 500 的条件下如果满足大于 300 的条件，也就是 300 到 500 之间），那么 price 等于 0.10；如果 count 的值在 100 到 300 之间，那么 price 等于 0.05；如果 count 在 0 到 100 之间，那么 price 等于 0。这种形式的 if 语句比前两种都要复杂，它能处理的分支也多，这个例子就能处理 4 个分支，而且分支的数目还可以无限扩展。

尽管 if 语句看起来很简单，但在实际的应用中还是有许多问题需要注意的。

(1) 不要忘记在每个语句后面加分号。分号是 C 语句中不可缺少的部分，任何语句，不管它是不是作为 if 语句的一部分，结尾都要加上分号表示一个语句的结束。如果忘记在语句末尾加分号，则会出现语法错误。

(2) 要注意 if 语句的嵌套问题。

在一个 if 语句中又包含一个或多个 if 语句时，就形成了嵌套 if 语句。嵌套 if 语句的一般形式如下：

```
if (表达式 1)
    if (表达式 2)    语句 1
    else            语句 2
else
    if (表达式 3)    语句 3
    else            语句 4
```

要注意嵌套 if 语句中 if 与 else 的匹配问题。记住一个原则：else 语句总是与前面最近的未匹配的 if 语句相配对。按照这个原则，上面第 3 行的 else 跟第 2 行的 if 相匹配，第 4 行的 else 跟第 1 行的 if 相匹配，第 6 行的 else 跟第 5 行的 if 相匹配。而初学者往往容易忘记这个原则。容易出错的形式一般是这样的：

```
if (表达式 1)
    if (表达式 2)    语句 1
else
    if (表达式 3)    语句 2
    else            语句 3
```

这种形式的程序很可能是想让第 3 行的 else 跟第 1 行的 if 相匹配，可事实上按照 else 总是跟最近的 if 相匹配的原则，第 3 行的 else 是跟第 2 行的 if 相匹配的。那怎样实现原来的意图呢，可以用花括号来确定配对关系。例如：

```
if (表达式 1)
    {if (表达式 2)    语句 1}
else
    if (表达式 3)    语句 2
    else            语句 3
```

这时{}说明了第 2 行是一个完整的内嵌 if 语句，不需要别的 else 来匹配了，因此第 3 行的 else 就与第 1 行的 if 相匹配了。

## for 循环语句

C 语言中的 for 语句使用最为灵活，不仅可以用于循环次数已经确定的情况，而且可以用于循环次数不确定而只给出循环结束条件的情况。for 循环的一般形式为：

```
for(表达式 1; 表达式 2; 表达式 3)
    循环语句
```

for 语句的执行过程是这样的：①先求解表达式 1；②再求解表达式 2，如果它的值为真，则执行循环语句，然后执行下面第③步；③如果表达式的值为假，则循环结束，转到第④步；④求

解表达式 3，然后转到第②步；⑤执行 for 语句后面的语句。

for 循环的 3 个表达式各有不同的作用，因此也有不同的形式。最普通的，表达式 1 作为初始化是个赋值语句，表达式 2 作为循环的测试条件一般是关系表达式，表达式 3 作为循环前进的动力是个增量表达式。这 3 个表达式都是可以省略的，但后面的分号“;”不能省。表达式 1 被省略相当于循环变量的初始化被省略了，那么一般来说在 for 语句之前应该给循环变量赋初值。执行的时候，只是跳过“求解表达式 1”这一步，其他都不变。如果作为循环测试条件的表达式 2 省略了，那么循环测试条件将被认为永远是 true，循环也就无休止地进行下去。如果表达式 3 省略了，那么此时程序应该另外设法保证循环能正常结束。

本实例程序先输出提示信息，要求用户输入数列的项数 n，接着通过 if 语句判断所输入的 n 是否大于等于 1（只有不小于 1 数列才有意义），如果小于 1，则程序直接返回退出。实例通过两个 for 循环语句实现数列的求和。

## 程序代码

### 【程序 7】 数列求和

```
/* 计算数列的和*/
#include <stdio.h>
void main()
{
    int i,j,n;                  /* 定义循环变量 i,j, 数列项数 n */
    long int sum=0,temp=0;       /* 定义数列的和及临时变量*/
    printf("Please input a number to n:\n"); /* 提示输入数列项数*/
    scanf("%d",&n);
    if(n<1)                    /* 如果输入的数小于 1*/
    {
        printf("The n must be no less than 1!\n"); /* 提示输入有误*/
        return;                   /* 程序返回，退出*/
    }
    for(i=1;i<=n;i++)          /* 循环计算数列的和*/
    {
        temp=0;
        for(j=1;j<=i;j++)
            temp+=j;
        sum+=temp;
    }
    printf("The sum of the sequence(%d) is %d\n",n,sum); /* 输出数列的和*/
}
```

## 归纳注释

for 语句的表达式 1 可以是设置循环变量初值的赋值表达式，也可以是与循环变量无关的其他表达式。比如“`for( sum=0,i=1;i<=100;i++) sum=sum+1;`”表达式可以是一个简单的表达式，也可以是逗号表达式，即包含一个以上的简单表达式，中间用逗号间隔。这样在 for 语句前面就可以不必为 sum 赋初值了，程序看起来就更紧凑了。

表达式 2 一般是关系表达式（如 `i<=100`）或逻辑表达式（如 `a<b&&x<y`），但也可以是数值表达式或字符表达式，只要它的值为非零，就执行循环体，否则就不执行。这就提供了把简短的循环语句“寄生”在表达式 2 中的方法。比如：`for(i=0; (c=getchar())!='\n';i+=c);` 表达式 2 中有一

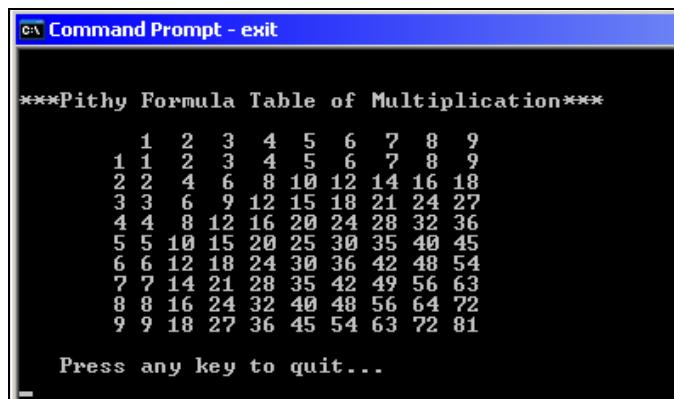
个 `getchar()` 函数，它从终端接收一个字符，然后把该字符赋给 `c`，然后判断接收到的字符是不是换行符。如果不是，那么表达式的值为真，继续循环，由于循环语句是空语句，马上执行 `i+=c`，它的作用是把输入字符的 ASCII 码相加；如果表达式的值为假，也就是接收到一个换行符，那么退出循环。

由于 `main()` 函数可以返回一个自定义的值，返回 `void` 类型时，可用 `return` 直接退出程序。

## 实例 8 乘法口诀表

### 实例说明

输出乘法口诀表，目的是加深对 `for` 循环以及光标位置控制等内容的理解。程序运行结果如图 8-1 所示。



The screenshot shows a Windows Command Prompt window with a blue title bar labeled "Command Prompt - exit". The main area contains the text "\*\*\*\*Pithy Formula Table of Multiplication\*\*\*\*" followed by a 9x9 multiplication table. The table is as follows:

1	2	3	4	5	6	7	8	9
1	2	3	4	5	6	7	8	9
2	4	6	8	10	12	14	16	18
3	6	9	12	15	18	21	24	27
4	8	12	16	20	24	28	32	36
5	10	15	20	25	30	35	40	45
6	12	18	24	30	36	42	48	54
7	14	21	28	35	42	49	56	63
8	16	24	32	40	48	56	64	72
9	18	27	36	45	54	63	72	81

At the bottom of the window, the text "Press any key to quit..." is displayed.

图 8-1 实例 8 程序运行结果

### 实例解析

本程序设计主要思路是：先清屏，并在屏幕上显示提示信息，说明本程序显示的是乘法口诀表。接着，显示横轴 1~9 的数字和纵轴 1~9 的数字。最后，计算  $1 \times 1 \sim 9 \times 9$ ，并在屏幕上显示。

程序算法如下：

```
main()
{
    定义相关变量;
    清屏,显示提示信息;
    循环显示横轴数字;
    循环显示纵轴数字;
    循环计算并显示  $1 \times 1 \sim 9 \times 9$ ;
    提示按任意键退出程序;
    读取任意键;
    程序结束;
}
```



## 程序代码

### 【程序 8】 乘法口诀表

```
#include <stdio.h>
#include <conio.h>
void main(void)
{
    int i,j,x,y;
    clrscr(); /* 清屏 */
    printf("\n\n***Pithy Formula Table of Multiplication***\n\n");
        /*显示提示信息*/
    x=9;
    y=5;
    /* 输出横轴数字 */
    for(i=1;i<=9;i++)
    {
        gotoxy(x,y); /* 移到指定的光标位置 */
        printf("%2d ",i); /* 打印横轴数字 */
        x+=3;
    }
    x=7;
    y=6;
    /* 输出纵轴数字 */
    for(i=1;i<=9;i++)
    {
        gotoxy(x,y); /* 移到指定的光标位置 */
        printf("%2d ",i); /* 打印纵轴数字 */
        y++;
    }
    x=9;
    y= 6;
    /* 计算并显示 1×1~9×9 */
    for(i=1;i<=9;i++)
    {
        for(j=1;j<=9;j++)
        {
            gotoxy(x,y); /* 移到指定的光标位置 */
            printf("%2d ",i*j); /* 打印乘法结果 */
            y++;
        }
        y-=9;
        x+=3;
    }
    printf("\n\n Press any key to quit...\n");
    getch();
}
```



## 归纳注释

clrscr 函数是清屏函数 (Clear Screen)，用于清除屏幕上显示的所有信息，并把光标放在第一行第一列。该函数的功能相当于在 DOS 操作系统下，或者在 Windows 操作系统下的命令行模式

中的 clr 命令。

gotoxy(int x, int y)函数用于将光标移动到指定的(x,y)坐标位置。该函数通常用于控制在屏幕上指定位置的内容显示。

## 实例 9 猜数字游戏

### 实例说明

实现一个简单的猜数字游戏，学习 while 循环语句的用法。程序运行结果如图 9-1 所示。

```
c:\ Command Prompt - exit
====This is a Number Guess Game! ====
Please input Password:
12
Please input Password:
1234
Please input a number between 1 and 100:
12
Your input number is 12
Too Small! Press any key to try again!
Please input a number between 1 and 100:
56
Your input number is 56
Sorry, Only a little smaller! Press any key to try again!
Please input a number between 1 and 100:
59
Your input number is 59
Sorry, Only a little bigger! Press any key to try again!
Please input a number between 1 and 100:
58
Your input number is 58
OK! You are right! Bye Bye!
```

图 9-1 实例 9 程序运行结果

### 实例解析

#### while 循环语句

while 语句的一般形式为：

while(表达式)语句；

其中表达式是循环条件，语句为循环体。

while 语句的语义是：计算表达式的值，当值为真（非 0）时，执行循环体语句。使用 while 语句应注意以下几点。

- (1) while 语句中的表达式一般是关系表达或逻辑表达式，只要表达式的值为真（非 0）即可继续循环。
- (2) 循环体如包含有一个以上的语句，则必须用{}括起来，组成复合语句。
- (3) 应注意循环条件的选择以避免死循环。
- (4) 允许 while 语句的循环体中包含 while 语句，从而形成双重循环。

#### do-while 语句

do-while 语句的一般形式为：

```
do 语句;  
while(表达式);
```

其中语句是循环体，表达式是循环条件。

**do-while** 语句的语义是：先执行循环体语句一次，再判别表达式的值，若为真（非 0）则继续循环，否则终止循环。

**do-while** 语句和 **while** 语句的区别在于 **do-while** 是先执行后判断，因此 **do-while** 至少要执行一次循环体。而 **while** 是先判断后执行，如果条件不满足，则一次循环体语句也不执行。**while** 语句和 **do-while** 语句一般可以相互改写。

对于 **do-while** 语句还应注意以下几点。

(1) 在 **if** 语句和 **while** 语句中，表达式后面都不能加分号，而在 **do-while** 语句的表达式后面则必须加分号。

(2) **do-while** 语句也可以组成多重循环，而且也可以和 **while** 语句相互嵌套。

(3) 在 **do** 和 **while** 之间的循环体由多个语句组成时，也必须用{}括起来组成一个复合语句。

(4) **do-while** 和 **while** 语句相互替换时，要注意修改循环控制条件。

**do-while** 语句也可与 **while**，**for** 语句相互嵌套，构成多重循环。以下 3 种都是合法的嵌套。

```
(1)  
for()  
{  
    ...  
    while()  
    {...}  
    ...  
}  
(2)  
do{  
    ...  
    for()  
    {...}  
    ...  
}while();  
(3)  
while()  
{  
    ...  
    for()  
    {...}  
    ...  
}
```

本程序主要思路是：先使用 **while** 循环语句控制输入密码的过程，如果 3 次输入错误，则给出提示信息并退出程序。密码通过后，使用 **while** 语句控制程序流程，如果输入的数值不等于程序给定的值，则程序一直循环运行下去，直到猜中给定的值。在这层 **while** 内部又用 **do-while** 语句控制输入值的范围，如果输入值不在 1 和 100 之间，就要求重新输入；然后通过 **if...else** 语句判断输入值的范围，并给出相应的提示信息，直到猜中给定值，程序结束。

## 程序代码

### 【程序 9】 猜数字游戏

```
/* 猜数字游戏 */
```

```

#include <stdio.h>
#include <conio.h>
void main()
{
    int Password=0,Number=0,price=58,i=0;
    clrscr(); /* 清屏 */
    printf("\n====This is a Number Guess Game!====\n"); /* 提示信息 */
    while( Password != 1234 ) /* 当输入密码错误时 */
    {
        if( i >= 3 ) /* 如果输入错误次数大于 3 就退出 */
        {
            printf("\n Please input the right password!\n ");
            return;
        }
        i++;
        puts("Please input Password: ");
        scanf("%d",&Password); /* 要求重新输入密码 */
    }

    i=0;
    while( Number!=price )
    {
        do{
            puts("Please input a number between 1 and 100: "); /* 提示猜数 */
            scanf("%d",&Number);
            printf("Your input number is %d\n",Number);
        }while( !(Number>=1 && Number<=100) ); /* 判断范围是否正确 */
        if( Number >= 90 )/* 输入大于 90 的情况 */
        {
            printf("Too Bigger! Press any key to try again!\n");
        }
        else if( Number >= 70 && Number < 90 ) /* 比较大的情况 */
        {
            printf("Bigger!\n");
        }
        else if( Number >= 1 && Number <= 30 ) /* 太小的情况 */
        {
            printf("Too Small! Press any key to try again!\n");
        }
        else if( Number > 30 && Number <= 50 ) /* 比较小的情况 */
        {
            printf("Small! Press any key to try again!\n");
        }
        else
        {
            if( Number == price )
            {
                printf("OK! You are right! Bye Bye!\n");
            }
            else if( Number < price ) /* 相差不多的情况 */
            {
                printf("Sorry, Only a little smaller! Press any key to try again!\n");
            }
        }
    }
}

```

```

        else if( Number > price )
            printf(" Sorry, Only a little bigger! Press any key to try again!\n");
    }
    getch();
}

```



## 归纳注释

常见的循环应用有①计数循环。②输入验证循环：为了避免操作员按键错误，程序包含输入验证循环是非常必须的，比如本例中的密码验证。③哨兵循环：循环程序不停地读、检查和处理数据，直到遇到事先指定的表示结束的值，循环才终止。此值即为“哨兵值”，它控制着循环结束。④延时循环：循环中不实现任何功能，只是使CPU等待一定的时间后，再继续执行程序，即实现计时器的功能。⑤查找循环：按给定的对象进行查找。⑥无限循环：有时程序需要永不停息地执行下去，例如危险信号的监测中经常用到。

# 实例 10 模拟 ATM（自动柜员机）界面



## 实例说明

通过对自动柜员机（ATM）界面的模拟来学习 switch 语句的用法。程序运行结果如图 10-1~图 10-7 所示。

```

Command Prompt - exit
=====
| Please select key:
| 1. Quary
| 2. Credit
| 3. Debit
| 4. Return
=====

```

图 10-1 ATM 界面模拟程序运行主界面

```

Command Prompt - exit
=====
| Your balance is $1000.
| Press any key to return...
=====

```

图 10-2 在主界面中选【1. Quary】后的查询结果界面

```

Command Prompt - exit
=====
| Please select Credit money:
| 1. $50
| 2. $100
| 3. Return
=====

```

图 10-3 在主界面中选【2. Credit】后的存款界面

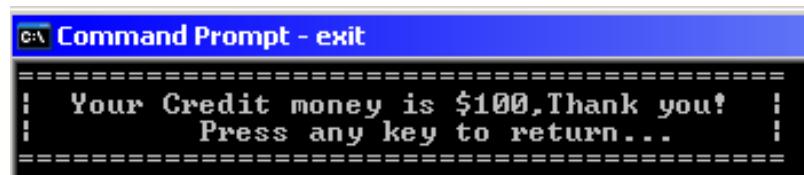


图 10-4 在存款界面中选【2. \$100】后的提示

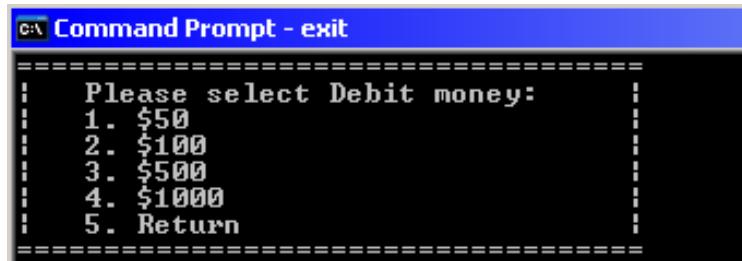


图 10-5 在主界面中选【3. Debit】后的取款界面

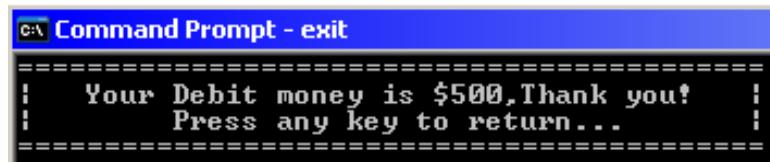


图 10-6 在取款界面中选【3. \$500】后的提示



图 10-7 在主界面中选择【4. Return】后的提示

## 实例解析

### switch 语句

switch 的中文意思是开关，因此 switch 语句又叫开关语句。switch 语句是专门用来处理多分支的，例如，学生成绩分类（90 分以上为‘A’等；80~90 分为‘B’等；70~79 分为‘C’等）；人口统计分类（按年龄分为老、中、青、少、儿童）；工资统计分类；银行存款分类等。当然这些都可以用嵌套的 if 语句，也就是第 3 种形式的 if 语句来处理，但如果分支较多，就显得嵌套的层数太多了，程序冗长且可读性降低。switch 语句就是专门为了解决多分支的问题而设计的。

switch 语句的一般形式如下：

```
switch(变量或表达式)
{
    case 常量表达式 1:
        语句 1
    case 常量表达式 2:
        语句 2
    .
    .
    case 常量表达式 n:
        语句 n
    default:
```

```
    语句 n+1  
}
```

其中, default 是可有可无的。

switch 语句的执行过程是这样的: 计算表达式的值, 判断变量或表达式的值是否等于常量表达式 1, 如果相等, 则执行语句 1, 如果不等, 则判断是否等于常量表达式 2, 如果相等, 则执行语句 2, ……依此类推, 直到结束。如果没有相等的则执行 default 后面的语句 n+1 (如果有的话)。在语句 1~n 执行的过程中, 若遇到 break, 则直接跳出 switch, 不再对下面的 case 语句进行判断。

本程序就是利用 switch 语句对各种选择进行判断和相应处理。本实例是自动柜员机功能菜单的线性模拟程序。如果把程序中的各 case 语句的 puts 语句换成实际的 ATM 中的各功能子程序(函数), 并在主程序外加上各个子程序(函数), 即可成为一个完整的 ATM 程序。

## 程序代码

### 【程序 10】 模拟 ATM 界面

// 本实例源码参见光盘

## 归纳注释

本实例程序在每个 do-while 语句中都控制了输入的有效数值, 保证 switch 语句判断时, 变量所包含的值不超出所有 case 的情况, 因此没有用 default 语句。

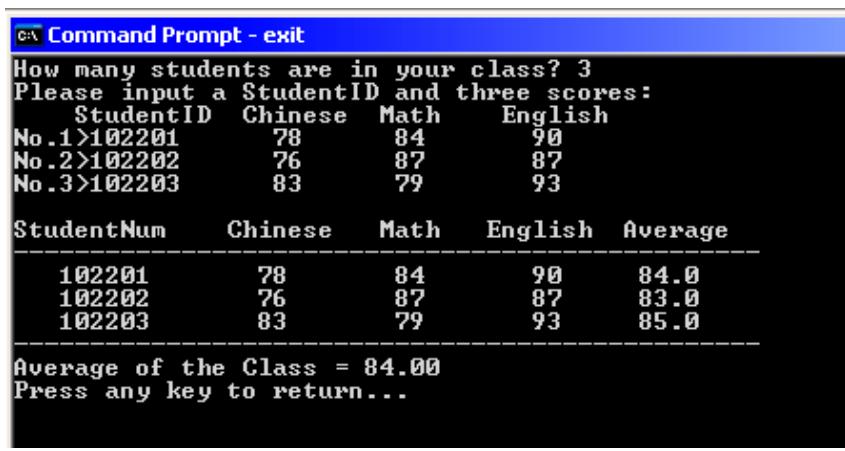
本程序中大量用到 puts 语句, 用于将一个字符串输出在屏幕上。也可以用 printf 语句实现。

本程序中的界面采用“=”、“|”来做成方框的形式, 以增加美观的效果。也可以采用“\*”、“\$”等其他符号来实现。

## 实例 11 用一维数组统计学生成绩

## 实例说明

记录并统计一个班级学生的成绩来学习一维数组的使用。运行本实例程序, 可以对用户输入的学生成绩进行统计, 并输出统计结果。程序运行结果如图 11-1 所示。



The screenshot shows the output of a C program running in a Command Prompt window titled "Command Prompt - exit". The program asks for the number of students in the class (3), then prompts for StudentID and three scores for each student. It then displays a table of student information and calculates the average score for the class.

StudentNum	Chinese	Math	English	Average
102201	78	84	90	84.0
102202	76	87	87	83.0
102203	83	79	93	85.0

Average of the Class = 84.00  
Press any key to return...

图 11-1 实例 11 程序运行结果



## 实例解析

### 一维数组的定义

一维数组的定义方式为：

类型说明符      数组名[常量表达式];

例如：

```
int a[10];
float b[15];
char c[20];
```

其中，a[10]表示数组名为a，此数组有10个元素。

在定义一维数组的同时，需要注意以下几点。

(1) 数组名命名规则和变量名相同，遵循标识符命名规则。

(2) 数组名后是用方括弧括起来的常量表达式，不能使用圆括弧，下面的用法是不正确的：

```
int a(10);
```

(3) 常量表达式表示元素的个数，即数组长度。例如，a[10]中的10表示a数组中有10个元素，下标从0开始，这10个元素分别是：a[0], a[1], a[2], a[3], a[4], a[5], a[6], a[7], a[8], a[9]。

(4) 常量表达式中可以包括常量和符号常量，不能包括变量。也就是说，不允许对数组的大小进行动态定义，即数组的大小不依赖于程序运行过程中变量的值。例如，下面的定义方法就是不可以的：

```
int num;
scanf("%d", &num);
int a[num];
.....
```

### 一维数组的初始化

可以用赋值语句或输入语句给数组中的元素赋值，但是这样做占用不少运行时间。可以使数组在程序运行之前初始化，即在编译阶段得到初值。

对数组元素的初始化可以用以下方法实现。

(1) 在定义数组时对数组元素赋值。例如：

```
static int a[10]={0,1,2,3,4,5,6,7,8,9};
```

将数组元素的初始值依次放在一对大括弧中。

经过上面的定义和初始化之后，a[0]=0, a[1]=1, a[2]=2, a[3]=3, a[4]=4, a[5]=5, a[6]=6, a[7]=7, a[8]=8, a[9]=9。

(2) 也可以只给一部分元素赋值。例如：

```
static int a[10]={0,1,2,3,4};
```

定义a数组有10个元素，但是大括弧内只提供了5个初始值，这表示只给前面5个元素赋初值，后5个元素的值默认都为0。

(3) 如果要使数组中全部元素值为0，可以写成：

```
static int a[10]={0,0,0,0,0,0,0,0,0,0};
```

但不能写成：

```
static int a[10]={0*10};
```

这是与FORTRAN语言不同的。

(4) 在对全部数组元素赋初值时，可以不指定数组长度。例如：

```
static int a[5]={1,2,3,4,5};
```

可以写成：

```
static int a[]={1,2,3,4,5};
```

在第2种写法中，大括弧中有5个数，系统会据此自动定义a数组的长度为5。但若被定义的数组长度与提供初值的个数不相同，则数组长度不能省略。例如，若定义数组长度为10，就不能省略数组长度的定义，而必须写为：

```
static int a[10]={1,2,3,4,5};
```

只初始化前5个元素，后5个元素为0。

本实例利用一维数组存储班级学生的学号，语文、数学、外语等各科成绩，并求出各学生的平均分。程序首先要求输入班级人数（最大为50，可以在程序中修改该最大值），然后要求输入每个学生的学号及各门成绩，并存入相应的一维数组中，最后计算每个学生的平均成绩及全班平均成绩，并输出结果。

## 程序代码

### 【程序11】 用一维数组统计学生成绩

//本实例源码参见光盘

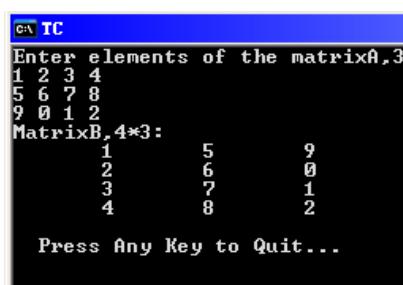
## 归纳注释

本实例中，一维数组的操作基本依靠循环语句来实现，用得最多的是for循环。例如，可以用for循环对一维数组初始化，利用for循环对一维数组的值进行运算等。在循环的基础上，可以进行查找、插入、删除等操作。

# 实例 12 用二维数组实现矩阵转置

## 实例说明

本实例将输入的 $3\times 4$ 矩阵转置为 $4\times 3$ 矩阵，并输出结果。通过本实例，可以学习如何使用二维数组。程序运行结果如图12-1所示。



```
C:\ TC
Enter elements of the matrixA,3
1 2 3 4
5 6 7 8
9 0 1 2
MatrixB,4*3:
      1      5      9
      2      6      0
      3      7      1
      4      8      2
Press Any Key to Quit...
```

图12-1 实例12程序运行结果

## 实例解析

### 二维数组的定义

二维数组定义的一般形式为：

类型说明符 数组名[常量表达式][常量表达式]

例如：

```
int a[3][4], b[7][8];
```

定义 a 为  $3 \times 4$  (3 行 4 列) 的数组, b 为  $7 \times 8$  (7 行 8 列) 的数组。

## 二维数组的引用

二维数组的元素也称为双下标变量, 二维数组的元素的表示形式为:

数组名[下标][下标]

例如 a[3][4], 下标可以是整型常量或是整型表达式, 如 a[2\*2-1][3+1]。特别强调不要写成: a[3,4]或者 a[2\*2-1,3+1]的形式。

数组元素可以出现在表达式中, 也可以被赋值, 例如:

```
b[1][2]=a[2][3]/3;
```

在使用数组元素时, 应该注意下标值应在已定义的数组大小范围内。定义 a 为  $3 \times 4$  的数组, 它可用的行下标值最大为 2, 列坐标值最大为 3。用 a[3][4]则超过了数组的定义范围。

下标变量和数组说明在形式中有些相似, 但两者具有完全不同的含义。数组说明的方括号中给出的是某一维的长度, 即可取下标的最大值; 而数组元素中的下标是该元素在数组中的位置标识。前者只能是常量, 后者可以是常量、变量或表达式。

## 二维数组的初始化

可以用下面的方法对二维数组初始化。

(1) 分行给二维数组赋初值。例如:

```
static int a[3][4]={{1,2,3,4},{5,6,7,8},{9,10,11,12}};
```

这种赋初值方法比较直观, 把第一个大括弧内的数据赋给第一行的元素, 第二个大括弧内的数据赋给第二行的元素……。

(2) 可以将所有数据写在一个大括弧中, 按数组排列的顺序对各元素赋初值。例如:

```
static int a[3][4]={1,2,3,4,5,6,7,8,9,10,11,12};
```

效果与前种方法相同。但第一种方法比较好, 一行对一行。用第二种方法, 如果数据多, 写出来一大片, 就比较容易遗漏, 有错误也不容易检查出来。

(3) 可以对部分元素赋初值, 例如:

```
static int a[3][4]={{1},{3},{5}};
```

它的作用是只对各行第一列的元素赋初值, 其余元素值自动设为 0。赋初值后数组各元素为:

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 3 & 0 & 0 & 0 \\ 5 & 0 & 0 & 0 \end{pmatrix}$$

也可以对各行中的某一元素赋初值:

```
static int a[3][4]={{1},{0,3},{0,0,5}};
```

初始化后的元素如下:

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 \\ 0 & 0 & 5 & 0 \end{pmatrix}$$

这种方法对非 0 元素少时比较方便, 不必将所有的 0 都写出来, 只需输入少量数据。

也可以只对某几行元素赋初值, 例如:

```
static int a[3][4]={{1},{3,5}};
```

数组元素为:

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 3 & 5 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

以上对第 3 行不赋初值。也可以对第 2 行不赋初值，例如：

```
static int a[3][4]={{1},{},{}},
```

(4) 如果对全部元素都赋初值（即提供全部初始数据），则定义数组时对第一维的长度可以不指定，但第二维的长度不能省。如：

```
static int a[3][4]={1,2,3,4,5,6,7,8,9,10,11,12};
```

与下面的定义等价：

```
static int a[][4]={1,2,3,4,5,6,7,8,9,10,11,12};
```

系统会根据数据总个数分配存储空间，一共 12 个数据，每行 4 列，当然可确定为 3 行。

在定义时也可以只对部分元素赋初值而省略第一维的长度，但应当分行赋初值。如：

```
static int a[][4]={{0,0,3},{},{}},
```

这样的写法，能通知编译系统数组共有 3 行。数组各元素为：

$$\begin{pmatrix} 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 10 & 0 & 0 \end{pmatrix}$$

本实例先输出提示信息，要求输入矩阵 A 的值，接着通过两个 for 嵌套来访问二维数组 A，并将其转置存入数组 B，最后输出矩阵 B 的值。



## 程序代码

### 【程序 12】 用二维数组实现矩阵转置

```
/* 用二维数组实现矩阵的转置 */
#include <stdio.h>
#define ROW 3           /* 矩阵的行数 */
#define COL 4           /* 矩阵的列数 */
main()
{
    int matrixA[ROW][COL],matrixB[COL][ROW];/* 矩阵的定义 */
    int i,j;

    clrscr();      /* 清屏 */
    printf("Enter elements of the matrixA,"); /* 提示信息 */
    printf("%d*%d:\n",ROW,COL);
    for( i=0; i<ROW; i++ )
    {
        for( j=0; j<COL; j++ )
        {
            scanf("%d",&matrixA[i][j]);          /* 输入矩阵 A 的值 */
        }
    }

    for( i=0; i<ROW; i++ )
    {
        for( j=0; j<COL; j++ )
        {
            matrixB[j][i] = matrixA[i][j];    /* 转置 */
        }
    }
}
```

```

}

printf("MatrixB,"); /* 输出矩阵 B */
printf("%d*%d:\n",COL,ROW);
for( i=0; i<COL; i++ )
{
    for( j=0; j<ROW; j++ )
    {
        printf("%8d",matrixB[i][j]);
    }
    printf("\n");
}
printf("\n Press Any Key to Quit... \n");
getch();
}

```



### 归纳注释

本程序实现对  $3 \times 4$  矩阵的转置，通过改变行列的数目，可以实现更大规模的矩阵的转置，并且进一步实现更复杂的运算，比如矩阵的加减乘运算，方阵的求逆等。

## 实例 13 求解二维数组的最大/最小元素



### 实例说明

在  $n$  行  $n$  列的二维整数数组中，按以下要求选出两个数。首先从每行选出最大数，再从选出的  $n$  个大数中选出最小数；其次，从每行选出最小数，再从选出的  $n$  个小数中选出最大数。程序运行结果如图 13-1 所示。

```

C:\TC
Please input the order of the matrix:
5
Please input the elements of the matrix,
from a[0][0] to a[4][4]:
1 2 3 4 5
6 7 8 9 10
12 324 545 76 7
34 56 7 89 9
12 3 5 6 7
The minimum of maximum number is 1
The maximum of minimum numbers is 7
Press any key to quit...

```

图 13-1 实例 13 程序运行结果



### 实例解析

设二维数组为  $a[ ][ ]$ ，首先在  $n$  行  $n$  列二维整数数组中，从每行选出最大数，再从选出的  $n$  个大数中选出最小数，可以用以下算法实现：

[算法 1] 在二维整数数组中，从每行选出最大数，再从选出的大数中选出最小数

```
/* 设最小元变量用 min 标记，各行的最大元变量用 max 标记 */
for(min=a[0][0],row=0;row<n;row++)

```

```

{
    /*从每行选出最大数*/
    for(max=a[row][0],col=1;col<n;col++)
        if(max<a[row][col])
            max=a[row][col];
        if(min>max)/*保存至 row 行的最小数*/
            min=max;
    }
    printf("The minimum of maximum numbers is %d\n",min);
}

```

从每行选出最小数，再从选出的 n 个小数中选出最大数，有如下的算法：

[算法 2]在二维整数数组中，从每行选出最小数，再从选出的小数中选出最大数

```

/*设最大元变量用 max 标记，各行的最小元变量用 min 标记*/
for(max=a[0][0],row=0;row<n;row++)
{
    /*从每行选出最大数*/
    for(min=a[row][0],col=1;col<n;col++) /*从 row 行选出最小数*/
        if(min>a[row][col])
            min=a[row][col];
        if(max<min)/*保存至 row 行的最大数*/
            max=min;
    }
    printf("The maximum of minimum numbers is %d\n",max);
}

```



## 程序代码

### 【程序 13】 求解二维数组的最大/最小元素

```

/* 计算二维数组的最大最小值 */
#define MAXN 20
int a[MAXN][MAXN];
main()
{
    int min,      /* 存储最小值 */
        max; /* 存储最大值 */
    int row,col,n;
    clrscr();
    printf("Please input the order of the matrix:\n"); /* 输入方阵的阶次 */
    scanf("%d",&n);
    printf("Please input the elements of the matrix,\n from a[0][0] to
a[%d][%d]:\n",n-1,n-1);
    for(row=0;row<n;row++)
        for(col=0;col<n;col++)
            scanf("%d",&a[row][col]);
    for(min=a[0][0],row=0;row<n;row++)
    {
        /* 从每行选出最大数 */
        for(max=a[row][0],col=1;col<n;col++)/*从 row 行选出最大数 */
            if(max<a[row][col])
                max=a[row][col];
        if(min>max)/* 保存至 row 行的最小数 */
            min=max;
    }
}

```

```

}
printf("The minimum of maximum number is %d\n",min);
for(max=a[0][0],row=0;row<n;row++)
{
    /* 每行选出最小数 */
    for(min=a[row][0],col=1;col<n;col++)/* 从 row 行选出最小数 */
        if(min>a[row][col])
            min=a[row][col];
    if(max<min)/*保存至 row 行的最大数 */
        max=min;
}
printf("The maximum of minimum numbers is %d\n",max);
printf("\nPress any key to quit...\n");
getch();
}

```



## 归纳注释

本例中，实现数组元素输入的语句也可以改为 `scanf("%d",a[row]+col)`。

# 实例 14 利用数组求前 $n$ 个质数



## 实例说明

要求确定一个数  $m$  是否是质数，可以用已求出的质数对  $m$  的整除性来确定。程序运行结果如图 14-1 所示。

```

C:\ Command Prompt - exit
The first 50 prime numbers are:
 2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71
 73 79 83 89 97 101 103 107 109 113 127 131 137 139 149 151 157 163 167 173
179 181 191 193 197 199 211 223 227 229
Press any key to quit...

```

图 14-1 实例 14 程序运行结果



## 实例解析

对任意整数  $m$ ，如果它不能被小于它的质数整除，则  $m$  也是质数。引入质数表 `primes[]`，已求得的质数个数为 `pc`。求前  $n$  个质数的过程可用以下算法描述：

[算法]求前  $n$  个质数

```

{primes[0]=2; /* 2 是第一个质数 */
 pc=1; /* 已有一个质数 */
 m=3; /* 被测试的数从 3 开始 */
 while(pc<N)
 {
    调整 m, 使 m 为下一个质数;
    primes[pc++]=m;
    m+=2; /* 除 2 外, 其余质数均是奇数 */
}

```

```

    }
    输出 primes[0]至 primes[pc-1];
}

```

为调整  $m$  使它是下一个质数，可按顺序用已求得的质数  $\text{primes}[k]$  去测试它对  $m$  的整除性。如果  $m$  能被某个  $\text{primes}[k]$  整除，则  $m$  是合数，让  $m$  增 2，并重新从第一个质数开始对它测试。数  $m$  为质数的条件是存在一个  $k$ ，使得  $\text{primes}[0]$  至  $\text{primes}[k-1]$  不能整除  $m$ ，且“ $\text{primes}[k]*\text{primes}[k]>m$ ”成立。代码描述如下：

```

k=0;
while(primes[k]*primes[k]<=m)
if(m%prime[k]==0)
{
    /*m 是合数*/
    m+=2; /*让 m 取下一个奇数*/
    k=1; /*不必用 primes[0]=2 去测试 m*/
}
else
    k++; /*继续用下一个质数去测试*/

```



## 程序代码

### 【程序 14】 利用数组求前 $n$ 个质数

```

#define N 50
main()
{
    int primes[N];
    int pc,m,k;

    clrscr();
    printf("\n The first %d prime numbers are:\n",N);
    primes[0]=2; /*2 是第一个质数*/
    pc=1; /*已有第一个质数*/
    m =3; /*被测试的数从 3 开始*/
    while(pc<N)
    {
        /*调整 m 使它为下一个质数*/
        k=0;
        while(primes[k]*primes[k]<=m)
            if(m%primes[k]==0)
                { /*m 是合数*/
                    m+=2; /*让 m 取下一个奇数*/
                    k=1; /*不必用 primes[0]=2 去测试 m，所以 k 从 1 开始*/
                }
            else
                k++; /*继续用下一个质数去测试*/
        primes[pc++]=m;
        m+=2; /*除 2 外，其余质数均是奇数*/
    }
    /*输出 primes[0]至 primes[pc-1]*/
    for(k=0;k<pc;k++)
        printf("%4d",primes[k]);
    printf("\n\n Press any key to quit...\n ");
    getch();
}

```

}



## 归纳注释

本例应用数组 primes[] 来存储已经找到的前  $n-1$  个质数，再通过对第  $n-1$  个质数加 2，并测试是否能被前  $n-1$  个质数整除来寻找第  $n$  个质数，从而可以求出前  $n$  个质数。这里  $n$  可以足够大（取决于 int 型数据的取值范围，可以改为 long 型数组，以便求取更多的质数）。

本例子中求取前  $n$  个质数， $n$  用宏定义来实现，也可以通过用户的输入来进行。比如定义一个最大的数以便定义数组“#define MAX 1000”，数组定义改为 int primes[MAX]， $n$  则由用户输入。

# 实例 15 编制万年历



## 实例说明

编制输入年份，则输出该年年历的程序，通过程序学习和掌握三维数组的用法。程序运行结果如图 15-1 所示。

```
ct "M:\Debug\Text1.exe"
Please input the year whose calendar you want to know: 2004
=====
| 1 SUN MON TUE WED THU FRI SAT | 7 SUN MON TUE WED THU FRI SAT |
|       1   2   3           4   5   6   7   8   9   10 |       1   2   3           4   5   6   7   8   9   10 |
| 4   5   6   7   8   9   10           11  12  13  14  15  16  17 |       11  12  13  14  15  16  17 |
| 11  12  13  14  15  16  17           18  19  20  21  22  23  24 |       18  19  20  21  22  23  24 |
| 18  19  20  21  22  23  24           25  26  27  28  29  30  31 |       25  26  27  28  29  30  31 |
| 25  26  27  28  29  30  31           1   2   3           4   5   6   7 |       1   2   3           4   5   6   7 |
| 2   SUN MON TUE WED THU FRI SAT | 8 SUN MON TUE WED THU FRI SAT |
|       1   2   3   4   5   6   7           1   2   3   4   5   6   7 |       1   2   3   4   5   6   7 |
| 8   9   10  11  12  13  14           8   9   10  11  12  13  14 |       8   9   10  11  12  13  14 |
| 15  16  17  18  19  20  21           15  16  17  18  19  20  21 |       15  16  17  18  19  20  21 |
| 22  23  24  25  26  27  28           22  23  24  25  26  27  28 |       22  23  24  25  26  27  28 |
| 29                           29  30  31           29  30  31           29  30  31 |
| 3   SUN MON TUE WED THU FRI SAT | 9 SUN MON TUE WED THU FRI SAT |
|       1   2   3   4   5   6           1   2   3           4   5   6 |       1   2   3           4   5   6 |
| 7   8   9   10  11  12  13           5   6   7   8   9   10  11 |       7   8   9   10  11  12  13 |
| 14  15  16  17  18  19  20           12  13  14  15  16  17  18 |       14  15  16  17  18  19  20 |
| 21  22  23  24  25  26  27           19  20  21  22  23  24  25 |       21  22  23  24  25  26  27 |
| 28  29  30  31                           26  27  28  29  30           26  27  28  29  30 |
| 4   SUN MON TUE WED THU FRI SAT | 10 SUN MON TUE WED THU FRI SAT |
|       1   2   3           1           1   2           3 |       1   2           3 |
| 4   5   6   7   8   9   10           3   4   5   6   7   8   9 |       4   5   6   7   8   9 |
| 11  12  13  14  15  16  17           10  11  12  13  14  15  16 |       11  12  13  14  15  16 |
| 18  19  20  21  22  23  24           17  18  19  20  21  22  23 |       17  18  19  20  21  22  23 |
| 25  26  27  28  29  30                           24  25  26  27  28  29  30 |       24  25  26  27  28  29  30 |
| 31                           31           31           31           31           31 |
| 5   SUN MON TUE WED THU FRI SAT | 11 SUN MON TUE WED THU FRI SAT |
|       1           1           1   2   3           4   5   6 |       1           1           1   2   3           4   5   6 |
| 2   3   4   5   6   7   8           7   8   9   10  11  12  13 |       2   3   4   5   6   7   8   9   10  11  12  13 |
| 9   10  11  12  13  14  15           14  15  16  17  18  19  20 |       9   10  11  12  13  14  15  16  17  18  19  20 |
| 16  17  18  19  20  21  22           21  22  23  24  25  26  27 |       16  17  18  19  20  21  22  23  24  25  26  27 |
| 23  24  25  26  27  28  29           28  29  30           28  29  30           28  29  30           28  29  30 |
| 30  31                           30           31           31           31           31           31 |
| 6   SUN MON TUE WED THU FRI SAT | 12 SUN MON TUE WED THU FRI SAT |
|       1           1           1   2   3           4   5   6 |       1           1           1   2   3           4   5   6 |
| 6   7   8   9   10  11  12           5   6   7   8   9   10  11 |       6   7   8   9   10  11  12 |
| 13  14  15  16  17  18  19           12  13  14  15  16  17  18 |       13  14  15  16  17  18  19 |
| 20  21  22  23  24  25  26           19  20  21  22  23  24  25 |       20  21  22  23  24  25  26 |
| 27  28  29  30                           26  27  28  29  30           26  27  28  29  30           26  27  28  29  30 |
|=====
Press any key to quit...
```

图 15-1 实例 15 程序运行结果



## 实例解析

利用自定义的函数 f() 和 g() 求出该年一月一日的星期，由闰年的判定条件，确定该年是否是闰年。为了便于按星期输出各月的月历，引入三维数组的日期表 int date[12][6][7]。其中，12 表示 12 个月；6 表示一个月最多有 6 个星期；7 表示每星期 7 天。程序首先将日期表置 0，然后顺序将各月的日期填写到日期表中，各月的天数按月取自数组 day\_tb1[][]。该数组是二维数组，第一行存储平年的各月的天数，第二行存储闰年的各月的天数。程序的输出格式分两栏，左面一栏是 1~6 月的月历，右面一栏是 7~12 月的月历。输出时，若日期数为 0，就用 4 个空白符代替，否则输出日期。



## 程序代码

### 【程序 15】 编制万年历

```
/* 输入年份，输出该年的年历 */
#include "stdio.h"
long int f(int year,int month)
{ /*f(年, 月)=年-1, 如月<3;否则, f(年, 月)=年*/
    if(month<3) return year-1;
    else return year;
}
long int g(int month)
{ /*g(月)=月+13, 如月<3;否则, g(月)=月+1*/
    if(month<3) return month+13;
    else return month+1;
}

long int n(int year,int month,int day)
{
    /*N=1461*f(年、月)/4+153*g(月)/5+日*/
    return 1461L*f(year,month)/4+153L*g(month)/5+day;
}

int w(int year,int month,int day)
{
    /*w=(N-621049)%7(0<=w<7)*/
    return(int)((n(year,month,day)%7-621049L%7+7)%7);
}

int date[12][6][7];
int day_tb1[ ][12]={ {31,28,31,30,31,30,31,31,30,31,30,31},
                     {31,29,31,30,31,30,31,31,30,31,30,31}};

main()
{
    int sw,leap,i,j,k,wd,day;
    int year; /*年*/
    char title[]="SUN MON TUE WED THU FRI SAT";

    printf("Please input the year whose calendar you want to know: "); /*输入年*/
    scanf("%d%c",&year); /*输入年份值及回车*/
}
```

```

sw=w(year,1,1);
leap=year%4==0&&year%100||year%400==0/*判断闰年*/
for(i=0;i<12;i++)
    for(j=0;j<6;j++)
        for(k=0;k<7;k++)
            date[i][j][k]=0/*日期表置0*/
for(i=0;i<12;i++)/*一年12个月*/
    for(wd=0,day=1;day<=day_tbl[leap][i];day++)
    {/*将第i+1月的日期填入日期表*/
        date[i][wd][sw]=day;
        sw=++sw%7/*每星期7天,以0~6计数*/
        if(sw==0) wd++;/*日期表每7天一行,星期天开始新的一行*/
    }

printf("\n|=====
The Calendar of Year %d =====|\n",year);
for(i=0;i<6;i++)
{/*先测算第i+1月和第i+7月的最大星期数*/
    for(wd=0,k=0;k<7;k++)/*日期表的第6行有日期,则wd!=0*/
        wd+=date[i][5][k]+date[i+6][5][k];
    wd=wd%6;
    printf("%2d %s %2d %s |\n",i+1,title,i+7,title);
    for(j=0;j<wd;j++)
    {
        printf("   ");/*输出3个空白符*/
        /*左栏为第i+1月,右栏为第i+7月*/
        for(k=0;k<7;k++)
            if(date[i][j][k])
                printf("%4d",date[i][j][k]);
            else printf("   ");
            printf("   ");/*输出6个空白符*/
        for(k=0;k<7;k++)
            if(date[i+6][j][k])
                printf("%4d",date[i+6][j][k]);
            else printf("   ");
        printf(" |\n");
    }
}
puts("=====|");
puts("\n Press any key to quit... ");
getch();
}

```



## 归纳注释

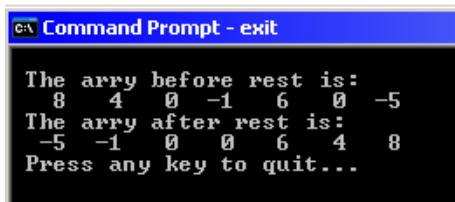
本程序的运行结果是在 Visual C++ 6.0 中实现的。若在 TC2.0 中,由于命令行窗口的高度限制,无法观察到程序运行结果的全貌。

在程序中,也可以将 12 个月的日历分 6 次输出,此时在输出完两个月的日历后,应添加如下语句 `scanf("%*c");/*键入回车输出下一个月的日历*/`,以便按回车继续输出后面的日历。

# 实例 16 对数组元素排序

## 实例说明

本实例对一个一维整型数组进行排序，使其元素的顺序按从小到大排列，即 0 在中间，负数在前面，正数在后面。通过此例子，可以学习数组的各种操作。程序运行结果如图 16-1 所示。



```
The arry before rest is:  
8 4 0 -1 6 0 -5  
The arry after rest is:  
-5 -1 0 0 6 4 8  
Press any key to quit...
```

图 16-1 实例 16 程序运行结果

## 实例解析

函数 `rest(int a[], int n)` 可以实现对具有  $n$  个元素的数表  $a$  的排序。其实现思路为，对  $a[]$  的元素依次访问，如果  $a[i]$  大于 0，则将  $a[i]$  与最后一个元素交换，同时，最后一个元素指向最后第二个；如果  $a[i]$  为 0，则访问下一个元素；如果  $a[i]$  小于 0，则与第一个元素交换，并将第一个元素指向第二个元素。

[算法] 对数组元素排序

```
for 循环依次访问 a[ ]所有元素，直到 i<=high
{
    if(a[i]>0)
    {
        a[i]与 a[high]交换;
        high--;
    }
    else if(a[i]==0)
        略过该元素;
    else
    {
        a[i]与 a[low]交换，然后 low 增 1， i 增 1;
    }
}
```

## 程序代码

### 【程序 16】 对数组元素排序

```
rest(int a[], int n)
{
    int i,low,high,t;

    for(i=0,low=0,high=n-1;i<=high;)
    {
```

```

if(a[i]>0)
{
    /*a[i]与 a[high]交换，随之 high 减 1*/
    t=a[i];
    a[i]=a[high];
    a[high]=t;
    high--;
}
else if(a[i]==0)
    i++; /*略过该元素*/
else
{
    /*a[i]与 a[low]交换，随之 low 增 1, i 增 1*/
    t=a[i];
    a[i]=a[low];
    a[low]=t;
    low++;
    i++;
}
}

int s[]={8,4,0,-1,6,0,-5};
main() /*主函数用于测试 rest 函数*/
{
    int i;
    clrscr(); /*清屏*/
    printf("\n The arry before rest is:\n");
    for(i=0;i<sizeof(s)/sizeof(s[0]);i++)
        printf("%4d",s[i]);
    rest(s,sizeof(s)/sizeof(s[0]));
    printf("\n The arry after rest is:\n");
    for(i=0;i<sizeof(s)/sizeof(s[0]);i++)
        printf("%4d",s[i]);
    printf("\n Press any key to quit...\n");
    getch();
}

```



## 归纳注释

有关数列排序、查找等操作和算法在数据结构篇中将会详细介绍。

本实例主函数中的 s 数组也可以由用户输入，此时应当先提示输入数组中元素个数，再通过 for 循环依次读入键盘输入的元素。

# 实例 17 任意进制数的转换



## 实例说明

将一个无符号整数转换为任意 d 进制数 ( $2 \leq d \leq 16$ )。程序运行结果如图 17-1 所示。

```
TC
Please input a number to translate:
253
The number you input is 253.
The translation results are:
 253 = 1111101<2>
 253 = 100101<3>
 253 = 3331<4>
 253 = 2003<5>
 253 = 1101<6>
 253 = 511<7>
 253 = 375<8>
 253 = 311<9>
 253 = 253<10>
 253 = 210<11>
 253 = 191<12>
 253 = 166<13>
 253 = 141<14>
 253 = 11D<15>
 253 = FD<16>
253 => (1) Error!
Press any key to quit...
```

图 17-1 实例 17 程序运行结果

## 实例解析

求  $n$  整除  $d$  的余数，就能得到  $n$  的  $d$  进制数的最低位数字，重复上述步骤，直至  $n$  为 0，依次得到  $n$  的  $d$  进制数表示的最低位至最高位数字。由各位数字取出相应字符，就能得到  $n$  的  $d$  进制的字符串。

主函数用于测试 `trans` 函数，要求输入一个待转换的正整数，接着调用 `trans` 函数进行转换，并输出该正整数的 2~16 进制数转换结果。

## 程序代码

### 【程序 17】 无符号整数的任意进制数转换

```
/* 函数 trans 将无符号整数 n 转换成 d (2<=d<=16) 进制表示的字符串 s */
#define M sizeof(unsigned int)*8
int trans(unsigned n, int d, char s[])
{
    static char digits[] = "0123456789ABCDEF"; /* 十六进制数字的字符 */
    char buf[M+1];
    int j, i = M;
    if(d<2||d>16)
    {
        s[0]='\0'; /* 不合理的进制，置 s 为空字符串 */
        return 0; /* 不合理的进制，函数返回 0 */
    }
    buf[i]='\0';
    do
    {
        buf[--i]=digits[n%d]; /* 得出最低位，将对应字符存入对应数组中 */
        n/=d;
    }while(n);
    /* 将工作数组中的字符串复制到 s */
    for(j=0;(s[j]=buf[i])!='\0';j++,i++);
}
```

```

    /* 其中控制条件可简写成 s[j]=buf[i] */
    return j;
}
/* 主函数用于测试函数 trans() */
main()
{
    unsigned int num = 0;
    int scale[] = {2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,1};
    char str[33];
    int i;
    clrscr();
    puts("Please input a number to translate:");
    scanf("%d", &num);
    printf("The number you input is %d.\nThe translation results are:\n", num);
    for(i=0;i<sizeof(scale)/sizeof(scale[0]);i++)
    {
        if(trans(num,scale[i],str))
            printf("%5d = %s(%d)\n", num,str,scale[i]);
        else
            printf("%5d => (%d) Error! \n", num,scale[i]);
    }
    printf("\n Press any key to quit...\n");
    getch();
}

```



## 归纳注释

本实例将 0~F 等 16 个字符存储在静态字符数组里，通过该静态字符数组的引用来实现任意进制数的转换。

# 实例 18 判断回文数



## 实例说明

判定正整数 n 的 d 进制表示形式是否是回文数。程序运行结果如图 18-1 所示。

```

TC
232 -> <2> is not a Circle Number!
232 -> <10> is a Circle Number!
232 -> <16> is not a Circle Number!
27 -> <2> is a Circle Number!
27 -> <10> is not a Circle Number!
27 -> <16> is not a Circle Number!
851 -> <2> is not a Circle Number!
851 -> <10> is not a Circle Number!
851 -> <16> is a Circle Number!

Press any key to quit...
-
```

图 18-1 实例 18 程序运行结果



## 实例解析

回文数就是顺着看和倒着看相同的数。例如， $n=232$ ，它的十进制表示是回文数； $n=27$ ，它的二进制表示 $11011_{(2)}$ 是回文数； $n=851$ ，它的 $16$ 进制表示 $353_{(16)}$ 是回文数。

设判断是否为回文数的函数为 circle(int n, int d)，它有两个参数 n 和 d。其中 n 为要判定是否为回文数的整数，d 指将 n 转成 d 进制表示。为判定 n 是否是 d 进制表示形式中的回文数，有两种办法：一是顺序译出 n 的 d 进制表示的各位数字，然后首末对应位数字两两比较，若对应位都相同，则 n 是 d 进制表示形式中的回文数，否则 n 不是回文数；二是首先顺序译出 n 的 d 进制的各位数字，然后把译出的各位数字将低位当高位按 d 进制的转换方法译成一个整数，若 n 是回文数，则转换所得整数应与 n 相等；否则，n 不是回文数。下面的函数是参照方法二编写的，参照方法一编写相应函数的工作留给读者完成。



## 程序代码

### 【程序 18】 判断回文数

```
/* 函数 circle 用于判断正整数 n 的 d 进制数表示形式是否是回文数 */
int circle(int n, int d)
{
    int s=0,m=n;
    while(m)
    {
        s=s*d+m%d;
        m/=d;
    }
    return s==n;
}
/* main 函数用于测试 circle 函数 */
int num[]={232,27,851};
int scale[]={2,10,16};
main()
{
    int i,j;
    clrscr();
    for(i=0;i<sizeof(num)/sizeof(num[0]);i++)
        for(j=0;j<sizeof(scale)/sizeof(scale[0]);j++)
            if(circle(num[i],scale[j]))
                printf("%d -> (%d) is a Circle Number!\n",num[i],scale[j]);
            else
                printf("%d -> (%d) is not a Circle Number!\n",num[i],scale[j]);
    printf("\n Press any key to quit...\n");
    getch();
}
```



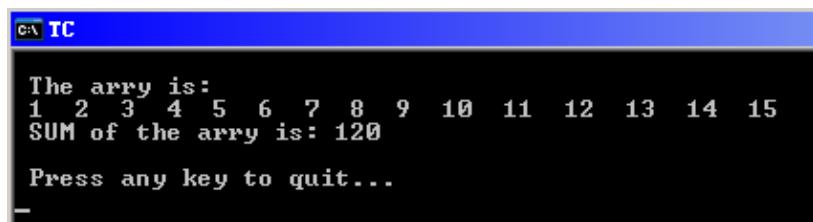
## 归纳注释

本实例中主函数中的待判断的数，也可以由用户从键盘输入，并要求输入相应的所要判断的 d 进制数。

# 实例 19 求数组前 $n$ 元素之和

## 实例说明

编制递归函数来求取数组的前  $n$  个元素的和，从而求取该数组的所有元素的和。程序运行结果如图 19-1 所示。



```
C:\TC
The arry is:
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
SUM of the arry is: 120
Press any key to quit...
```

图 19-1 实例 19 程序运行结果

## 实例解析

编制一个递归函数  $\text{sum}(\text{int } \text{a}[ ], \text{int } n)$ ，求已知数组  $\text{a}[ ]$  的前  $n$  个元素的和。数组  $\text{a}[ ]$  的前  $n$  个元素的和为  $\text{sum} = \text{a}[n-1] + \dots + \text{a}[0]$ 。

数组  $\text{a}[ ]$  的前  $n$  个元素之和是  $\text{a}[n-1]$  加上数组  $\text{a}[ ]$  的前  $n-1$  个元素的和，即当  $n > 0$  时， $\text{sum}(\text{a}, n) = \text{a}[n-1] + \text{sum}(\text{a}, n-1)$ ，当  $n=0$  时， $\text{sum}(\text{a}, 0)=0$ 。

## 程序代码

### 【程序 19】 求数组前 $n$ 个元素之和

```
int a[]={1,2,3,4,5,6,7,8,9,10,11,12,13,14,15};
main()
{
    int i;
    clrscr();
    printf("\n The arry is:\n");
    for(i=0;i<sizeof(a)/sizeof(a[0]);i++)
        printf(" %d ",a[i]);
    printf("\n SUM of the arry is: %d\n",sum(a,sizeof(a)/sizeof(a[0])));
    printf("\n Press any key to quit...\n");
    getch();
}
sum(int a[],int n)
{
    if(n<=0)
        return 0;
    return a[n-1]+sum(a,n-1);
}
```



## 归纳注释

递归求解是一种比较常用而且非常有效的求解方法。比如实例 7 也可以用递归的方法求解。此时，可以编写函数 long sumofseq(int n) 来求取数列的前 n 项的和。该函数的实现代码为 long sumofseq(int n) { if(n==1) return 1; else return sumofseq(n-1)+n\*(n-1)/2;}

在编写递归求解的函数时，要注意递归求解的边界条件。比如本例中，当  $n \leq 0$  时就返回 0。在其他一些情况下，还可能需要判断  $n=1$  和  $n=2$  两种边界条件，因为递归调用使用的起始条件可能是  $n=1$  和  $n=2$  时的值。

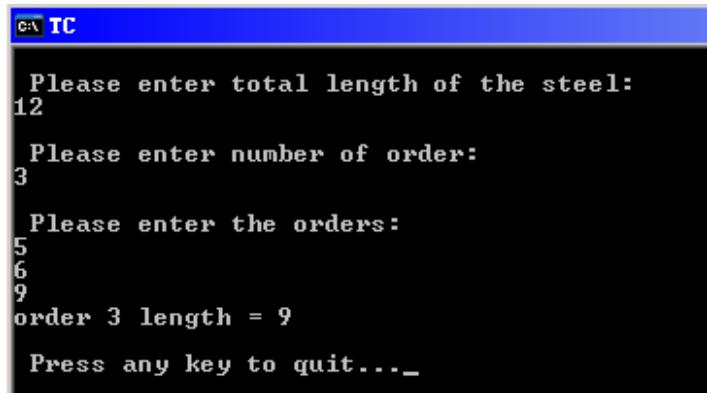
# 实例 20 求解钢材切割的最佳订单



## 实例说明

已知线型钢材总长、定单数和各定单需要的钢材长度，编制程序从定单中选择一组定单对钢材作切割加工，使钢材得到最佳利用。约定，每次切割会损耗固定长度的钢材。

本实例求取线性钢材切割的最佳订单。程序运行结果如图 20-1 所示。



```
C:\> TC
Please enter total length of the steel:
12
Please enter number of order:
3
Please enter the orders:
5
6
9
order 3 length = 9
Press any key to quit....
```

图 20-1 实例 20 程序运行结果



## 实例解析

采用递归方法求解本问题，主函数的工作是输入线型钢材总长、定单数和各定单所需要的钢材长，然后调用求解函数，获得一组定单编号，最后输出被选中的定单编号及所选定的钢材长。

为采用递归求解本问题，递归函数以选中的定单信息为基础。为表示选中的定单，引入数组，若第  $i$  张定单被选中，则其下标为  $i$  的元素值为 1；否则，其值为 0。递归函数被调用后，首先计算当前被选中一组定单的用料数量，若这组选择将超过钢材总长，则不再被继续考虑；若一组选择是可取的，则当这组选择是更好的选择时，应将该选择保护，作为到目前为止的最佳解。然后对还未被选中的所有定单循环并作递归调用，尝试这样的选择是否会得到更佳的解。在递归调用返回后，准备对下一个还未选的定单作选择尝试之前，应重新恢复前次尝试中的定单未被选中状态，否则变成顺序地连续被选中的处理方式，这不是算法所要求的。



## 程序代码

### 【程序 20】 求解钢材切割的最佳订单

```
#include <stdio.h>
#define N 20
#define DELTA 2
int bestlen;
int bestsele[N];
int sele[N];
int n;
int orderlen[N];
int total;
main()
{
    int i;
    clrscr();
    printf("\n Please enter total length of the steel:\n"); /* 输入钢材总长 */
    scanf("%d",&total);
    printf("\n Please enter number of order:\n"); /* 输入订单数 */
    scanf("%d",&n);
    printf("\n Please enter the orders:\n"); /* 输入各订单 */
    for(i=0;i<n;i++)
        scanf("%d",&orderlen[i]);
    bestlen=0; /*最佳解的初值 */
    for(i=0;i<n;i++)
        sele[i]=bestsele[i]=0; /*给当前选择和最佳选择置初值 */
    try(); /* 调用函数求解 */
    for(i=0;i<n;i++) /* 输出结果 */
        if(bestsele[i])
            printf("order %d length = %d\n",i+1,orderlen[i]);
    printf("\n Press any key to quit...");
    getch();
}
try()
{
    int i,len;
    for(len=i=0;i<n;i++) /* 求当前选中的用料量 */
        if(sele[i])
            len+=orderlen[i]+DELTA;
    if(len-DELTA<=total) /* 注意最后一段可能不需要切割 */
    {
        if(bestlen < len)
        {
            /* 找到一个更好的解 */
            bestlen = len;
            for(i=0;i<n;i++)
                bestsele[i]=sele[i];
        }
        for(i=0;i<n;i++) /* 对所有未选订单逐一选中尝试循环 */
            if(!sele[i])
            {
                sele[i]=1; /* 做选中尝试 */
                if(bestlen < len)
                    bestlen = len;
                for(i=0;i<n;i++)
                    bestsele[i]=sele[i];
            }
    }
}
```

```
    try();
    sele[i]=0;
}
}
```



## 归纳注释

本程序使用 sele[ ]数组来保存是否对某一订单做选定的标志，使用 orderlen[ ]数组来保存  $n$  个订单的相应长度。通过递归调用实现对最佳订单的查找。

此外，通过对本程序的改进，还可以实现对最佳切割方案的查找。即通过对最小切割次数的控制来实现对切割方案的寻优。

# 实例 21 通过指针比较整数大小



## 实例说明

本实例通过指针实现整数大小的比较。程序要求定义 3 个整型变量用于存储读入的 3 个整数。另定义 3 个指向整型变量的指针变量，并利用它们实现将 3 个整型变量中的 3 个整数按值从小到大顺序输出。

程序运行结果如图 21-1 所示。

```
TC
Please input x,y,z:
89
56
78
x = 56  y = 78  z = 89
Press any key to quit...
-
```

图 21-1 程序运行结果



## 实例解析

### 指针的概念

指针是 C 语言中广泛使用的一种数据类型。运用指针编程是 C 语言最主要的风格之一。利用指针变量可以表示各种数据结构；能很方便地使用数组和字符串；并能像汇编语言一样处理内存地址，从而编出精练而高效的程序。

在计算机中，所有的数据都是存放在存储器中的。一般把存储器中的一个字节称为一个内存单元，不同的数据类型所占用的内存单元数不等，如整型量占 2 个单元，字符量占 1 个单元等。为了正确地访问这些内存单元，必须为每个内存单元编上号，这样根据一个内存单元的编号即可准确地找到该内存单元。内存单元的编号也叫做地址。因为根据内存单元的编号或地址就可以找到所需的内存单元，所以通常也把这个地址称为指针。

内存单元的指针和内存单元的内容是两个不同的概念。可以用一个通俗的例子来说明它们之间的关系。我们到银行去存取款时，银行工作人员将根据我们的账号去找存款单，找到之后在存单上写入存款、取款的金额。在这里，账号就是存单的指针，存款数是存单的内容。对于一个内存单元来说，单元的地址即为指针，其中存放的数据才是该单元的内容。

在C语言中，允许用一个变量来存放指针，这种变量称为指针变量。因此，一个指针变量的值就是某个内存单元的地址或称为某内存单元的指针。设有字符变量 C，其内容为“K”（ASCII 码为十进制数 75），C 占用了 011A 号单元（地址用十六进制数表示）。设有指针变量 P，内容为 011A，这种情况称为 P 指向变量 C，或说 P 是指向变量 C 的指针。严格地说，一个指针是一个地址，是一个常量。而一个指针变量却可以被赋予不同的指针值，是变量。但通常把指针变量简称为指针。

既然指针变量的值是一个地址，那么这个地址不仅可以是变量的地址，也可以是其他数据结构的地址。在一个指针变量中存放一个数组或一个函数的首地址有何意义呢？因为数组或函数都是连续存放的，所以通过访问指针变量便可以取得数组或函数的首地址，也就找到了该数组或函数。这样以来，凡是出现数组、函数的地方都可以用一个指针变量来表示，只要给该指针变量中赋予数组或函数的首地址即可。在C语言中，一种数据类型或数据结构往往都占有一组连续的内存单元。用“地址”这个概念并不能很好地描述一种数据类型或数据结构，而“指针”虽然实际上也是一个地址，但它却是一个数据结构的首地址，它是“指向”一个数据结构的，因而概念更为清楚，表示更为明确。

### 指针变量的类型说明

对指针变量的类型说明包括 3 个内容。

- (1) 指针类型说明，即定义变量为一个指针变量。
- (2) 指针变量名。
- (3) 变量值（指针）所指向的变量的数据类型。

其一般形式为：

类型说明符 \* 变量名；

其中，\* 表示这是一个指针变量，变量名即为定义的指针变量名，类型说明符表示本指针变量所指向的变量的数据类型。

例如“int \*p1;”表示 p1 是一个指针变量，它的值是某个整型变量的地址。或者说 p1 指向一个整型变量。至于 p1 究竟指向哪一个整型变量，应由向 p1 赋予的地址来决定。

再如：

```
staic int *p2; /*p2 是指向静态整型变量的指针变量*/  
float *p3; /*p3 是指向浮点变量的指针变量*/  
char *p4; /*p4 是指向字符变量的指针变量*/
```

应该注意的是，一个指针变量只能指向同类型的变量，如 p3 只能指向浮点变量，不能时而指向一个浮点变量，时而又指向一个字符变量。

### 指针变量的赋值

指针变量同普通变量一样，使用之前不仅要定义说明，而且必须赋予具体的值。未经赋值的指针变量不能使用，否则将造成系统混乱，甚至死机。指针变量的赋值只能赋予地址值，决不能赋予任何其他数据类型，否则将引起错误。在C语言中，变量的地址是由编译系统分配的，对用户完全透明，用户不知道变量的具体地址。C语言中提供了地址运算符“&”来表示变量的地址。其一般形式为：

& 变量名；

如 &a 表示变量 a 的地址，&b 表示变量 b 的地址。变量本身必须预先声明。假设有指向整型

变量的指针变量 p，要把整型变量 a 的地址赋予 p 可以有以下两种方式。

(1) 指针变量初始化的方法

```
int a;  
int *p=&a;
```

(2) 赋值语句的方法

```
int a;  
int *p;  
p=&a;
```

不允许把一个数赋予指针变量，故下面的赋值是错误的：

```
int *p;  
p=1000;
```

被赋值的指针变量前不能再加“\*”说明符，如写为“\*p=&a”也是错误的（注意不同于初始化的方法）。

## 指针变量的运算

指针变量只能进行赋值运算和部分算术运算及关系运算。

### 1. 指针运算符

#### (1) 取地址运算符&

取地址运算符&是单目运算符，其结合性为自右至左，其功能是取变量的地址。在 scanf 函数及前面介绍指针变量赋值中，已经使用了&运算符。

#### (2) 取内容运算符\*

取内容运算符\*是单目运算符，其结合性为自右至左，用来表示指针变量所指的变量。在\*运算符之后跟的变量必须是指针变量。需要注意的是，指针运算符\*和指针变量说明中的指针说明符\*不同。在指针变量说明中，“\*”是类型说明符，表示其后的变量是指针类型。而表达式中出现的“\*”则是一个运算符，用以表示指针变量所指的变量，例如：

```
main()  
{  
    int a=5,*p=&a;  
    printf ("%d", *p);  
}
```

表示指针变量 p 取得了整型变量 a 的地址。本语句表示输出变量 a 的值。

### 2. 指针变量的运算

#### (1) 赋值运算

指针变量的赋值运算有以下几种形式。

① 指针变量初始化赋值，前面已介绍。

② 把一个变量的地址赋予指向相同数据类型的指针变量。例如：

```
int a,*pa;  
pa=&a; /*把整型变量 a 的地址赋予整型指针变量 pa*/
```

③ 把一个指针变量的值赋予指向相同类型变量的另一个指针变量。例如：

```
int a,*pa=&a,*pb;  
pb=pa; /*把 a 的地址赋予指针变量 pb*/
```

由于 pa,pb 均为指向整型变量的指针变量，因此可以相互赋值。

④ 把数组的首地址赋予指向数组的指针变量。

例如：

```
int a[5],*pa;  
pa=a; 错误！链接无效。数组名表示数组的首地址，故可赋予指向数组的指针变量 pa */
```

也可写为：

```
pa=&a[0]; /*数组第一个元素的地址也是整个数组的首地址，也可赋予 pa*/
```

当然也可采取初始化赋值的方法：

```
int a[5], *pa=a;
```

⑤ 把字符串的首地址赋予指向字符类型的指针变量。

例如 “char \*pc;pc="c language";” 或用初始化赋值的方法写为 “char \*pc="C Language";” 这里并不是把整个字符串装入指针变量，而是把存放该字符串的字符数组的首地址装入指针变量。

⑥ 把函数的入口地址赋予指向函数的指针变量。例如：

```
int (*pf)(); pf=f; /*f 为函数名*/
```

(2) 加减算术运算

对于指向数组的指针变量，可以加上或减去一个整数 n。设 pa 是指向数组 a 的指针变量，则 pa+n,pa-n,pa++,++pa,pa--,--pa 运算都是合法的。指针变量加或减一个整数 n 的意义是把指针指向的当前位置(指向某数组元素)向前或向后移动 n 个位置。

应该注意，数组指针变量向前或向后移动一个位置和地址加 1 或减 1 在概念上是不同的。因为数组可以有不同的类型，各种类型的数组元素所占的字节长度是不同的。如指针变量加 1，即向后移动 1 个位置表示指针变量指向下一个数据元素的首地址，而不是在原地址基础上加 1。

例如：

```
int a[5], *pa;  
pa=a; /*pa 指向数组 a，也是指向 a[0]*/  
pa=pa+2; /*pa 指向 a[2]，即 pa 的值为&a[2]*/
```

指针变量的加减运算只能对数组指针变量进行，对指向其他类型变量的指针变量做加减运算是毫无意义的。两个指针变量之间的运算只有指向同一数组的两个指针变量之间才能进行运算，否则运算毫无意义。

① 两指针变量相减

两指针变量相减所得之差是两个指针所指数组元素之间相差的元素个数。实际上是两个指针值(地址)相减之差再除以该数组元素的长度(字节数)。例如 pf1 和 pf2 是指向同一浮点数组的两个指针变量，设 pf1 的值为 2010H，pf2 的值为 2000H，而浮点数组每个元素占 4 个字节，所以 pf1-pf2 的结果为(2000H-2010H)/4=4，表示 pf1 和 pf2 之间相差 4 个元素。两个指针变量不能进行加法运算。

② 两指针变量进行关系运算

指向同一数组的两指针变量进行关系运算可表示它们所指数组元素之间的关系。



## 程序代码

### 【程序 21】 变量所占的字节数

```
main()  
{  
    int x,y,z; /* 定义 3 个 int 型变量 */  
    int *xp = &x, /* 定义指针变量 xp，并赋值为 x 的地址，使 xp 指向 x */  
        *yp = &y, /* 定义指针变量 yp，并赋值为 y 的地址，使 yp 指向 y */  
        *zp = &z; /* 定义指针变量 zp，并赋值为 z 的地址，使 zp 指向 z */  
    int t;  
    clrscr();  
    printf("\nPlease input x,y,z:\n");  
    scanf("%d%d%d",xp,yp,zp); /* 通过变量的指针，为变量输入值 */  
    if(*xp>*yp) /* 通过指向变量的指针引用变量的值 */  
    {
```

```

t=*xp; /* 通过指向变量的指针引用变量的值 */
*xp=*yp; /* 通过指向变量 x 的指针 xp, 引用变量 x 的值 */
*yp=t; /* 通过指向变量 y 的指针 yp, 引用变量 y 的值 */
}
if(*xp>*zp) /* 通过指向变量的指针, 引用变量的值 */
{
    t=*xp; /* 通过指向变量 x 的指针 xp, 引用变量 x 的值 */
    *xp=*zp; /* 通过指向变量 x 的指针 xp, 引用变量 x 的值 */
    *zp=t; /* 通过指向变量 z 的指针 zp, 引用变量 z 的值 */
}
if(*yp>*zp) /* 通过指向变量的指针, 引用变量的值 */
{
    t=*yp; /* 通过指向变量的指针, 引用变量的值 */
    *yp=*zp; /* 通过指向变量 y 的指针 yp, 引用变量 y 的值 */
    *zp=t; /* 通过指向变量 z 的指针 zp, 引用变量 z 的值 */
}
printf("x = %d\ty = %d\tz = %d\n",x,y,z);
printf("\nPress any key to quit...\n");
getch();
}

```



## 归纳注释

本实例说明如何用指针实现数的大小比较。设 3 个变量分别为 x, y, z, 与它们对应的 3 个指针变量为 xp,yp,zp。两变量 x 和 y 之间的比较“x>y”,利用指向它们的指针 xp 和 yp 可以写成“\*xp>\*yp”。与指针有关的类型说明如下所示（以整型为例）。

```

int i;          /* 定义整型变量 i */
int *p;         /* p 为指向整型数据的指针变量 */
int a[n];       /* 定义整型数组 a, 它有 n 个元素 */
int *p[n];     /* 定义指针数组 p, 它由 n 个指向整型数据的指针元素组成 */
int (*p)[n];   /* p 为指向含 n 个元素的一维数组的指针变量 */
int f();        /* f 为带回整型函数值的函数 */
int *p();       /* p 为带回一个指针的函数, 该指针指向整型数据 */
int (*p)();    /* p 为指向函数的指针, 该函数返回一个整型值 */
int **p;        /* p 是一个指针变量, 它指向一个指向整型数据的指针变量 */

```

# 实例 22 指向数组的指针



## 实例说明

本实例实现通过指向数组的指针引用数组、利用数组名及下标引用数组等。  
程序运行结果如图 22-1 所示。

```
ca TC
a[0] = 1    a[1] = 2    a[2] = 3    a[3] = 4    a[4] = 5
*p = 1    *p = 2    *p = 3    *p = 4    *p = 5
*(p+0) = 1    *(p+1) = 2    *(p+2) = 3    *(p+3) = 4    *(p+4) = 5
p[-4] = 1    p[-3] = 2    p[-2] = 3    p[-1] = 4    p[-0] = 5
Press any key to quit...
```

图 22-1 程序运行结果



## 实例解析

程序可预定义一个初始化的数组，然后用数组名和下标遍历数组。程序另定义一个指针变量，让指针顺序指向数组的各元素，实现遍历数组；指针与游标变量结合，让指针指向数组中的某元素，不改动指针，顺序改变游标变量也能遍历数组。

程序先输出提示说明信息，接着依次输出各个类型变量所占的字节数。



## 程序代码

### 【程序 22】 指向数组的指针

```
int a[ ]={1,2,3,4,5};
#define N sizeof a/sizeof a[0]
main()
{
    int j, /*游标变量*/
        *p; /*指针变量*/
    clrscr();
    for(j=0;j<N;j++)/*用数组名和下标顺序访问数组元素*/
        printf("a[%d]\t= %d\t",j,a[j]);
    printf("\n");
    for(p=a;p<a+N;p++)/*让指针顺序指向数组的各元素，遍历数组*/
    printf("*p\t= %d\t",*p);
    printf("\n");/*指针与游标变量结合，改变游标变量遍历数组*/
    for(p=a,j=0;p+j<a+N;j++)
        printf("*(p+%d)\t= %d\t",j,* (p+j));
    printf("\n");/*指针与游标变量结合，用指针和下标遍历数组*/
    for(p=a+N-1,j=N-1;j>=0;j--)
        printf("p[-%d]\t= %d\t",j,p[-j]);
    printf("\n");
    Press any key to quit...\n";
    getch();
}
```



## 归纳注释

指向数组的指针变量称为数组指针变量。一个数组是由连续的一块内存单元组成的。数组名就是这块连续内存单元的首地址。一个数组也是由各个数组元素（下标变量）组成的。每个数组元素按其类型不同占有不同连续的内存单元。一个数组元素的首地址是指它所占有的几个内存单元的首地址。一个指针变量既可以指向一个数组，也可以指向一个数组元素，可把数组名或第一

个元素的地址赋予它。

若要使指针变量指向第 i 号元素，可以把 i 元素的首地址赋予它，或把数组名加 i 赋予它。设有实数组 a，指向 a 的指针变量为 pa，则有以下关系：pa、a、&a[0]均指向同一单元，它们是数组 a 的首地址，也是 0 号元素 a[0]的首地址。pa+1、a+1、&a[1]均指向 1 号元素 a[1]。类推可知 a+i、a+i、&a[i]指向 i 号元素 a[i]。应该说明的是 pa 是变量，而 a,&a[i]都是常量。

数组指针变量说明的一般形式为：类型说明符 \* 指针变量名。其中类型说明符表示所指数组的类型。从一般形式可以看出指向数组的指针变量和指向普通变量的指针变量的说明是相同的。引入指针变量后，就可以用两种方法来访问数组元素了。第一种方法为下标法，即用 a[i]形式访问数组元素。第二种方法为指针法，即采用\*(pa+i)形式，用间接访问的方法来访问数组元素。

## 实例 23 寻找指定元素的指针

### 实例说明

在已知数表中找出第一个与指定值相等的元素的下标和指针。

程序运行结果如图 23-1 所示。

```
C:\ TC
The elements of array a is:
90 80 70 60 50 40 30 20 10 9 8 7 6 5 42 40 50 1 2 3
The address of a[0] is: 404.

Please input the key number you want to search:
40

The label number of the key number 40 in the array is: 5.
The point value of the key number 40 in the array is: 414.

Press any key to quit...
```

图 23-1 程序运行结果

### 实例解析

设要编制的函数为“int search(int \*apt,int n,int key)”，其中 apt 为给定的数表的首元素的指针；n 为数表中的元素个数；key 为要寻找的元素的值。函数返回的整型值为找到的元素在已知数表中的下标，如数表中没有值为 key 的元素，则函数返回-1 值。

[函数 1]在已知数表中找第一个与指定值相等的元素

```
int search(int *apt,/*已知数表的首元指针*/
int n,/*数表中元素个数*/
int key)/*要寻找的值*/
{
    int *p;
    for(p=apt;p<apt+n;p++)
        if(*p==key)
            return p-apt; /*返回找到元素的下标*/
    return -1;
}
```

设要编制的函数为“int \*find(int \*apt,int n,int key)”，其参数的意义与上述函数 search()的参数的意义相同，但函数 find()返回找到的元素的指针。

[函数 2]在已知数表中找第一个与指定值相等的元素

```
int *find(int *apt,/*已知数表的首元指针*/
          int n,/*数表中元素个数*/
          int key)/*要寻找的值*/
{
    int *p;
    for(p=apt;p<apt+n;p++)
        if(*p==key)
            return p; /*返回找到元素的指针*/
    return NULL;
}
```



## 程序代码

### 【程序 23】 寻找指定元素的指针

```
#include <stdio.h>
int search(int *apt,/*已知数表的首元指针*/
           int n,/*数表中元素个数*/
           int key)/*要寻找的值*/
{
    int *p;
    for(p=apt;p<apt+n;p++)
        if(*p==key)
            return p-apt; /*返回找到元素的下标*/
    return -1;
}
int *find(int *apt,/*已知数表的首元指针*/
          int n,/*数表中元素个数*/
          int key)/*要寻找的值*/
{
    int *p;
    for(p=apt;p<apt+n;p++)
        if(*p==key)
            return p; /*返回找到元素的指针*/
    return NULL;
}

int a[]={90,80,70,60,50,40,30,20,10,9,8,7,6,5,42,40,50,1,2,3};
main()
{
    int i,key;
    clrscr();
    printf("The elements of array a is:\n");
    for(i=0;i<sizeof(a)/sizeof(a[0]);i++)
        printf(" %d",a[i]);
    printf("\nThe address of a[0] is: %d.\n",&a[0]);
    puts("\nPlease input the key number you want to search:");
    scanf("%d",&key);
    i=search(a,sizeof(a)/sizeof(a[0]),key);
    printf("\nThe label number of the key number %d in the array is: %d.",key,i);
```

```
    printf("\n\nThe point value of the key number %d in the array is: %d.",key,find(a,  
sizeof(a)/sizeof(a[0]),key));  
  
    puts("\n\n Press any key to quit...");  
    getch();  
}
```



## 归纳注释

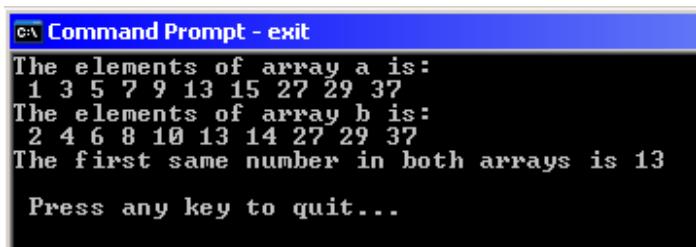
由于整型变量在内存中占用两个字节，所以，a[0]的首地址是 404，而 a[5]的首地址是 404+(5-0)×2=414。

# 实例 24 寻找相同元素的指针



## 实例说明

在已知两个从小到大的有序的数表中寻找出现的相同元素在第一个数表中的指针。程序运行结果如图 24-1 所示。



```
Command Prompt - exit  
The elements of array a is:  
1 3 5 7 9 13 15 27 29 37  
The elements of array b is:  
2 4 6 8 10 13 14 27 29 37  
The first same number in both arrays is 13  
Press any key to quit...
```

图 24-1 实例 24 程序运行结果



## 实例解析

设两个数表的首元素指针分别为 pa, pb, 两个数表分别有元素 an 和 bn 个。另外，引入两个指针变量 ca, cb 分别指向两个数表的当前访问元素。由于两个数表从小到大有序，可让 ca 和 cb 同时顺序访问两数表实现寻找，不必对一个数表的每个元素与另一数表的所有元素都要比较。当一个数表的当前元素小于另一个数表的当前元素时，就调整那个当前元素值小的元素的指针，使它指向下一个元素（如果下一个元素存在）如此比较直至当两个数表的当前元素相等，该元素就是在两个数表中都出现的第一个元素，或者其中某个数表已查找完，则判断在两数表中不存在值相等的元素。



## 程序代码

### 【程序 24】 寻找相同元素的指针

```
/* [函数]在已知两个从小到大有序的数表中寻找都出现的第一个元素的指针 */  
#define NULL 0  
int *search2(int *pa,int *pb,int an,int bn)  
{  
    int *ca,*cb;
```

```

ca=pa;cb=pb; /*为 ca、 cb 设定初值*/
while(ca<pa+an&&cb<pb+bn)/*两个数表都未考察完*/
{
/*在两个数表中找下一个相等的元素*/
if(*ca<*cb)/*数表 1 的当前元素<数表 2 的当前元素*/
    ca++; /*调整数表 1 的当前元素指针*/
else if(*ca>*cb)/*数表 1 的当前元素>数表 2 的当前元素*/
    cb++; /*调整数表 2 的当前元素指针*/
else /*数表 1 的当前元素==数表 2 的当前元素*/
    /*在前两个数表中找到相等元素*/
    return ca; /*返回在这两个数表中找到相等元素*/
}
return NULL;
}

main()/*只是为了引用函数 search2()*/
{
int *vp,i;
int a[ ]={1,3,5,7,9,13,15,27,29,37};
int b[ ]={2,4,6,8,10,13,14,27,29,37};
clrscr();
puts("The elements of array a is:");
for(i=0;i<sizeof(a)/sizeof(a[0]);i++)
    printf(" %d",a[i]);
puts("\nThe elements of array b is:");
for(i=0;i<sizeof(b)/sizeof(b[0]);i++)
    printf(" %d",b[i]);
vp=search2(a,b,sizeof a/sizeof a[0],sizeof b/sizeof b[0]);
if(vp) printf("\nThe first same number in both arrays is %d\n",*vp);
else printf("Not found!\n");
puts("\n Press any key to quit...\n");
getch();
}

```



## 归纳注释

本例子程序在主函数中，通过调用 search2() 函数返回的是第一个相同元素的指针，即该元素在第一个数组中的地址，通过取值\*vp 得到该元素的值，也可以直接将该指针的值输出，则应为该元素在第一个数组中的地址。

# 实例 25 阿拉伯数字转换为罗马数字



## 实例说明

编写一个将整数 n( $1 \leq n \leq 9999$ ) 转化成罗马数字表示的程序。程序运行的结果如图 25-1 所示，将 1~22 的数字都转换为罗马数字，并显示出来。如图 25-2 所示则是输入“25.exe 2000”后的运行结果。

```
C:\> C:\WINDOWS\System32\cmd.exe
C:\>tc> roman 22
1 i
2 ii
3 iii
4 iv
5 v
6 vi
7 vii
8 viii
9 ix
10 x
11 xi
12 xii
13 xiii
14 xiv
15 xv
16 xvii
17 xviii
18 xviiii
19 xix
20 xx
21 xxi
22 xxii
C:\>
```

图 25-1 输入“25.exe 22”的运行结果

```
C:\> C:\WINDOWS\System32\cmd.exe
C:\>tc> roman 2000
1978 mcmlxviii
1979 mcmlxix
1980 mcmlx
1981 mcmlxii
1982 mcmlxiii
1983 mcmlxiiii
1984 mcmlxiv
1985 mcmlxv
1986 mcmlxvi
1987 mcmlxvii
1988 mcmlxviii
1989 mcmlxix
1990 mcmxc
1991 mcmxci
1992 mcmxcii
1993 mcmxciii
1994 mcmxciv
1995 mcmxcv
1996 mcmxcvi
1997 mcmxcvii
1998 mcmxcviii
1999 mcmxcix
2000 mm
C:\>
```

图 25-2 输入“25.exe 2000”的运行结果



## 实例解析

整数  $n(1 \leq n \leq 9999)$  与罗马数字表示有以下对应关系：

1000 用一个字符  $m$  来表示，有几个 1000 就用几个  $m$  来表示；  
900 用两个字符  $cm$  来表示；  
500 用一个字符  $d$  来表示；  
400 用两个字符  $cd$  来表示；  
100 用一个字符  $c$  来表示；有几个 100 就用几个  $c$  来表示；  
90 用两个字符  $xc$  来表示；  
50 用一个字符  $l$  来表示；  
40 用两个字符  $xl$  来表示；  
10 用一个字符  $x$  来表示；有几个 10 就用几个  $x$  来表示；  
9 用两个字符  $iv$  来表示；  
5 用一个字符  $v$  来表示；  
4 用两个字符  $iv$  来表示；  
1 用一个字符  $i$  来表示；有几个 1 就用几个  $i$  来表示。

为了便于程序处理，将阿拉伯数与对应的罗马数字表示分存在两个数组中。转换时，从尽可能大的数开始考察，要转换的罗马字符能被当前考察的数相减后仅大于等于 0 的次数，就是该考察数所对应的罗马数字可连续出现的次数。例如数 23，能连续减 10 两次仅大于等于 0，能连续减 1 三次仅大于等于 0，所以其罗马数字有两个字符  $x$  和 3 个字符  $i$ ，即  $xxiii$ 。

另外，程序将要转换的整数作为程序的参数。如只有一个参数，表示 1 至该数范围内的整数转换成罗马数字，如有两个参数，表示该两数范围内的整数转换成罗马数字。

程序还能检查参数的合理性，并将完成转换的工作和检查的工作分别由两个函数完成。



## 程序代码

### 【程序 25】 阿拉伯数字转换为罗马数字

```
#include <stdio.h>
#define ROWS 4
#define COLS 4
```

```

int nums[ROWS][COLS]={{1000,1000,1000,1000},
                      {900,500,400,100},
                      {90,50,40,10},
                      {9,5,4,1}};
char *roms[ROWS][COLS]={{"m","m","m","m"},
                        {"cm","d","cd","c"},
                        {"xc","l","xl","x"},
                        {"ix","v","iv","i"}};
main(int argc,char *argv[])
{
    int low,high;
    char roman[25];
    if(argc<2)
    {   printf("Usage:roman decimal_number\n");/*运行程序需带整数参数*/
        exit(0);
    }
    high=low=atoi(argv[1]);/*将第一个参数转换成整数*/
    checknum(low);
    if(argc>2)
    {/*带两个参数*/
        high=atoi(argv[2]);
        checknum(high);
        if(low>high)
        {
            low=high;
            high=atoi(argv[1]);
        }
    }
    else
        low=1;
    for(;low<=high;low++)
    {
        to_roman(low,roman);
        printf("%d\t%s\n",low,roman);
    }
}
checknum(int val)/*检查参数合理性*/
{
    if(val<1||val>9999)
    {
        printf("The number must be in range 1..9999.\n");
        exit(0);
    }
}
to_roman(int decimal,char roman[])/*将整数转换成罗马数字表示*/
{
    int power,index;
    roman[0]='\0';
    for(power=0;power<ROWS;power++)
        for(index=0;index<COLS;index++)
            while(decimal>=nums[power][index])
            {
                strcat(roman,roms[power][index]);
                decimal-=nums[power][index];
            }
}

```

```
    }  
}
```



## 归纳注释

程序使用带命令行参数的 main 函数，将 1~n 的所有数都转换为罗马数字并输出。修改某些语句，也可以实现仅转换一个整数的功能。

# 实例 26 字符替换



## 实例说明

编制一个字符替换函数 rep(char \*s,char \*s1,char \*s2)，实现将已知字符串 s 中所有属于字符串 s1 中的字符都用字符串 s2 中的对应字符代替。程序运行结果如图 26-1 所示。

```
TC  
Please input the string for s:  
ABCDEFGHIACDEFHACYCT  
Please input the string for s1:  
AC  
Please input the string for s2:  
XY  
The string of s after displace is:  
XBYDEFXYDXYDEFGHXYYYT  
Press any key to quit...
```

图 26-1 实例 26 程序运行结果



## 实例解析

例如设字符串 s,s1, s2 分别为如下所示。

```
char s[ ]= "ABCABC", s1[ ]= "AC", s2[ ]= "xy";
```

则调用函数 rep(s,s1,s2) 将使字符串 s 的内容变为 "xByxBy"。

为了实现上述要求，可用一个循环顺序访问字符串 s 中的字符，并检查该字符是否在字符串 s1 中出现，若不在字符串 s1 中出现，则略过该字符；若在字符串 s1 中出现，则用字符串 s2 中的对应字符代替 s 中的字符。



## 程序代码

### 【程序 26】 字符替换

```
#include <stdio.h>  
#define MAX 50  
/* 函数 rep 实现对 s 中出现的 s1 中的字符替换为 s2 中相应的字符 */  
rep(char *s,char *s1,char *s2)  
{  
    char *p;  
    for(;*s;s++) /*顺序访问字符串 s 中的每个字符*/
```

```

{
    for(p=s1; *p&&*p!=*s; p++) /*检查当前字符是否在字符串 s1 中出现*/
        if(*p)*s==*(p-s1+s2) /*当前字符在字符串 s1 中出现，用字符串 s2 中的对应字符代替 s 中
的字符*/
    }
main( )/*示意程序*/
{
    char s[MAX]; /*="ABCABC";*/
    char s1[MAX],s2[MAX];
    clrscr();
    puts("Please input the string for s:");
    scanf("%s",s);
    puts("Please input the string for s1:");
    scanf("%s",s1);
    puts("Please input the string for s2:");
    scanf("%s",s2);
    rep(s,s1,s2);
    puts("The string of s after displace is:");
    printf("%s\n",s);
    puts("\n Press any key to quit...");
    getch();
}

```



### 归纳注释

查找替换是许多文字处理软件，比如 word 必备的基本功能，本例程序实现了较为复杂的替换功能，其基本原理是相同的。

## 实例 27 从键盘读入实数



### 实例说明

编制一个从键盘读入实数的函数 `readreal(double *rp)`。函数将读入的实数字符列转换成实数后，利用指针参数 `rp`，将实数存于指针所指向的变量`*rp`。程序运行结果如图 27-1 所示。

```

C:\ TC

Please input real numbers (use nonreal char to end input):
123456.789
The real number you input is: 123456.789000
12345678
The real number you input is: 12345678.000000
1234560
The real number you input is: 123456.000000

You have inputted nonreal char.
Press any key to quit...

```

图 27-1 实例 27 程序运行结果



## 实例解析

函数在返回之前，将最后读入的结束实数字符列的字符返还给系统，以便随后读字符时能再次读入该字符。函数若能正常读入实数，函数返回整数 1；如果函数在读入过程中，未遇数字符之前，遇到不能构成数字的情况，函数返回-1，表示未读到实数。

在输入实数时，在实数之前可以有个数不定的空白类字符，组成实数的字符列有数的符号字符、实数的整数部分、小数点和实数的小数部分，其中某些部分可以缺省。设实数字符列有以下几种可能形式：

数符	整数部分
数符	整数部分.
数符	整数部分.小数部分
数符	.小数部分

其中数符或为空，或为字符‘+’，或为字符‘-’，分别表示不带符号、带正号和带负号。整数部分和小数部分至少要有一个数字符组成。

上述实数形式说明，在实数转换过程中，同一字符在不同情况下会有不同的意义。为标记当前实数转换的不同情况，程序引入状态变量，由状态变量的不同值表示当前实数转换过程中的不同情况。

共有以下多种不同情况：正准备开始转换、转换了数的符号字符、正在转换实数的整数部分、正在转换实数的小数部分、发现输入错误、转换正常结束。设状态变量为 0 表示正准备开始转换，还未遇到任何与实数有关的字符；状态变量为 1 表示已遇数的符号字符；状态变量为 2 表示正在转换实数的整数部分；状态变量为 3 表示在未遇数字字符之前先遇小数点，必须要有小数部分；状态变量为 4 表示在转换整数部分之后遇小数点，这种情况可以没有小数部分；状态变量为 5(Err) 表示转换发现错误；状态变量为 6(OK) 表示转换正常结束。程序将输入字符分成数的符号字符、数字符、小数点、其他字符等几类，各状态遇各类字符后，应变成的新状态，如下所示。

	数符	数字符	小数点	其他字符
0	1	2	3	Err
1	Err	2	3	Err
2	Ok	2	4	Ok
3	Err	4	Err	Err
4	Ok	4	Ok	Ok

下面的程序将处理数的符号、转换整数部分、转换小数部分等工作都编写成函数，它们在读入实数函数的控制下工作。读实数函数另有两张表：一是转换函数表，二是状态表。函数反复读入字符，将字符分类，根据当前状态和当前字符类调用对应转换函数，根据当前状态和当前字符类，从状态表取出新的状态。当新状态为 Err 或 Ok 时，转换结束，前者表示发现错误，后者表示正确转换了一个实数。



## 程序代码

### 【程序 27】 从键盘读入实数

//本实例源码参见光盘



## 归纳注释

本实例中的 `readreal` 函数采用指针作为函数的参数，对变量作实参时，与普通变量一样，也是“值传递”。即将指针变量的值（一个地址）传递给被调用函数的形参（必须是一个指针变量）。被调用函数不能改变实参指针变量的值，但可以改变实参指针变量所指向的变量的值。因此，为了利用被调用函数改变的变量值，应该使用指针（或指针变量）作为函数实参。其机制为在执行被调用函数时，使形参指针变量所指向的变量的值发生变化；函数调用结束后，通过不变的实参指针（或实参指针变量）将变化的值保留下来。

本程序实现的是从键盘读入实数的功能，做相应的改动后，也可以实现从键盘读入整数、字符等功能。

# 实例 28 字符行排版



## 实例说明

将字符行内单字之间的空白符平均分配插入到单字之间，以实现字符行排版。程序运行结果如图 28-1 所示。

```
C:\ TC
This is a typeset program!
Please input a character line:
The quick brown fox           jumps over the      lazy dog !
The character line after typeset is:
The   quick    brown    fox    jumps    over    the    lazy    dog    !
Press any key to quit...
```

图 28-1 实例 28 程序运行结果



## 实例解析

首先要统计字符行内单字个数、字符行内的空白字符数。然后计算出单字之间应平均分配的空白字符数，另约定多余的空白字符插在前面的单字间隔中，前面的每个间隔多一个空白符，插完为止。然后，将字符行复制到一个工作字符数组中，顺序扫视工作字符数组，略过其中的空白符，将连续的空白符复制到字符行，接着复制平均个数的空白符，如余额空白符还未用完，再复制一个空白符。在上述复制过程中，当复制了最后一个单字后，排版结束，函数返回。另外，因字符行单字个数小于 2 不需要排版，函数发现这种情况将立即返回。



## 程序代码

### 【程序 28】 字符行排版

// 本实例源码参见光盘



## 归纳注释

本实例中用到的转换函数表是一个函数指针数组。该数组是二维数组，其元素是指向 sign、integer、decimal 等函数的指针。这些函数的返回值都是 int 型的。

应该特别注意函数指针变量和指针型函数这两者在写法和意义上的区别。如 `int(*p)()` 和 `int *p()` 是两个完全不同的量。`int(*p)()` 是一个变量说明，说明 `p` 是一个指向函数入口的指针变量，该函数的返回值是整型量，`(*p)` 两边的括号不能少。`int *p()` 则不是变量说明而是函数说明，`p` 是一个指针型函数，其返回值是一个指向整型量的指针，`*p` 两边没有括号。作为函数说明，在括号内最好写入形式参数，这样便于与变量说明区别。

# 实例 29 字符排列



## 实例说明

用已知字符串 `s` 中的字符，生成由其中 `n` 个字符组成的所有字符排列。设 `n` 小于字符串 `s` 的字符个数，其中 `s` 中的字符在每个排列中最多出现一次。例如，对于 `s[ ] = "abc", n=2`，则所有字符排列有：`ba,ca,ab,cb,ac,bc`。程序运行结果如图 29-1 所示。

```

C:\ Command Prompt - exit
This is a char permutation program!
Please input a string:
abc

Please input the char number of permuted:
2
The permuted chars are:

ba
ca
ab
cb
ac
bc

Press any key to quit...

```

图 29-1 实例 29 程序运行结果



## 实例解析

可采用递归方法来生成用字符串 `s` 中的字符生成由 `n` 个字符组成的字符排列。设程序用字符数组 `w[ ]` 存储生成的字符排列。令递归函数为 `perm(int n,char *s)`，其功能是用字符串 `s` 中的字符生成由 `n` 个字符组成的所有排列。其方法是从字符串 `s` 依次选用其中的每个字符填写到字符排列的第 `n` 个位置 (`w[n-1]`) 中，然后通过递归调用生成由 `n-1` 个字符组成的排列。当一个字符被选用后，进一步递归调用时可选用的字符中应不再包含该字符。另外，当发现一个字符排列生成后，就不再递归调用，而是输出生成的字符列。综合以上的思路，函数 `perm(int n, char *s)` 的算法如下。

[算法] 函数 `perm(int n, char *s)` 的算法

```

{
    if(一个排列已经生成)printf("%s\n",w);
    else
    {
        保存本层可使用的字符串;
        依次选本层可用字符循环完成以下工作;
        {
            将选用字符填入正在生成的字符排列中;
            形成进一步递归可使用的字符串;
            递归调用;
        }
    }
}

```

## 程序代码

### 【程序 29】 字符排列

```

#define N 20
char w[N];
perm(int n, char *s)
{
    char s1[N];
    int i;
    if(n<1)
        printf("%s\n",w); /* 一个排列生成输出 */
    else
    {
        strcpy(s1,s);/* 保存本层可使用的字符 */
        for(i=0;*(s1+i);i++) /* 依次选本层可用字符 */
        {
            *(w+n-1)=*(s1+i);/* 将选用字符填入正在生成的字符排列中 */
            *(s1+i)=*s1;
            *s1=*(w+n-1);
            perm(n-1,s1+1); /* 递归 */
        }
    }
}
main()
{
    int n=2;
    char s[N];
    w[n]='\0';
    clrscr();
    printf("This is a char permutation program!\nPlease input a string:\n");
    scanf("%s",s);
    puts("\nPlease input the char number of permuted:\n");
    scanf("%d",&n);
    puts("The permuted chars are:\n");
    perm(n,s);
    puts("\nPress any key to quit...");
    getch();
}

```



## 归纳注释

在本例子中，当递归调用生成 0 个字符的排列时，一个字符排列已经生成。为保存本层次可使用的字符串，可引入一个字符数组。为存储进一步递归可使用的字符串，可做如下处理，因进一步递归可使用的字符总比本层次可使用的字符少一个字符，可把本层次正在选用的字符与其字符串中的首字符交换，而进一步递归可使用的字符串就是从次字符开始的字符串。

# 实例 30 判断字符串是否回文



## 实例说明

“回文”是指顺读和反读内容均相同的字符串，例如，“121”、“ABBA”、“X”等。本例将编写函数判断字符串是否是回文。程序运行结果如图 30-1 所示。

```

C:\ TC
Please input the string you want to judge <input ^ to quit>:
aba
    aba is a cycle string.
Please input the string you want to judge <input ^ to quit>:
bccb
    bccb is a cycle string.
Please input the string you want to judge <input ^ to quit>:
123456
    123456 is not a cycle string.
Please input the string you want to judge <input ^ to quit>:
123321
    123321 is a cycle string.
Please input the string you want to judge <input ^ to quit>:
^
Thank you for your using,bye bye!

```

图 30-1 实例 30 程序运行结果



## 实例解析

引入两个指针变量，开始时，两指针分别指向字符串的首末字符，当两指针所指字符相等时，两指针分别向后和向前移一个字符位置，并继续比较，直至两指针相遇，说明该字符串是回文。若比较过程中，发现两字符不相等，则可以判断该字符串不是回文。



## 【程序 30】 判断字符串是否回文

```

#include <stdio.h>
#define MAX 50
int cycle(char *s)
{
    char *h,*t;

    for(h=s,t=s+strlen(s)-1;t>h;h++,t--)

```

```

        if(*h!=*t) break;
    return t<=h;
}

main()
{
    char s[MAX];
    clrscr();
    while(1)
    {
        puts("Please input the string you want to judge (input ^ to quit):");
        scanf("%s",s);
        if(s[0]=='^')
            break;
        if(cycle(s))
            printf(" %s is a cycle string.\n",s);
        else
            printf(" %s is not a cycle string.\n",s);
    }
    puts("\nThank you for your using,bye bye!\n");
}

```



## 归纳注释

主程序采用 while 循环，对每个输入的字符串进行判断，直到输入“^”为止（只要输入的字符串第一个是“^”，不管后面是什么字符都退出程序）。

# 实例 31 通讯录的输入输出



## 实例说明

编制一个包含姓名、地址、邮编和电话的通讯录输入和输出函数。程序运行结果如图 31-1 所示。

```

C:\ Command Prompt - exit

Please input the Name:
John
Please input the Address:
Zhedalu
Please input the Zip code:
310027
Please input the Phone number:
057187952520

Please input the Name:
^Z
Name : John
Address : Zhemalu
Zip : 310027
Phone : 057187952520

Press any key to quit...

```

图 31-1 实例 31 程序运行结果



## 实例解析

在实际问题中，一组数据往往具有不同的数据类型。例如，在学生登记表中，姓名为字符型，学号为整型或字符型，年龄为整型，性别为字符型，成绩为整型或实型。显然不能用一个数组来存放这一组数据。因为数组中各元素的类型和长度都必须一致，以便于编译系统处理。为了解决这个问题，C语言中给出了另一种构造数据类型——结构。它相当于其他高级语言中的记录。

“结构”是一种构造类型，它是由若干“成员”组成的。每一个成员可以是一个基本数据类型或者又是一个构造类型。结构是一种“构造”而成的数据类型，在说明和使用之前必须先定义它，也就是构造它。如同在说明和调用函数之前要先定义函数一样。

### 结构的定义

定义一个结构的一般形式为：

```
struct 结构名  
{  
    成员表列  
};
```

成员表由若干个成员组成，每个成员都是该结构的一个组成部分。对每个成员也必须进行类型说明，其形式为：

类型说明符 成员名；

成员名的命名应符合标识符的书写规定。例如：

```
struct stu  
{  
    int num;  
    char name[20];  
    char sex;  
    float score;  
};
```

在这个结构定义中，结构名为stu，该结构由4个成员组成。第一个成员为num，整型变量；第二个成员为name，字符数组；第三个成员为sex，字符变量；第四个成员为score，实型变量。应注意在括号后的分号是不可少的。结构定义之后，即可进行变量说明。凡说明为结构stu的变量都由上述4个成员组成。由此可见，结构是一种复杂的数据类型，是数目固定，类型不同的若干有序变量的集合。

### 结构类型变量的说明

说明结构变量有以下3种方法，以上面定义的stu为例。

先定义结构，再说明结构变量。例如：

```
struct stu  
{  
    int num;  
    char name[20];  
    char sex;  
    float score;  
};  
struct stu boy1,boy2;
```

以上说明了两个变量boy1和boy2为stu结构类型。也可以用宏定义用一个符号常量来表示

一个结构类型，例如：

```
#define STU struct stu
STU
{
    int num;
    char name[ 20 ];
    char sex;
    float score;
};

STU boy1,boy2;
```

在定义结构类型的同时说明结构变量。例如：

```
struct stu
{
    int num;
    char name[ 20 ];
    char sex;
    float score;
}boy1,boy2;
```

直接说明结构变量。例如：

```
struct
{
    int num;
    char name[ 20 ];
    char sex;
    float score;
}boy1,boy2;
```

第 3 种方法与第 2 种方法的区别在于第 3 种方法中省去了结构名，而直接给出结构变量。3 种方法中说明的 boy1, boy2 变量都具有相同的结构。说明了 boy1, boy2 变量为 stu 类型后，即可向这两个变量中的各个成员赋值。在上述 stu 结构定义中，所有的成员都是基本数据类型或数组类型。成员也可以又是一个结构，即构成了嵌套的结构。

在 ANSI C 中除了允许具有相同类型的结构变量相互赋值以外，一般对结构变量的使用，包括赋值、输入、输出、运算等都是通过结构变量的成员来实现的。

表示结构变量成员的一般形式是“结构变量名.成员名”例如“boy1.num”即第一个人的学号，“boy2.sex”即第二个人的性别。如果成员本身又是一个结构，则必须逐级找到最低级的成员才能使用。例如“boy1.birthday.month”即第一个人出生的月份成员，可以在程序中单独使用，与普通变量完全相同。结构变量的赋值就是给各成员赋值。

如果结构变量是全局变量或为静态变量，则可对它进行初始化赋值。对局部或自动结构变量不能进行初始化赋值。

## 结构数组

数组的元素也可以是结构类型的。因此可以构成结构型数组。结构数组的每一个元素都是具有相同结构类型的下标结构变量。在实际应用中，经常用结构数组来表示具有相同数据结构的一个群体。如一个班的学生档案，一个车间职工的工资表等。

结构数组的定义方法和结构变量相似，只需说明它为数组类型即可。例如：

```
struct stu
{
    int num;
    char *name;
```

```
char sex;
float score;
}boy[5];
```

以上定义了一个结构数组 boy1，共有 5 个元素，boy[0]~boy[4]。每个数组元素都具有 struct stu 的结构形式。

### 结构指针变量

当使用一个指针变量指向一个结构变量时，称之为结构指针变量。结构指针变量中的值是所指向的结构变量的首地址。通过结构指针即可访问该结构变量，这与数组指针和函数指针的情况是相同的。结构指针变量说明的一般形式为：

struct 结构名\*结构指针变量名

例如，要说明一个指向 stu 的指针变量 pstu，可写为：

```
struct stu *pstу;
```

当然也可在定义 stu 结构时同时说明 pstu。与前面讨论的各类指针变量相同，结构指针变量也必须要先赋值后才能使用。赋值是把结构变量的首地址赋予该指针变量，不能把结构名赋予该指针变量。如果 boy 是被说明为 stu 类型的结构变量，则 “pstу=&boy” 是正确的，而 “pstу=&stu” 是错误的。

结构名和结构变量是两个不同的概念，不能混淆。结构名只能表示一个结构形式，编译系统并不对它分配内存空间。只有当某变量被说明为这种类型的结构时，对该变量分配存储空间。因此上面 “&stu” 写法是错误的，不可能去取一个结构名的首地址。有了结构指针变量，就能更方便地访问结构变量的各个成员。

其访问的一般形式为： (\*结构指针变量).成员名 或为：

结构指针变量->成员名

例如 “(\*pstу).num” 或者 “pstу->num”。

应该注意(\*pstу)两侧的括号不可少，因为成员符 “.” 的优先级高于 “\*”。如去掉括号写作 \*pstу.num 则等效于\*(pstу.num)，这样，意义就完全不对了。

```
结构变量.成员名
```

```
(*结构指针变量).成员名
```

```
结构指针变量->成员名
```

这 3 种用于表示结构成员的形式是完全等效的。结构指针变量可以指向一个结构数组，这时结构指针变量的值是整个结构数组的首地址。结构指针变量也可指向结构数组的一个元素，这时结构指针变量的值是该结构数组元素的首地址。设 ps 为指向结构数组的指针变量，则 ps 也指向该结构数组的 0 号元素，ps+1 指向 1 号元素，ps+i 则指向 i 号元素。这与普通数组的情况是一致的。

### 结构指针变量作为函数参数

在 ANSI C 标准中允许用结构变量作为函数参数进行整体传送。但是这种传送要将全部成员逐个传送，特别是成员为数组时将会使传送的时间和空间开销很大，严重地降低了程序的效率。因此最好的办法就是使用指针，即用指针变量作为函数参数进行传送。这时由实参传向形参的只是地址，从而减少了时间和空间的开销。

在本例中，设一个通信录由以下几项数据信息构成：

数据项	类型
姓名	字符串
地址	字符串

邮政编码	字符串
电话号码	字符串

本例要求为通信录数据定义类型和定义通信录变量，并写出输入一个通信录和输出一个通信录的函数。

在 C 程序中，通信录数据可用一个结构类型来描述，设其中的姓名、地址、邮政编码和电话号码都用字符数组来存储，各数组的长度分别为 20, 40, 10 和 10。通信录数据类型如下：

```
#define NAMELEN 20
#define ADDRLEN 40
#define ZIPLEN 10
#define PHONLEN 10
struct addr
{
    char name[NAMELEN]; /*姓名*/
    char address[ADDRLEN]; /*地址*/
    char zip[ZIPLEN]; /*邮政编码*/
    char phone[PHONLEN]; /*电话号码*/
};
```

因其中的姓名、地址对不同的人可能需要不同数量的字符，更好的方法是对其中的姓名和地址用字符指针来表示，实际存储姓名和地址的存储空间可向系统申请得到。重新说明通信录数据类型如下：

```
#define ZIPLEN 10
#define PHONLEN 10
struct addr
{char *name; /*姓名*/
char *address; /*地址*/
char zip[ZIPLEN]; /*邮政编码*/
char phone[PHONLEN]; /*电话号码*/
};
```

相应地以下变量定义都是通信录变量的定义：

```
struct addr col,cdm[100];
```

其中 col 是单个通信录变量， cdm[] 是一个通信录变量数组。

设输入通信录的函数以指向通信录变量的指针为参数，当正确输入一个通信录时，函数返回 1，不能正常输入时，函数返回 0，并采用交互式方式输入通信录的每一项数据。

## 程序代码

### 【程序 31】 通讯录的输入和输出

```
//本实例源码参见光盘
```

## 归纳注释

关于动态内存分配问题：在数组中，数组的长度是预先定义好的，在整个程序中固定不变。C 语言中不允许动态数组类型。例如 “int n;scanf(“%d”,&n);int a[n];” 用变量表示长度，这是错误的。但是在实际的编程中，往往会发生这种情况，即所需的内存空间取决于实际输入的数据，而无法预先确定。对于这种问题，用数组的办法很难解决。为了解决上述问题，C 语言提供了一些内存管理函数，这些内存管理函数可以按需要动态地分配内存空间，也可把不再使用的空间回收待用，为有效地利用内存资源提供了手段。常用的内存管理函数有以下 3 个。

### (1) 分配内存空间函数 malloc

调用形式:

```
(类型说明符*) malloc (size)
```

在内存的动态存储区中分配一块长度为“size”字节的连续区域。函数的返回值为该区域的首地址。“类型说明符”表示把该区域用于何种数据类型。“(类型说明符\*)”表示把返回值强制转换为该类型指针。“size”是一个无符号数。例如“pc=(char \*) malloc (100);”表示分配 100 个字节的内存空间，并强制转换为字符数组类型，函数的返回值为指向该字符数组的指针，把该指针赋予指针变量 pc。

### (2) 分配内存空间函数 calloc

calloc 也用于分配内存空间。调用形式:

```
(类型说明符*) calloc(n,size)
```

在内存动态存储区中分配 n 块长度为“size”字节的连续区域。函数的返回值为该区域的首地址。“(类型说明符\*)”用于强制类型转换。calloc 函数与 malloc 函数的区别仅在于一次可以分配 n 块区域。例如“ps=(struct stu\*) calloc(2,sizeof (struct stu));”其中的 sizeof(struct stu) 是计算 stu 的结构长度。因此该语句的意思是按 stu 的长度分配 2 块连续区域，强制转换为 stu 类型，并把其首地址赋予指针变量 ps。

### (3) 释放内存空间函数 free

调用形式:

```
free(void*ptr);
```

释放 ptr 所指向的一块内存空间，ptr 是一个任意类型的指针变量，它指向被释放区域的首地址。被释放区应是由 malloc 或 calloc 函数所分配的区域。

## 实例 32 扑克牌的结构表示



使用“结构”定义一副扑克牌，并对变量赋值。程序运行结果如图 32-1 所示。

```
CD\TC
<CLUBS A> <DIAMONDS A> <HEARTS A> <SPADES A>
<CLUBS 2> <DIAMONDS 2> <HEARTS 2> <SPADES 2>
<CLUBS 3> <DIAMONDS 3> <HEARTS 3> <SPADES 3>
<CLUBS 4> <DIAMONDS 4> <HEARTS 4> <SPADES 4>
<CLUBS 5> <DIAMONDS 5> <HEARTS 5> <SPADES 5>
<CLUBS 6> <DIAMONDS 6> <HEARTS 6> <SPADES 6>
<CLUBS 7> <DIAMONDS 7> <HEARTS 7> <SPADES 7>
<CLUBS 8> <DIAMONDS 8> <HEARTS 8> <SPADES 8>
<CLUBS 9> <DIAMONDS 9> <HEARTS 9> <SPADES 9>
<CLUBS 10> <DIAMONDS 10> <HEARTS 10> <SPADES 10>
<CLUBS J> <DIAMONDS J> <HEARTS J> <SPADES J>
<CLUBS Q> <DIAMONDS Q> <HEARTS Q> <SPADES Q>
<CLUBS K> <DIAMONDS K> <HEARTS K> <SPADES K>
Press any key to quit...
```

图 32-1 实例 32 程序运行结果



扑克牌有 4 种花色：草花、方块、红心和黑桃，可将花色说明为枚举类型。扑克牌的类型为结构类型，包含两个成分，分别存储牌的花色和牌的面值，其中面值为字符数组。



## 程序代码

### 【程序 32】 扑克牌的结构表示

```
enum suits{CLUBS,DIAMONDS,HEARTS,SPADES};
struct card
{
    enum suits suit;
    char value[3];
};

struct card deck[52];
char cardval[][3]={"A","2","3","4","5","6","7","8","9","10","J","Q","K"};
char suitsname[][9]={"CLUBS","DIAMONDS","HEARTS","SPADES"};

main()
{
    int i,j;
    enum suits s;
    clrscr();
    for(i=0;i<=12;i++)
        for(s=CLUBS;s<=SPADES;s++)
        {
            j=i*4+s;
            deck[j].suit=s;
            strcpy(deck[j].value,cardval[i]);
        }
    for(j=0;j<52;j++)

        printf("(%s%3s)%c",suitsname[deck[j].suit],deck[j].value,j%4==3?'\\n':'\\t');
    puts("\nPress any key to quit...");
    getch();
}
```



## 归纳注释

在实际问题中，有些变量的取值被限定在一个有限的范围内。例如，一个星期内只有七天，一年只有十二个月，一个班每周有六门课程等等。如果把这些量说明为整型，字符型或其他类型显然是不妥当的。为此，C 语言提供了一种称为“枚举”的类型。应该说明的是，枚举类型是一种基本数据类型，而不是一种构造类型，因为它不能再分解为任何基本类型。

## 实例 33 用“结构”统计学生成绩



## 实例说明

设学生信息包括学号、姓名和五门功课的成绩，要求编写输入输出学生信息的函数。在输入一组学生信息后，以学生成绩的总分从高到低顺序输出学生信息。程序运行结果如图 33-1 所示。

```
Number : 1031101
Name : Herry
Scores : 80 90 70 68 92

Number : 1031102
Name : Marry
Scores : 78 89 93 82 92

Number : ^Z
Number : 1031102
Name : Marry
Scores : 78 89 93 82 92

Number : 1031101
Name : Herry
Scores : 80 90 70 68 92

Press any key to quit...
```

图 33-1 实例 33 程序运行结果

## 实例解析

学生信息的学号用 10 个字符来表示；学生的姓名在学生结构里只存储姓名字符串的指针，实际存储学生姓名的空间向系统申请；成绩用一个整数数组来存储。存储学生信息的变量的数据类型说明如下：

```
#define SCORES 5
#define NUMLEN 10
struct std_type
{
    char no[NUMLEN]; /*学号*/
    char *name; /*名字字符串指针*/
    char scores[SCORES]; /*五门功课的成绩*/
};
```

设输入学生信息的函数以存储学生信息的结构变量的指针为参数，当正确输入一个学生信息时，函数返回 1，不能正常输入时，函数返回 0，并采用交互方式输入学生信息的每一项数据。而输出学生信息的函数的参数也是指向存储学生信息的变量的指针，一个学生信息的 3 项数据分别输出在 3 行上。

程序引入一个结构数组依次存储输入的学生信息，为了在一组学生信息排序时避免交换整个学生结构，另外引入一个存储下标的数组。开始时，该数组依次存储各学生结构在结构数组中的下标，当排序过程中要改变两个学生结构的顺序时，就改变对应下标的顺序。此外，为了避免反复求学生总分，又开设一个数组存储各位学生的总分。

## 程序代码

### 【程序 33】 用“结构”统计学生成绩

```
#include <stdio.h>
#define N 200
#define SCORES 5
#define NUMLEN 10
struct std_type{
    char no[NUMLEN]; /*学号*/
    char *name; /*名字字符串指针*/
    int scores[SCORES]; /*五门功课的成绩*/
};
```

```

struct std_type students[N];
int order[N];
int total[N];

/*[函数]输入一个学生信息函数*/
int readastu(struct std_type *spt)
{
    int len,j;
    char buf[120];/*输入字符串的缓冲区*/

    printf("\nNumber : ");/*输入学号*/
    if(scanf("%s",buf)==1)
        strncpy(spt->no,buf,NUMLEN-1);
    else
        return 0;/*Ctrl+Z 结束输入*/
    printf("Name : ");/*输入姓名*/
    if(scanf("%s",buf)==1)
    {
        len=strlen(buf);
        spt->name=(char *)malloc(len+1);/*申请存储姓名的空间*/
        strcpy(spt->name,buf);
    }
    else return 0;/*Ctrl+Z 结束输入*/
    printf("Scores : ");/*输入成绩*/
    for(j=0;j<SCORES;j++)
        if(scanf("%d",spt->scores+j)!=1)
            break;
        if(j==0)/*一个成绩也未输入*/
    {
        free(spt->name);/*释放存储姓名的空间*/
        return 0;
    }
    for(;j<SCORES;j++)/*少数未输入的成绩用 0 分代替*/
        spt->scores[j]=0;
    return 1;
}

/*[函数]输出一个学生信息的函数*/
int writeastu(struct std_type *spt)
{
    int i;

    printf("Number : %s\n",spt->no);/*输出学号*/
    printf("Name : %s\n",spt->name);/*输出姓名*/
    printf("Scores : ");/*输出成绩*/
    for(i=0;i<SCORES;i++)
        printf("%4d",spt->scores[i]);
    printf("\n\n");
}

main()
{
    int n,i,j,t;

```

```

clrscr();
for(n=0;readastu(students+n);n++)
/*采用冒泡法对学生信息数组排序*/
for(i=0;i<n;i++)
{
    order[i]=i; /*预置第 i 个输入的学生*/
    for(t=0,j=0;j<SCORES;j++) /*求第 i 个学生的总分*/
        t+=students[i].scores[j];
    total[i]=t;
}
/*冒泡排序*/
for(i=0;i<n-1;i++) /*共扫描 n-1 遍*/
    for(j=0;j<n-1-i;j++)
        if(total[order[j]]<total[order[j+1]])
            /*交换名次*/
            t=order[j];
            order[j]=order[j+1];
            order[j+1]=t;
for(j=0;j<n;j++) /*输出*/
    writeastu(students+order[j]);
printf("\n Press any key to quit...\n");
getch();
}

```



## 归纳注释

程序在对学生成绩的总分进行排序时，采用了冒泡排序的排序算法。关于“冒泡”排序的更多内容可参考【实例 43】。

# 实例 34 报数游戏



## 实例说明

设由  $n$  个人站成一个圈，分别被编号 1, 2, ……,  $n$ 。第一个人从 1 开始报数，每报数为  $m$  的人被从圈中推出，其后的人再次从 1 开始报数，重复上述过程，直至所有人都从圈中退出。要求程序由用户输入整数  $m$  和  $n$ ，求这  $n$  个人从圈中退出的先后顺序。

```

TC
Please input n&m:
29 3
The numbers of who will quit the cycle in turn are:
3 6 9 12 15 18 21 24 27 1 5 10 14 19 23 28 4 11 17 25
Press any key to quit...

```

图 34-1 实例 34 程序运行结果

程序运行结果如图 34-1 所示。

## 实例解析

可利用链表求解本问题，先由 n 形成一个有 n 个表元组成的环，其中 n 个表元依次置值 1~n。然后，从环的第一个表元出发，连续略过 m-1 个表元，第 m-1 个表元的后继表元是第 m 个表元，将该表元从环中退出。接着再次从下一个表元出发，重复以上过程，直至环中表元都退出为止。

## 程序代码

### 【程序 34】 报数游戏

//本实例源码参见光盘

## 归纳注释

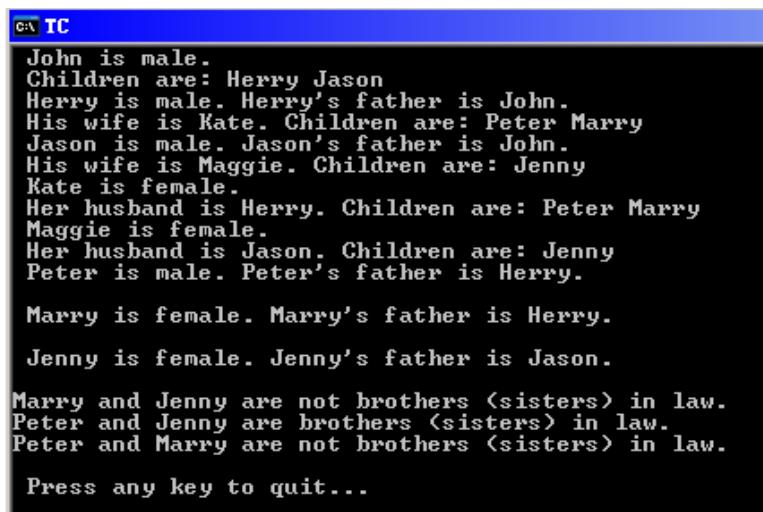
程序首先定义链表结构，接着清屏，提示输入 n 和 m 的值，接着申请空间，设置链表表元编号，然后依次打印输出报数出圈的编号。

## 实例 35 模拟社会关系

## 实例说明

设计一个模拟社会关系的数据结构，每个人的信息用结构表示，包含名字、性别和指向父亲、母亲、配偶、子女的指针（设只限两个子女）。要求编写以下函数：（1）增加一个新人的函数；（2）建立人与人之间关系的函数：父—子、母—子、配偶等；（3）检查某两人之间是否为堂兄妹。

程序运行结果如图 35-1 所示。



```
John is male.
Children are: Herry Jason
Herry is male. Herry's father is John.
His wife is Kate. Children are: Peter Marry
Jason is male. Jason's father is John.
His wife is Maggie. Children are: Jenny
Kate is female.
Her husband is Herry. Children are: Peter Marry
Maggie is female.
Her husband is Jason. Children are: Jenny
Peter is male. Peter's father is Herry.

Marry is female. Marry's father is Herry.

Jenny is female. Jenny's father is Jason.

Marry and Jenny are not brothers <sisters> in law.
Peter and Jenny are brothers <sisters> in law.
Peter and Marry are not brothers <sisters> in law.

Press any key to quit...
```

图 35-1 实例 35 程序运行结果



## 实例解析

由问题的要求，描述人的数据类型可说明如下：

```
#define CHILDREN 2
struct person{
    char *name; /*名字字符串指针*/
    char sex; /*性别：男性用字符‘M’；女性用字符‘F’*/
    struct person *father; /*指向父亲*/
    struct person *mother; /*指向母亲*/
    struct person *mate; /*指向配偶*/
    struct person *children[CHILDREN]; /*指向子女*/
}
```

设增加一个新人时，只为新人提供名字和性别，关于新人的其他信息通过调用其他函数建立。增加新人的函数首先向系统申请一个存储人的信息的空间，然后再申请存储名字的空间，填写名字字符串和性别字符，将表示其他的人与人之间关系的指针置空，最后返回新人数据的指针。

建立父—子、母—子、配偶等人与人之间关系的函数，给出两个人的指针，函数根据其功能在代表两个人的变量中分别填写有关的指针。

某两人之间是否是堂兄妹，要看他们的父亲是否为兄弟，或他们的父亲的父亲是否为兄弟等等。若两人是堂兄妹，函数返回 1，否则函数返回 0。



## 程序代码

### 【程序 35】 模拟社会关系

//本实例源码参见光盘



## 归纳注释

主程序用于验证前面的几个函数。先新增几个人，然后分别建立他们的关系：John（男）有两个儿子 Herry 和 Jason，他们的妻子分别是 Kate 和 Maggie。Herry 和 Kate 有一个儿子 Peter 和一个女儿 Marry。Jason 和 Maggie 有一个女儿 Jenny。显然，Peter 和 Jenny 的父亲是亲兄弟，所以，他们是堂兄妹。

# 实例 36 统计文件的字符数



## 实例说明

编写统计文件的字符数的程序，主要学习文件的概念和文件操作等内容。程序运行结果如图 36-1 所示。

```
C:\ TC
Please input the file's name:
36.c
There are 454 characters in file 36.c.
Press any key to quit...
```

图 36-1 实例 36 程序运行结果



## 实例解析

所谓“文件”是指一组相关数据的有序集合。这个数据集有一个名称，叫做文件名。实际上在前面的各章中已经多次使用了文件，例如源程序文件、目标文件、可执行文件、库文件（头文件）等。文件通常是驻留在外部介质（如磁盘等）上的，在使用时才调入内存中来。从不同的角度可对文件作不同的分类。

从用户的角度看，文件可分为普通文件和设备文件两种。普通文件是指驻留在磁盘或其他外部介质上的有序数据集，可以是源文件、目标文件、可执行程序；也可以是一组待输入处理的原始数据，或者是一组输出的结果。源文件、目标文件、可执行程序可以称做程序文件，对输入输出数据可称做数据文件。设备文件是指与主机相联的各种外部设备，如显示器、打印机、键盘等。在操作系统中，把外部设备也看作是一个文件来进行管理，把它们的输入、输出等同于对磁盘文件的读和写。通常把显示器定义为标准输出文件，一般情况下，在屏幕上显示有关信息就是向标准输出文件输出。如前面经常使用的 `printf`, `putchar` 函数就是这类输出。键盘通常被指定为标准的输入文件，从键盘上输入就意味着从标准输入文件上输入数据。`Scanf`, `getchar` 函数就属于这类输入。

从文件编码的方式来看，文件可分为 ASCII 码文件和二进制码文件两种。

ASCII 文件也称为文本文件，这种文件在磁盘中存放时每个字符对应一个字节，用于存放对应的 ASCII 码。例如，数 5678 的存储形式为：

ASCII 码： 00110101 00110110 00110111 00111000

↓      ↓      ↓      ↓

十进制码： 5      6      7      8

共占用 4 个字节。ASCII 码文件可在屏幕上按字符显示，例如源程序文件就是 ASCII 文件，用 DOS 命令 TYPE 可显示文件的内容。由于是按字符显示，因此能读懂文件内容。

二进制文件是按二进制的编码方式来存放文件的。例如，数 5678 的存储形式为 00010110 00101110 只占两个字节。二进制文件虽然也可在屏幕上显示，但其内容无法读懂。C 系统在处理这些文件时，并不区分类型，都看成是字符流，按字节进行处理。输入输出字符流的开始和结束只由程序控制而不受物理符号（如回车符）的控制。因此也把这种文件称作“流式文件”。

文件指针在 C 语言中用一个指针变量指向一个文件，这个指针称为文件指针。通过文件指针就可对它所指的文件进行各种操作。定义文件指针的一般形式为“FILE\* 指针变量标识符；”其中 FILE 应为大写，它实际上是由系统定义的一个结构，该结构中含有文件名、文件状态和文件当前位置等信息。在编写源程序时不必关心 FILE 结构的细节。例如“FILE \*fp；”表示 fp 是指向 FILE 结构的指针变量，通过 fp 即可找到存放某个文件信息的结构变量，然后按结构变量提供的信息找到该文件，实施对文件的操作。习惯上也笼统地把 fp 称为指向一个文件的指针。

在 C 语言中，文件操作都是由库函数来完成的。文件操作主要有打开、读写和关闭等。

### 文件打开函数 `fopen`

`fopen` 函数用来打开一个文件，其调用的一般形式为：

文件指针名=`fopen(文件名, 使用文件方式)`

其中，“文件指针名”必须被说明为 FILE 类型的指针变量，“文件名”是被打开文件的文件名。“使用文件方式”是指文件的类型和操作要求。“文件名”是字符串常量或字符串数组。例如：

```
FILE *fp;  
fp=( "file.a", "r" );
```

其意义是在当前目录下打开文件 file.a，只允许进行“读”操作，并且使 fp 指向该文件。

又如：

```
FILE *fphzk  
fphzk= ("c:\\hzk16", "rb")
```

其意义是打开 C 驱动器磁盘的根目录下的文件 hzk16，这是一个二进制文件，只允许按二进制方式进行读操作。两个反斜线 “\\” 中的第一个表示转义字符，第二个表示根目录。使用文件的方式共有 12 种，如表 36-1 所示。

表 36-1 文件的使用方式

文件使用方式	意    义	文件使用方式	意    义
“rt”	只读打开一个文本文件，只允许读数据	“rt+”	读写打开一个文本文件，允许读和写
“wt”	只写打开或建立一个文本文件，只允许写数据	“wt+”	读写打开或建立一个文本文件，允许读写
“at”	追加打开一个文本文件，并在文件末尾写数据	“at+”	读写打开一个文本文件，允许读，或在文件末追加数据
“rb”	只读打开一个二进制文件，只允许读数据	“rb+”	读写打开一个二进制文件，允许读和写
“wb”	只写打开或建立一个二进制文件，只允许写数据	“wb+”	读写打开或建立一个二进制文件，允许读和写
“ab”	追加打开一个二进制文件，并在文件末尾写数据	“ab+”	读写打开一个二进制文件，允许读，或在文件末追加数据

对于文件使用方式有以下几点说明。

文件使用方式由 r、w、a、t、b、+共 6 个字符组合成，各字符的含义是：r(read)读；w(write)写；a append)追加；t(text)文本文件，可省略不写；b(banary)二进制文件；“+”读和写。

凡用“r”打开一个文件时，该文件必须已经存在，且只能从该文件读出。

用“w”打开的文件只能向该文件写入。若打开的文件不存在，则以指定的文件名建立该文件，若打开的文件已经存在，则将该文件删去，重建一个新文件。

若要向一个已存在的文件追加新的信息，只能用“a”方式打开文件。但此时该文件必须是存在的，否则将会出错。

在打开一个文件时，如果出错，fopen 将返回一个空指针值 NULL。在程序中可以用这一信息来判别是否完成打开文件的工作，并做相应的处理。因此常用以下程序段打开文件：

```
if((fp=fopen("c:\\hzk16", "rb"))==NULL)  
{  
    printf("\nerror on open c:\\hzk16 file!");  
    getch();  
    exit(1);  
}
```

这段程序的意义是，如果返回的指针为空，表示不能打开 C 盘根目录下的 hzk16 文件，则给出提示信息“error on open c:\\hzk16 file！”，下一行 getch()的功能是从键盘输入一个字符，但不在屏幕上显示。在这里，语句的作用是等待，只有当用户从键盘敲任意键时，程序才继续执行，因此用户可利用这个等待时间阅读出错提示。

把一个文本文件读入内存时，要将 ASCII 码转换成二进制码，而把文件以文本方式写入磁盘时，也要把二进制码转换成 ASCII 码，因此文本文件的读写要花费较多的转换时间。对二进制文件的读写不存在这种转换。

标准输入文件（键盘），标准输出文件（显示器），标准出错输出（出错信息）是由系统打开的，可直接使用。文件一旦使用完毕，应用关闭文件函数 fclose 把文件关闭，以避免文件的数据

丢失等错误。

### 文件关闭函数 fclose

调用的一般形式是：

`fclose(文件指针);`

正常完成关闭文件操作时，fclose 函数返回值为 0。如返回非零值则表示有错误发生。

### 文件读写函数

在 C 语言中提供了多种文件读写的函数。字符读写函数 fgetc 和 fputc；字符串读写函数 fgets 和 fputs；数据块读写函数 fread 和 fwrite；格式化读写函数 fscanf 和 fprintf。使用以上函数都要求包含头文件 stdio.h。字符读写函数 fgetc 和 fputc 字符读写函数是以字符（字节）为单位的读写函数。每次可从文件读出或向文件写入一个字符。

#### 读字符函数 fgetc

fgetc 函数的功能是从指定的文件中读一个字符，函数调用的形式为：

`字符变量=fgetc(文件指针);`

例如 “ch=fgetc(fp);” 的意义是从打开的文件 fp 中读取一个字符并送入 ch 中。

对于 fgetc 函数的使用有以下几点说明：在 fgetc 函数调用中，读取的文件必须是以读或读写方式打开的。读取字符的结果也可以不向字符变量赋值，例如 “fgetc(fp);”，但是读出的字符不能保存。在文件内部有一个位置指针。用来指向文件的当前读写字节。在文件打开时，该指针总是指向文件的第一个字节。使用 fgetc 函数后，该位置指针将向后移动一个字节。因此可连续多次使用 fgetc 函数，读取多个字符。

应注意文件指针和文件内部的位置指针是不同的。文件指针是指向整个文件的，需在程序中定义说明，只要不重新赋值，文件指针的值是不变的。文件内部的位置指针用以指示文件内部的当前读写位置，每读写一次，该指针均向后移动，它不需在程序中定义说明，而是由系统自动设置的。

#### 写字符函数 fputc

fputc 函数的功能是把一个字符写入指定的文件中，函数调用的形式为：

`fputc(字符量, 文件指针);`

其中，待写入的字符量可以是字符常量或变量，例如 “fputc('a',fp);” 的意义是把字符 a 写入 fp 所指向的文件中。

对于 fputc 函数的使用有以下几点说明：被写入的文件可以用写、读写、追加方式打开，用写或读写方式打开一个已存在的文件时将清除原有的文件内容，写入字符从文件首开始。如需保留原有文件内容，希望写入的字符以文件末开始存放，必须以追加方式打开文件。每写入一个字符，文件内部位置指针将向后移动一个字节。fputc 函数有一个返回值，如写入成功则返回写入的字符，否则返回 EOF，以此来判断写入是否成功。

#### 字符串读写函数 fgets 和 fputs

读字符串函数 fgets 函数的功能是从指定的文件中读一个字符串到字符数组中，函数调用的形式为：

`fgets(字符数组名, n, 文件指针);`

其中 n 是一个正整数，表示从文件中读出的字符串不超过 n-1 个字符。在读入的最后一个字符后加上串结束标志'\0'。例如 “fgets(str,n,fp);” 的意义是从 fp 所指的文件中读出 n-1 个字符送

入字符数组 str 中。

对 fgets 函数有两点说明：在读出 n-1 个字符之前，如遇到了换行符或 EOF，则结束读操作。fgets 函数也有返回值，其返回值是字符数组的首地址。

写字符串函数 fputs 函数的功能是向指定的文件写入一个字符串，其调用形式为：

```
fputs(字符串, 文件指针);
```

其中字符串可以是字符串常量，也可以是字符数组名，或指针变量，例如“fputs(“abcd”, fp);” 的意义是把字符串 “abcd” 写入 fp 所指的文件之中。

### 数据块读写函数 fread 和 fwrite

C 语言还提供了用于整块数据的读写函数。可用来读写一组数据，如数组元素、结构变量的值等。读数据块函数调用的一般形式为：

```
fread(buffer, size, count, fp);
```

写数据块函数调用的一般形式为：

```
fwrite(buffer, size, count, fp);
```

其中 buffer 是一个指针，在 fread 函数中，它表示存放输入数据的首地址。在 fwrite 函数中，它表示存放输出数据的首地址。size 表示数据块的字节数。count 表示要读写的数据块块数。fp 表示文件指针。例如 “fread(fa,4,5,fp);” 其意义是从 fp 所指的文件中，每次读 4 个字节（一个实数）送入实数组 fa 中，连续读 5 次，即读 5 个实数到 fa 中。

### 格式化读写函数 fscanf 和 fprintf

fscanf 函数，fprintf 函数与前面使用的 scanf 和 printf 函数的功能相似，都是格式化读写函数。两者的区别在于 fscanf 函数和 fprintf 函数的读写对象不是键盘和显示器，而是磁盘文件。这两个函数的调用格式为：

```
fscanf(文件指针, 格式字符串, 输入表列);
```

```
fprintf(文件指针, 格式字符串, 输出表列);
```

例如：

```
fscanf(fp, "%d%s", &i, s);
```

```
fprintf(fp, "%d%c", j, ch);
```

在本例中，对输入文件中的字符逐个读入并处理的程序结构有以下形式：

```
{  
    读入输入文件的名字;  
    以读方式打开文件;  
    如果不能打开  
    {  
        报告文件不能打开;  
        结束程序;  
    }  
    设置处理的初值;  
    while (能正常读入一个字符 (文件未结束))  
    {  
        对当前读入的字符做某种处理;  
    }  
    关闭文件;  
    输出结果;  
}
```



## 程序代码

### 【程序 36】 统计文件字符数

```
#include <stdio.h>
main()
{
    char fname[80]; /*存储文件名*/
    FILE *rfp;
    long count; /*文件字符计数器*/

    clrscr();
    printf("Please input the file's name:\n");
    scanf("%s", fname);
    if((rfp=fopen(fname, "r"))==NULL)
    {
        printf("Can't open file %s.\n", fname);
        exit(1);
    }
    count=0;
    while(fgetc(rfp)!=EOF)
        count++;
    fclose(rfp); /*关闭文件*/
    printf("There are %ld characters in file %s.\n", count, fname);
    puts("\n Press any key to quit...");
    getch();
}
```



## 归纳注释

### 文件的随机读写

实际问题中常要求读写文件中某一指定的部分。为了解决这个问题可移动文件内部的位置指针到需要读写的位置，再进行读写，这种读写称为随机读写。实现随机读写的关键是要按要求移动位置指针，这称为文件的定位。

文件定位的函数主要有两个，rewind 函数和 fseek 函数。rewind 函数调用形式为：

```
rewind(文件指针);
```

它的功能是把文件内部的位置指针移到文件首。fseek 函数用来移动文件内部位置指针，其调用形式为：

```
fseek(文件指针, 位移量, 起始点);
```

其中“文件指针”指向被移动的文件。“位移量”表示移动的字节数，要求位移量是 long 型数据，以便在文件长度大于 64KB 时不会出错。当用常量表示位移量时，要求加后缀“L”。“起始点”表示从何处开始计算位移量，规定的起始点有 3 种：文件首、当前位置和文件尾。

起始点	表示符号	数字表示
文件首	SEEK—SET	0
当前位置	SEEK—CUR	1
文件末尾	SEEK—END	2

例如“fseek(fp,100L,0);”的意义是把位置指针移到离文件首 100 个字节处。

还要说明的是 fseek 函数一般用于二进制文件。在文本文件中由于要进行转换，故往往计算的位置会出现错误。文件的随机读写在移动位置指针之后，由于一般是读写一个数据块，因此常用 fread 和 fwrite 函数。

## 实例 37 同时显示两个文件的内容

### 实例说明

编制一个程序，实现将两个文件的内容同时显示在屏幕上，并且最左边的第 1 至第 30 列显示文件 1 的内容；右边第 41 至第 70 列显示文件 2 的内容；第 75 列至第 76 列显示两文件该行字符数总和；其余列显示空白符。另外，每输出 20 行内容后，另输出 2 行空行。程序运行结果如图 37-1 所示。

Column	Content	Length
1-30	File 1 Content	30
41-70	File 2 Content	30
75-76	Total Length	2
77-78	Blank	2
79-80	Blank	2

Press any key to quit...

图 37-1 实例 37 程序运行结果

### 实例解析

根据要求，主函数的主要工作包括：读入两文件的名字和打开文件；通过循环，控制在两个文件中有文件还未结束时顺序读入两文件当前行的内容并输出；最后关闭文件。

对于读文件并输出的循环，有几种不同情况需要考虑：在某文件已经结束的情况下应不再读该文件，其对应输出栏的内容都用空白符代替；另外，当前行的内容也不可能总是只有 30 个字符，为此不足部分也需要用空白符代替，超过时需分批读入，每批最多读 30 个字符。

程序引入函数 readline() 完成从指定的文件中读出字符并显示输出，对文件的当前行函数一次调用最多读出 30 个字符，函数返回实际读出字符并显示的字符数，以便让主函数计算填充空白符的个数并组织输出。主函数在一个循环中顺序以第一和第二文件调用函数 readline() 实现两文件对应行内容的左、右栏输出。另外，因每输出 20 行后还要输出 2 行空行，程序引入输出行计数器和函数 linecount()。主函数在输出一行后，就调用该函数。由它完成对输出行的计数和一页满后输出 2 行空行。

### 程序代码

#### 【程序 37】 同时显示两个文件的内容

//本实例源码参见光盘



## 归纳注释

C语言中把文件当作一个“流”，按字节进行处理。本实例将两个文件格式化输出，相当于简单的文字处理系统的“打印预览”功能。

# 实例 38 简单的文本编辑器



## 实例说明

编写一个简单的单行文本编辑器，设编辑命令有以下几种。

(1) E——指定所要编辑的文件。

(2) Q——结束编辑。

(3) R——用R命令后继的K行正文替代原始正文中的M行到N行的正文内容。命令格式为：

```
R K M N  
K行正文
```

其中K、M、N均为大于零的整数。

(4) I——将I命令后继的K行正文插入到原始正文第M行之后。命令格式为：

```
I K M  
K行正文
```

其中K、M均为大于零的整数。

(5) D——将原始正文中第M行至第N行的正文内容删去。命令格式为：

```
D M N
```

其中M、N均为大于零的整数。

程序只限编辑较短的正文，每行不超过80个字符，总行数不超过200行，正文行从0开始编号。程序运行结果如图38-1所示。

```
c:\ Command Prompt  
Input a command:[e,r,i,d,q].  
e test.txt  
The text is:  
  
Input a command:[e,r,i,d,q].  
i 3 0  
Hello!  
How are you?  
I am fine!  
The text is:  
Hello!  
How are you?  
I am fine!  
  
Input a command:[e,r,i,d,q].  
d 2 3  
The text is:  
Hello!  
  
Input a command:[e,r,i,d,q].  
q  
Save? <y/n>y
```

图38-1 实例38程序运行结果



## 实例解析

程序将读入的正文行字符串存储在向系统申请来的空间中，正文行的字符指针存储在一个指针数组中。



## 程序代码

### 【程序 38】 简单的文本编辑器

//本实例源码参见光盘



## 归纳注释

实例程序若加上可以实时显示的功能就是相当于 Linux 操作系统下 vi 编辑器的简单的行编辑器的功能。

# 实例 39 文件的字数统计程序



## 实例说明

编程实现统计一个或多个文件的行数、字数和字符数。程序运行结果如图 39-1 所示（设该程序文件名为 program.exe）。

```
C:\>Command Prompt
C:\>program
To run this program, usage: program -l -w -c file1 file2 ... filen
C:\>program 39.c
Lines =      91
Words =      171
Characters = 1903
C:\>program -w 39.c
Words =      171
C:\>program -wl 39.c
Lines =      91
Words =      171
C:\>program -wl 39.c 34.c 35.c
Lines =      261
Words =      535
C:\>
```

图 39-1 实例 39 程序运行结果



## 实例解析

一个行由一个换行符限定，一个字由空格分隔（包括空白符、制表符和换行符），字符是指

文件中的所有字符。要求程序另设 3 个任选的参数，让用户指定所要统计的内容。l——统计文件行数；w——统计文件字数；c——统计文件字符数。若用户未指定任选的参数，则表示 3 个统计都要。

运行本程序时的参数按以下格式给出：

-l -w -c 文件 1 文件 2 … 文件 n

其中，前 3 个任选参数 l、w、c 的出现与否和出现顺序任意，或任意组合在一起出现，如-lwc，-cwl，-lw，-wl，-lc，-cl，-cw 等。

为实现上述功能，程序引入 3 个计数器，分别用于统计行数、字数和字符数。行计数器在遇到字符行的第一个字符时增 1；字计数器在遇到每个字时增 1；字符计数器在遇到每个字符时增 1。为标识一行的开始和结束，引入一个标志变量，遇换行符时，该标志变量置 0，遇其他字符时，行计数器增 1，并置标志变量为 1；为表示一个字的开始和结束，也引入一个标志变量，遇空白类字符时，该标志变量置 0，遇其他字符时，字计数器增 1，并置标志变量为 1。程序另引入 3 个标志变量分别用于区分程序要统计的内容。



## 程序代码

### 【程序 39】 文件的字数统计程序

```
#include <stdio.h>
main(int argc, char **argv)
{
    FILE *fp;
    int lflg,wflg,cflg; /* l, w, c 3 个标志 */
    int inline,inword; /* 行内和字内标志 */
    int ccount,wcount,lcount; /* 字符, 字, 行 计数器 */
    int c;
    char *s;
    lflg=wflg=cflg=0;
    if(argc<2)
    {
        printf("To run this program, usage: program -l -w -c file1 file2 ... filen \n");
        exit(0);
    }
    while(--argc>=1&&(*++argv)[0]=='-')
    {
        for(s=argv[0]+1;*s]!='\0';s++)
        {
            switch(*s)
            {
                case 'l':
                    lflg=1;
                    break;
                case 'w':
                    wflg=1;
                    break;
                case 'c':
                    cflg=1;
                    break;
                default:
                    puts("To run this program, usage: program -l -w -c file1 file2 ...");
            }
        }
    }
}
```

```

filen");
        exit(0);
    }
}
if(lflg==0&&wflg==0&&cflg==0)
    lflg=wflg=cflg=1;
lcount=wcount=ccount=0;
while(--argc>=0)
{
    if((fp=fopen(*argv++,"r"))==NULL) /* 以只读方式打开文件 */
    {
        fprintf(stderr,"Can't open %s.\n",*argv);
        continue;
    }
    inword=inline=0;
    while((c=fgetc(fp))!=EOF)
    {
        if(cflg)
            ccount++;
        if(wflg)
            if(c=='\n' || c==' ' || c=='\t')
                inword=0;
            else if(inword==0)
            {
                wcount++;
                inword=1;
            }
        if(lflg)
            if(c=='\n')
                inline=0;
            else if.inline==0)
            {
                lcount++;
                inline=1;
            }
    }
    fclose(fp); /* 关闭文件 */
}
if(lflg)
    printf(" Lines =      %d\n",lcount);
if(wflg)
    printf(" Words =      %d\n",wcount);
if(cflg)
    printf(" Characters =  %d\n",ccount);
}

```



## 归纳注释

程序为了能同时对多个文件完成统计，把文件名参数定在最后，根据 main 函数的数组指针参数依次对多个文件进行统计。

# 实例 40 学生成绩管理程序

## 实例说明

编制一个统计存储在文件中的学生考试分数的管理程序。设学生成绩以一个学生一条记录的形式存储在文件中，每个学生记录包含的信息有姓名、学号和各门功课的成绩。要求编制具有以下几项功能的程序：求出各门课程的总分，平均分、按姓名，按学号寻找其记录并显示，浏览全部学生成绩和按总分由高到低显示学生信息等。程序运行结果如图 40-1~40-5 所示。

```
C:\ Command Prompt - exit
Please input the students marks record file's name: student.dat
The file student.dat doesn't exist, do you want to creat it? <Y/N> y
Please input the record number you want to write to the file: 2
Input the student's name: John
Input the student's code: 1041101
Input the Chinese mark: 80
Input the Mathematic mark: 90
Input the English mark: 83
Input the student's name: Mike
Input the student's code: 1041102
Input the Chinese mark: 85
Input the Mathematic mark: 94
Input the English mark: 88
Now you can input a command to manage the records.
m : mean of the marks.
t : total of the marks.
n : search record by student's name.
c : search record by student's code.
l : list all the records.
s : sort and list the records by the total.
q : quit!
Please input command:
```

图 40-1 实例 40 程序运行结果

```
C:\ Command Prompt - exit
Please input command:
l

Name : John
Code : 1041101
Marks :
    Chinese      : 80
    Mathematic   : 90
    English      : 83
Total : 253

Press ENTER to continue...

Name : Mike
Code : 1041102
Marks :
    Chinese      : 85
    Mathematic   : 94
    English      : 88
Total : 267

Press ENTER to continue...

Please input command:
```

图 40-2 输入 l 命令后的结果

```
C:\ Command Prompt - exit
Please input command:
s

Name : Mike
Code : 1041102
Marks :
    Chinese      : 85
    Mathematic   : 94
    English      : 88
Total : 267

Press ENTER to continue...

Name : John
Code : 1041101
Marks :
    Chinese      : 80
    Mathematic   : 90
    English      : 83
Total : 253

Press ENTER to continue...

Please input command:
```

图 40-3 输入 s 命令后的结果

```
cmd Command Prompt - exit
Please input command:
n
Please input the student's name you want to search: John
Name    : John
Code    : 1041101
Marks   :
    Chinese      : 80
    Mathematic   : 90
    English       : 83
Total   : 253
Please input command:
n
Please input the student's name you want to search: Kate
The student Kate is not in the file student.dat.
Please input command:
c
Please input the student's code you want to search: 100
The student 100 is not in the file student.dat.
Please input command:
c
Please input the student's code you want to search: 100
The student 100 is not in the file student.dat.
Please input command:
```

图 40-4 按姓名和学号查找记录的结果

```
cmd Command Prompt - exit
Please input command:
c
Please input the student's code you want to search: 1041102
Name    : Mike
Code    : 1041102
Marks   :
    Chinese      : 85
    Mathematic   : 94
    English       : 88
Total   : 267
Please input command:
m
Chinese      's average is: 82.50.
Mathematic   's average is: 92.00.
English       's average is: 85.50.
Please input command:
t
Chinese      's total mark is: 165.
Mathematic   's total mark is: 184.
English       's total mark is: 171.
Please input command:
```

图 40-5 计算平均分和总分

## 实例解析

设每位学生学习语文、数学和英语 3 门课程，主程序输入文件名之后，进入接受命令、执行命令处理程序的循环。

按问题的要求共设 5 条命令：求各门课程的总分、求各门课程的平均分、按学生名字寻找其信息、按学号寻找其信息、结束命令。

为求各门课程的总分，从文件逐一读出学生记录，累计各门课程的分数，待文件处理完即可得到各门课程的总分。为求各门课程的平均分，从文件逐一读出学生记录，累计各门课程的分数，并统计学生人数，待文件处理完毕，将得到的各门课程的总分除以人数就得到各门课程的平均分。按学生名字寻找学生信息的处理，首先要求输入待寻找学生的名字，顺序读入学生记录，凡名字与待寻找学生相同的记录都在屏幕上显示，直到文件结束。按学生学号寻找学生信息的处理，首先要求输入待寻找学生的学号，顺序读入学生记录，发现有学号与待寻找学生相同的记录就在屏

幕上显示，并结束处理。浏览学生全部成绩，顺序读入学生记录，并在屏幕上显示，直到文件结束。按总分由高到低显示学生信息，首先顺序读入学生记录并构造一个有序链表，然后顺序显示链表上的各元素。



## 程序代码

### 【程序 40】 学生成绩管理程序

```
#include <stdio.h>
#define SWN      3    /* 课程数 */
#define NAMELEN  20   /* 姓名最大字符数 */
#define CODELEN  10   /* 学号最大字符数 */
#define FNAMELEN 80   /* 文件名最大字符数 */
#define BUflen   80   /* 缓冲区最大字符数 */
/* 课程名称表 */
char schoolwork[SWN][NAMELEN+1] = {"Chinese", "Mathematic", "English"};
struct record
{
    char      name[NAMELEN+1];      /* 姓名 */
    char      code[CODELEN+1];      /* 学号 */
    int     marks[SWN];           /* 各课程成绩 */
    int     total;                /* 总分 */
}stu;

struct node
{
    char      name[NAMELEN+1];      /* 姓名 */
    char      code[CODELEN+1];      /* 学号 */
    int     marks[SWN];           /* 各课程成绩 */
    int     total;                /* 总分 */
    struct   node *next;          /* 后续表元指针 */
}*head; /* 链表首指针 */

int total[SWN];      /* 各课程总分 */
FILE *stfpt;         /* 文件指针 */
char stuf[FNAMELEN]; /* 文件名 */

/* 从指定文件读入一条记录 */
int readrecord(FILE *fpt, struct record *rpt)
{
    char buf[BUflen];
    int i;
    if(fscanf(fpt,"%s",buf)!=1)
        return 0;      /* 文件结束 */
    strncpy(rpt->name,buf,NAMELEN);
    fscanf(fpt,"%s",buf);
    strncpy(rpt->code,buf,CODELEN);
    for(i=0;i<SWN;i++)
        fscanf(fpt,"%d",&rpt->marks[i]);
    for(rpt->total=0,i=0;i<SWN;i++)
        rpt->total+=rpt->marks[i];
    return 1;
}
```

```

/* 对指定文件写入一条记录 */
writerecord(FILE *fpt, struct record *rpt)
{
    int i;
    fprintf(fpt, "%s\n", rpt->name);
    fprintf(fpt, "%s\n", rpt->code);
    for(i=0;i<SWN;i++)
        fprintf(fpt, "%d\n", rpt->marks[i]);
    return ;
}

/* 显示学生记录 */
displaystu(struct record *rpt)
{
    int i;
    printf("\nName : %s\n", rpt->name);
    printf("Code : %s\n", rpt->code);
    printf("Marks :\n");
    for(i=0;i<SWN;i++)
        printf("      %-15s : %4d\n", schoolwork[i], rpt->marks[i]);
    printf("Total : %4d\n", rpt->total);
}

/* 计算各单科总分 */
int totalmark(char *fname)
{
    FILE *fp;
    struct record s;
    int count,i;
    if((fp=fopen(fname, "r"))==NULL)
    {
        printf("Can't open file %s.\n", fname);
        return 0;
    }
    for(i=0;i<SWN;i++)
        total[i]=0;
    count=0;
    while(readrecord(fp,&s)!=0)
    {
        for(i=0;i<SWN;i++)
            total[i]+=s.marks[i];
        count++;
    }
    fclose(fp);
    return count; /* 返回记录数 */
}

/* 列表显示学生信息 */
void liststu(char *fname)
{
    FILE *fp;
    struct record s;
    if((fp=fopen(fname, "r"))==NULL)
    {

```

```

        printf("Can't open file %s.\n", fname);
        return ;
    }
    while(readrecord(fp,&s)!=0)
    {
        displaystu(&s);
        printf("\n      Press ENTER to continue...\n");
        while(getchar()!='\n');
    }
    fclose(fp);
    return;
}

/* 构造链表 */
struct node *makelist(char *fname)
{
    FILE *fp;
    struct record s;
    struct node *p,*u,*v,*h;
    int i;
    if((fp=fopen(fname,"r"))==NULL)
    {
        printf("Can't open file %s.\n", fname);
        return NULL;
    }
    h=NULL;
    p=(struct node *)malloc(sizeof(struct node));
    while(readrecord(fp,(struct record *)p)!=0)
    {
        v=h;
        while(v&&p->total<=v->total)
        {
            u=v;
            v=v->next;
        }
        if(v==h)
            h=p;
        else
            u->next=p;
        p->next=v;
        p=(struct node *)malloc(sizeof(struct node));
    }
    free(p);
    fclose(fp);
    return h;
}

/* 顺序显示链表各表元 */
void displaylist(struct node *h)
{
    while(h!=NULL)
    {
        displaystu((struct record *)h);
        printf("\n      Press ENTER to continue...\n");
    }
}

```

```

        while(getchar()!='\n');
        h=h->next;
    }
    return;
}
/* 按学生姓名查找学生记录 */
int retrievebyn(char *fname, char *key)
{
    FILE *fp;
    int c;
    struct record s;
    if((fp=fopen(fname, "r"))==NULL)
    {
        printf("Can't open file %s.\n", fname);
        return 0;
    }
    c=0;
    while(readrecord(fp,&s)!=0)
    {
        if(strcmp(s.name, key)==0)
        {
            displaystu(&s);
            c++;
        }
    }
    fclose(fp);
    if(c==0)
        printf("The student %s is not in the file %s.\n", key, fname);
    return 1;
}

/* 按学生学号查找学生记录 */
int retrievebyc(char *fname, char *key)
{
    FILE *fp;
    int c;
    struct record s;
    if((fp=fopen(fname, "r"))==NULL)
    {
        printf("Can't open file %s.\n", fname);
        return 0;
    }
    c=0;
    while(readrecord(fp,&s)!=0)
    {
        if(strcmp(s.code, key)==0)
        {
            displaystu(&s);
            c++;
            break;
        }
    }
    fclose(fp);
    if(c==0)

```

```

        printf("The student %s is not in the file %s.\n",key,fname);
        return 1;
    }

main()
{
    int i,j,n;
    char c;
    char buf[BUFSIZE];
    FILE *fp;
    struct record s;
    clrscr();
    printf("Please input the students marks record file's name: ");
    scanf("%s",stuf);
    if((fp=fopen(stuf,"r"))==NULL)
    {
        printf("The file %s doesn't exit, do you want to creat it? (Y/N) ",stuf);
        getchar();
        c=getchar();
        if(c=='Y' || c=='y')
        {
            fp=fopen(stuf,"w");
            printf("Please input the record number you want to write to the file: ");
            scanf("%d",&n);
            for(i=0;i<n;i++)
            {
                printf("Input the student's name: ");
                scanf("%s",&s.name);
                printf("Input the student's code: ");
                scanf("%s",&s.code);
                for(j=0;j<SWN;j++)
                {
                    printf("Input the %s mark: ",schoolwork[j]);
                    scanf("%d",&s.marks[j]);
                }
                writerecord(fp,&s);
            }
            fclose(fp);
        }
    }
    fclose(fp);
    getchar();
/*clrscr();*/
    puts("Now you can input a command to manage the records.");
    puts("m : mean of the marks.");
    puts("t : total of the marks.");
    puts("n : search record by student's name.");
    puts("c : search record by student's code.");
    puts("l : list all the records.");
    puts("s : sort and list the records by the total.");
    puts("q : quit!");
    while(1)
    {
        puts("Please input command:");

```

```

scanf(" %c",&c);           /* 输入选择命令 */
if(c=='q'||c=='Q')
{
    puts("\n Thank you for your using.");
    break;          /* q, 结束程序运行 */
}
switch(c)
{
    case 'm': /* 计算平均分 */
    case 'M':
        if((n=totalmark(stuf))==0)
        {
            puts("Error!");
            break;
        }
        printf("\n");
        for(i=0;i<SWN;i++)
printf("%-15s's average is: %.2f.\n",schoolwork[i], (float) total[i]/n);
        break;
    case 't': /* 计算总分 */
    case 'T':
        if((n=totalmark(stuf))==0)
        {
            puts("Error!");
            break;
        }
        printf("\n");
        for(i=0;i<SWN;i++)
printf("%-15s's total mark is: %d.\n",schoolwork[i], total[i]);
        break;
    case 'n': /* 按学生的姓名寻找记录 */
    case 'N':
        printf("Please input the student's name you want to search: ");

        scanf("%s",buf);
        retrievebyn(stuf,buf);
        break;
    case 'c': /* 按学生的学号寻找记录 */
    case 'C':
        printf("Please input the student's code you want to search: ");

        scanf("%s",buf);
        retrievebyc(stuf,buf);
        break;
    case 'l': /* 列出所有学生记录 */
    case 'L':
        liststu(stuf);
        break;
    case 's': /* 按总分从高到低排列显示 */
    case 'S':
        if((head=makelist(stuf))!=NULL)
            displaylist(head);
        break;
    default: break;
}

```

```
    }  
}  
}
```



## 归纳注释

本程序除为每个处理功能编写相应的函数外，另外编写从文件读学生记录的函数、写记录到文件和显示一个学生记录的函数，从而简化编程。

# 02

## 第二部分 数据结构篇

### 精彩导读

- 冒泡排序
- 堆排序
- 二叉树遍历
- 哈夫曼编码
- 图的遍历
- 八皇后问题

# 实例 41 插入排序

## 实例说明

将一个整数数组按从小到大的顺序进行排序，主要学习基本的插入排序和改进的冒泡排序的算法和应用。程序运行效果如图 41-1 所示。

```
Command Prompt - exit
Please input total element number of the sequence:
10
Please input the elements one by one:
12 34 89 90 89 1 78 56 2 2
The sequence you input is:
12 34 89 90 89 1 78 56 2 2
The sequence after insert_sort is:
1 2 2 12 34 56 78 89 89 90
Press any key to quit...
```

图 41-1 实例 41 插入排序程序运行结果

## 实例解析

### 排序 (sort)

所谓排序，就是要整理文件中的记录，使之按关键字递增（或递减）次序排列起来，其确切定义如下。

输入：n 个记录 R<sub>1</sub>, R<sub>2</sub>, …, R<sub>n</sub>，其相应的关键字分别为 K<sub>1</sub>, K<sub>2</sub>, …, K<sub>n</sub>。

输出：R<sub>i1</sub>, R<sub>i2</sub>, …, R<sub>in</sub>，使得 K<sub>i1</sub> ≤ K<sub>i2</sub> ≤ … ≤ K<sub>in</sub>。（或 K<sub>i1</sub> ≥ K<sub>i2</sub> ≥ … ≥ K<sub>in</sub>）。

被排序对象：文件，由一组记录组成。记录则由若干个数据项（或域）组成。其中有一项可用来标识一个记录，称为关键字项。该数据项的值称为关键字（Key）。在不易产生混淆时，将关键字项简称为关键字。

用来作为排序运算依据的关键字，可以是数字类型，也可以是字符类型。关键字的选取应根据问题的要求而定。例如，在高考成绩统计中将每个考生作为一个记录。每条记录包含准考证号、姓名、各科的分数和总分数等项内容。若要唯一地标识一个考生的记录，则必须用“准考证号”作为关键字。若要按照考生的总分数排名次，则需用“总分数”作为关键字。

### 排序的稳定性

当待排序记录的关键字均不相同时，排序结果是唯一的，否则排序结果不唯一。

在待排序的文件中，若存在多个关键字相同的记录，经过排序后，这些具有相同关键字记录之间的相对次序保持不变，该排序方法是稳定的；若具有相同关键字的记录之间的相对次序发生变化，则称这种排序方法是不稳定的。

注意：排序算法的稳定性是针对所有输入实例而言的。即在所有可能的输入实例中，只要有一个实例使得算法不满足稳定性要求，则该排序算法就是不稳定的。

## 排序方法的分类

### (1) 按是否涉及数据的内/外存交换分

在排序过程中，若整个文件都是放在内存中处理，排序时不涉及数据的内/外存交换，则称之为内部排序（简称内排序）；反之，若排序过程中要进行数据的内、外存交换，则称之为外部排序。

注意：① 内排序适用于记录个数不很多的小文件。② 外排序则适用于记录个数较多，不能一次将全部记录放入内存的大文件。

### (2) 按策略划分内部排序方法

可以分为 5 类：插入排序、选择排序、交换排序、归并排序和分配排序。

## 待排序文件的常用存储方式

(1) 以顺序表（或直接用向量）作为存储结构，其排序过程为对记录本身进行物理重排（即通过关键字之间的比较判定，将记录移到合适的位置）。

(2) 以链表作为存储结构，其排序过程无需移动记录，仅需修改指针。通常将这类排序称为链表（或链式）排序。

(3) 用顺序的方式存储待排序的记录，但同时建立一个辅助表（如包括关键字和指向记录位置的指针组成的索引表）。其排序过程为只需对辅助表的表目进行物理重排（即只移动辅助表的表目，而不移动记录本身）。此方法适用于难于在链表上实现，且需避免排序过程中移动记录的排序方法。

## 待排序文件的顺序存储结构表示

```
#define n 100 /* 假设的文件长度，即待排序的记录数目 */
typedef int KeyType; /* 假设的关键字类型 */
typedef struct{ /* 记录类型 */
    KeyType key; /* 关键字项 */
    InfoType otherinfo; /* 其他数据项，类型 InfoType 依赖于具体应用而定义 */
} RecType;
typedef RecType SeqList[n+1]; /* SeqList 为顺序表类型，表中第 0 个单元一般用作哨兵 */
```

若关键字类型没有比较算符，则可事先定义宏或函数来表示比较运算。例如，关键字为字符串时，可定义宏 “#define LT(a,b)(Stromp((a),(b))<0)” 。那么算法中 “a<b” 可用 “LT(a,b)” 取代。

## 排序算法性能评价

### (1) 评价排序算法好坏的标准

评价排序算法好坏的标准主要有两条：执行时间和所需的辅助空间；算法本身的复杂程度。

### (2) 排序算法的空间复杂度

若排序算法所需的辅助空间并不依赖于问题的规模  $n$ ，即辅助空间是  $O(1)$ ，则称之为就地排序（In-PlaceSort）。非就地排序一般要求的辅助空间为  $O(n)$ 。

### (3) 排序算法的时间开销

大多数排序算法的时间开销主要是关键字之间的比较和记录的移动。有些排序算法的执行时间不仅依赖于问题的规模，还取决于输入实例中数据的状态。

## 插入排序

插入排序（Insertion Sort）的基本思想是：每次将一个待排序的记录按其关键字大小插入到前面已经排好序的子文件中的适当位置，直到全部记录插入完成为止。

有两种插入排序方法：直接插入排序和希尔排序。

### 直接插入排序

#### (1) 基本思想

假设待排序的记录存放在数组  $R[1..n]$  中。初始时， $R[1]$  自成一个有序区，无序区为  $R[2..n]$ 。从  $i=2$  起直至  $i=n$  为止，依次将  $R[i]$  插入当前的有序区  $R[1..i-1]$  中，生成含  $n$  个记录的有序区。

#### (2) 第 $i-1$ 趟直接插入排序

通常将一个记录  $R[i](i=2,3,\dots,n-1)$  插入到当前的有序区，使得插入后仍保证该区间里的记录是按关键字有序的操作，称为第  $i-1$  趟直接插入排序。

排序过程的某一中间时刻， $R$  被划分成两个子区间  $R[1..i-1]$ （已排好序的有序区）和  $R[i..n]$ （当前未排序的部分，可称无序区）。

直接插入排序的基本操作是将当前无序区的第 1 个记录  $R[i]$  插入到有序区  $R[1..i-1]$  中适当的位置上，使  $R[1..i]$  变为新的有序区。因为这种方法每次使有序区增加 1 个记录，通常称增量法。

插入排序与打扑克时整理手上的牌非常类似。摸来的第 1 张牌无需整理，此后每次从桌上的牌（无序区）中摸最上面的 1 张并插入左手的牌（有序区）中正确的位置上。为了找到这个正确的位置，需自左向右（或自右向左）将摸来的牌与左手中已有的牌逐一比较。

### 一趟直接插入排序

#### (1) 简单方法

首先在当前有序区  $R[1..i-1]$  中查找  $R[i]$  的正确插入位置  $k(1 \leq k \leq i-1)$ ；然后将  $R[k..i-1]$  中的记录均后移一个位置，腾出  $k$  位置上的空间插入  $R[i]$ 。

注意：若  $R[i]$  的关键字大于等于  $R[1..i-1]$  中所有记录的关键字，则  $R[i]$  就插入原位置。

#### (2) 改进的方法

这是一种查找比较操作和记录移动操作交替进行的方法。

具体做法：将待插入记录  $R[i]$  的关键字从右向左依次与有序区中记录  $R[j](j=i-1,i-2,\dots,1)$  的关键字进行比较：①若  $R[j]$  的关键字大于  $R[i]$  的关键字，则将  $R[j]$  后移一个位置；②若  $R[j]$  的关键字小于或等于  $R[i]$  的关键字，则查找过程结束， $j+1$  即为  $R[i]$  的插入位置。

关键字比  $R[i]$  的关键字大的记录均已后移，所以  $j+1$  的位置已经腾空，只要将  $R[i]$  直接插入此位置即可完成一趟直接插入排序。

### 直接插入排序算法

算法描述如下：

```
void InsertSort(SeqList R)
{ //对顺序表R中的记录R[1..n]按递增序进行插入排序
    int i, j;
    for(i=2; i<=n; i++) //依次插入R[2], ..., R[n]
        if(R[i].key<R[i-1].key){ //若R[i].key大于等于有序区中所有的keys，则R[i]应在原有位置上
            R[0]=R[i]; j=i-1; //R[0]是哨兵，且是R[i]的副本
            do{ //从右向左在有序区R[1..i-1]中查找R[i]的插入位置
                R[j+1]=R[j]; //将关键字大于R[i].key的记录后移
                j--;
            }while(R[0].key<R[j].key); //当R[i].key≥R[j].key时终止
            R[j+1]=R[0]; //R[i]插入到正确的位置上
        } //endif
} //InsertSort
```



## 程序代码

### 【程序 41】 插入排序

```
#include <stdio.h>
#define MAX 255
int R[MAX];
void Insert_Sort(int n)
{ /* 对数组 R 中的记录 R[1..n] 按递增序进行插入排序 */
    int i,j;
    for(i=2;i<=n;i++) /* 依次插入 R[2], ..., R[n] */
        if(R[i]<R[i-1])
            /* 若 R[i] 大于等于有序区中所有的 R，则 R[i] 应在原有位置上 */
            R[0]=R[i]; j=i-1; /* R[0] 是哨兵，且是 R[i] 的副本 */
            do{ /* 从右向左在有序区 R[1.. i-1] 中查找 R[i] 的插入位置 */
                R[j+1]=R[j]; /* 将关键字大于 R[i] 的记录后移 */
                j--;
            }while(R[0]<R[j]); /* 当 R[i] ≥ R[j] 时终止 */
            R[j+1]=R[0]; /* R[i] 插入到正确的位置上 */
        }
}

main()
{
    int i,n;
    clrscr();
    puts("Please input total element number of the sequence:");
    scanf("%d",&n);
    if(n<=0 || n>MAX)
    {
        printf("n must more than 0 and less than %d.\n",MAX);
        exit(0);
    }
    puts("Please input the elements one by one:");
    for(i=1;i<=n;i++)
        scanf("%d",&R[i]);
    puts("The sequence you input is:");
    for(i=1;i<=n;i++)
        printf("%4d",R[i]);
    Insert_Sort(n);
    puts("\nThe sequence after insert_sort is:");
    for(i=1;i<=n;i++)
        printf("%4d",R[i]);
    puts("\n Press any key to quit...");
    getch();
}
```



## 归纳注释

哨兵的作用：算法中引进的附加记录 R[0] 称监视哨或哨兵（Sentinel）。哨兵有两个作用：① 进行查找（插入位置）循环之前，它保存了 R[i] 的副本，使不至于因记录后移而丢失 R[i] 的内容；

② 它的主要作用是在查找循环中“监视”下标变量  $j$  是否越界。一旦越界（即  $j=0$ ），因为  $R[0].key$  和自己比较，循环判定条件不成立使得查找循环结束，从而避免了在该循环内的每一次均要检测  $j$  是否越界（即省略了循环判定条件 “ $j \geq 1$ ”）。

实际上，一切为简化边界条件而引入的附加结点（元素）均可称为哨兵。例如单链表中的头结点实际上是一个哨兵。引入哨兵后可以使得测试查找循环条件的时间减少了一半，所以对于记录数较大的文件节约的时间就相当可观。对于类似于排序这样使用频率非常高的算法，要尽可能地减少其运行时间。所以不能把上述算法中的哨兵视为雕虫小技，而应该深刻理解并掌握这种技巧。

插入算法的时间性能分析：对于具有  $n$  个记录的文件，要进行  $n-1$  趟排序。各种状态下的时间复杂度：初始文件状态为正序、反序和无序（平均）时，时间复杂度分别为  $O(n)$ 、 $O(n^2)$  和  $O(n^2)$ 。

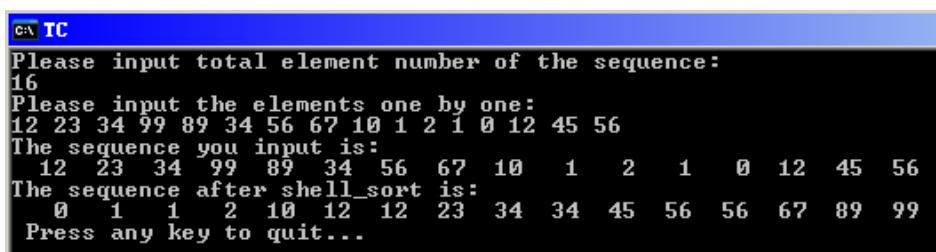
算法的空间复杂度分析：算法所需的辅助空间是一个监视哨，辅助空间复杂度  $S(n)=O(1)$ ，是一个就地排序。

直接插入排序的稳定性：直接插入排序是稳定的排序方法。

## 实例 42 希尔排序

### 实例说明

用希尔排序方法对数组进行排序。程序运行结果如图 42-1 所示。



```
TC
Please input total element number of the sequence:
16
Please input the elements one by one:
12 23 34 99 89 34 56 67 10 1 2 1 0 12 45 56
The sequence you input is:
 12 23 34 99 89 34 56 67 10 1 2 1 0 12 45 56
The sequence after shell_sort is:
 0 1 1 2 10 12 12 23 34 34 45 56 56 67 89 99
Press any key to quit...
```

图 42-1 实例 42 希尔排序程序运行结果

### 实例解析

希尔排序（Shell Sort）是插入排序的一种。希尔排序基本思想是先取一个小于  $n$  的整数  $d_1$  作为第一个增量，把文件的全部记录分成  $d_1$  个组。所有距离为  $d_1$  的倍数的记录放在同一个组中。先在各组内进行直接插入排序；然后，取第二个增量  $d_2 < d_1$  重复上述的分组和排序，直至所取的增量  $d_t=1(d_t < d_{t-1} < \dots < d_2 < d_1)$ ，即所有记录放在同一组中进行直接插入排序为止。该方法实质上是一种分组插入方法。

Shell 排序的算法描述如下：

```
void ShellPass(SeqList R, int d)
    {//希尔排序中的一趟排序, d 为当前增量
        for(i=d+1; i<=n; i++) //将 R[d+1.. n] 分别插入各组当前的有序区
            if(R[i].key<R[i-d].key){
                R[0]=R[i]; j=i-d; //R[0]只是暂存单元, 不是哨兵
                do { //查找 R[i] 的插入位置
                    R[j+d]=R[j]; //后移记录
                    j=j-d; //查找前一记录
                }while(j>0&&R[0].key<R[j].key);
```

```

        R[j+d]=R[0]; //插入 R[i]到正确的位置上
    } //endif
} //ShellPass

void ShellSort(SeqList R)
{
    int increment=n; //增量初值, 设 n>0
    do {
        increment=increment/3+1; //求下一增量
        ShellPass(R, increment); //一趟增量为 increment 的 Shell 插入排序
    }while(increment>1)
} //ShellSort

```

注意：当增量  $d=1$  时，希尔排序和插入排序基本一致，只是由于没有哨兵而在内循环中增加了一个循环判定条件“ $j>0$ ”，以防下标越界。



## 程序代码

### 【程序 42】 希尔排序

```

#include <stdio.h>
#define MAX 255
int R[MAX];
void ShellPass(int d, int n)
/* 希尔排序中的一趟排序, d 为当前增量 */
{
    int i,j;
    for(i=d+1;i<=n;i++) /* 将 R[d+1.. n] 分别插入各组当前的有序区 */
        if(R[i]<R[i-d])
        {
            R[0]=R[i];j=i-d; /* R[0] 只是暂存单元, 不是哨兵 */
            do /* 查找 R[i] 的插入位置 */
                R[j+d]=R[j];/* 后移记录 */
                j=j-d; /* 查找前一记录 */
            }while(j>0&&R[0]<R[j]);
            R[j+d]=R[0]; /* 插入 R[i] 到正确的位置上 */
        } /* endif */
    } /* end of ShellPass */

void Shell_Sort(int n)
{
    int increment=n; /* 增量初值, 设 n>0 */
    do {
        increment=increment/3+1; /* 求下一增量 */
        ShellPass(increment,n); /* 一趟增量为 increment 的 Shell 插入排序 */
    }while(increment>1);
} /* ShellSort */

void main()
{
    int i,n;
    clrscr();
    puts("Please input total element number of the sequence:");
    scanf("%d",&n);
}

```

```

if(n<=0 || n>MAX)
{
    printf("n must more than 0 and less than %d.\n",MAX);
    exit(0);
}
puts("Please input the elements one by one:");
for(i=1;i<=n;i++)
    scanf("%d",&R[i]);
puts("The sequence you input is:");
for(i=1;i<=n;i++)
    printf("%4d",R[i]);
Shell_Sort(n);
puts("\nThe sequence after shell_sort is:");
for(i=1;i<=n;i++)
    printf("%4d",R[i]);
puts("\n Press any key to quit...");
getch();
}

```



## 归纳注释

**增量序列的选择：**希尔排序的执行时间依赖于增量序列。好的增量序列的共同特征：①最后一个增量必须为1；②尽量避免序列中的值（尤其是相邻的值）互为倍数的情况。有人通过大量的实验，给出了目前较好的结果：当n较大时，比较和移动的次数约在 $n^{1.25} \sim 1.6n^{1.25}$ 之间。

希尔排序的时间性能优于直接插入排序，因为：①当文件初态基本有序时直接插入排序所需的比较和移动次数均较少。②当n值较小时，n和 $n^2$ 的差别也较小，即直接插入排序的最好时间复杂度 $O(n)$ 和最坏时间复杂度 $O(n^2)$ 差别不大。③在希尔排序开始时增量较大，分组较多，每组的记录数目少，故各组内直接插入较快，后来增量di逐渐缩小，分组数逐渐减少，而各组的记录数目逐渐增多，但由于已经按 $di-1$ 作为距离排过序，使文件较接近于有序状态，所以新的一趟排序过程也较快。因此，希尔排序在效率上较直接插入排序有较大的改进。

**稳定性：**希尔排序是不稳定的。因为两个相同关键字在排序前后的相对次序发生了变化。

# 实例 43 冒泡排序



## 实例说明

用冒泡排序方法的数组进行排序。程序运行结果如图43-1所示。

```

C:\ Command Prompt - exit
Please input total element number of the sequence:
12
Please input the elements one by one:
98 88 76 12 10 15 78 98 100 15 0 1
The sequence you input is:
 98 88 76 12 10 15 78 98 100 15 0 1
The sequence after bubble_sort is:
 0 1 10 12 15 15 76 78 88 98 98 100
Press any key to quit...

```

图43-1 实例43 冒泡排序程序运行结果



## 实例解析

交换排序的基本思想是两两比较待排序记录的关键字，发现两个记录的次序相反时即进行交换，直到没有反序的记录为止。

应用交换排序基本思想的主要排序方法有冒泡排序和快速排序。

### 冒泡排序

将被排序的记录数组  $R[1..n]$  垂直排列，每个记录  $R[i]$  看做是重量为  $R[i].key$  的气泡。根据轻气泡不能在重气泡之下的原则，从下往上扫描数组  $R$ 。凡扫描到违反本原则的轻气泡，就使其向上“飘浮”。如此反复进行，直到最后任何两个气泡都是轻者在上，重者在下为止。

(1) 初始， $R[1..n]$  为无序区。

(2) 第一趟扫描，从无序区底部向上依次比较相邻的两个气泡的重量，若发现轻者在下、重者在上，则交换二者的位置。即依次比较  $(R[n], R[n-1])$ ,  $(R[n-1], R[n-2])$ , ...,  $(R[2], R[1])$ ; 对于每对气泡  $(R[j+1], R[j])$ ，若  $R[j+1].key < R[j].key$ ，则交换  $R[j+1]$  和  $R[j]$  的内容。

第一趟扫描完毕时，“最轻”的气泡就飘浮到该区间的顶部，即关键字最小的记录被放在最高位置  $R[1]$  上。

(3) 第二趟扫描，扫描  $R[2..n]$ 。扫描完毕时，“次轻”的气泡飘浮到  $R[2]$  的位置上……最后，经过  $n-1$  趟扫描可得到有序区  $R[1..n]$

注意：第  $i$  趟扫描时， $R[1..i-1]$  和  $R[i..n]$  分别为当前的有序区和无序区。扫描仍是从无序区底部向上直至该区顶部。扫描完毕时，该区中最轻气泡飘浮到顶部位置  $R[i]$  上，结果是  $R[1..i]$  变为新的有序区。

### 冒泡排序算法

因为每一趟排序都使有序区增加了一个气泡，在经过  $n-1$  趟排序之后，有序区中就有  $n-1$  个气泡，而无序区中气泡的重量总是大于等于有序区中气泡的重量，所以整个冒泡排序过程至多需要进行  $n-1$  趟排序。

若在某一趟排序中未发现气泡位置的交换，则说明待排序的无序区中所有气泡均满足轻者在上，重者在下的原则，因此，冒泡排序过程可在此趟排序后终止。为此，在下面给出的算法中，引入一个布尔量  $exchange$ ，在每趟排序开始前，先将其置为 FALSE。若排序过程中发生了交换，则将其置为 TRUE。各趟排序结束时检查  $exchange$ ，若未曾发生过交换则终止算法，不再进行下一趟排序。

具体算法如下：

```
void BubbleSort(SeqList R)
{ //R (1..n) 是待排序的文件，采用自下向上扫描，对 R 做冒泡排序
    int i, j;
    Boolean exchange; //交换标志
    for(i=1;i<n;i++) { //最多做 n-1 趟排序
        exchange=FALSE; //本趟排序开始前，交换标志应为假
        for(j=n-1;j>=i; j--) //对当前无序区 R[i..n] 自下向上扫描
            if(R[j+1].key<R[j].key){ //交换记录
                R[0]=R[j+1]; //R[0] 不是哨兵，仅做暂存单元
                R[j+1]=R[j];
                R[j]=R[0];
                exchange=TRUE; //发生了交换，故将交换标志置为真
            }
    }
}
```

```
if(!exchange) //本趟排序未发生交换，提前终止算法
    return;
} //endfor(外循环)
} //BubbleSort
```



## 程序代码

### 【程序 43】 冒泡排序

```
#include <stdio.h>
#define MAX 255
int R[MAX];
void Bubble_Sort(int n)
{ /* R[1..n]是待排序的文件，采用自下向上扫描，对 R 做冒泡排序 */
    int i,j;
    int exchange; /* 交换标志 */
    for(i=1;i<n;i++){ /* 最多做 n-1 趟排序 */
        exchange=0; /* 本趟排序开始前，交换标志应为假 */
        for(j=n-1;j>=i;j--) /* 对当前无序区 R[i..n]自下向上扫描 */
            if(R[j+1]<R[j]){/* 交换记录 */
                R[0]=R[j+1]; /* R[0]不是哨兵，仅做暂存单元 */
                R[j+1]=R[j];
                R[j]=R[0];
                exchange=1; /* 发生了交换，故将交换标志置为真 */
            }
        if(!exchange) /* 本趟排序未发生交换，提前终止算法 */
            return;
    }
}

void main()
{
    int i,n;
    clrscr();
    puts("Please input total element number of the sequence:");
    scanf("%d",&n);
    if(n<=0||n>MAX)
    {
        printf("n must more than 0 and less than %d.\n",MAX);
        exit(0);
    }
    puts("Please input the elements one by one:");
    for(i=1;i<=n;i++)
        scanf("%d",&R[i]);
    puts("The sequence you input is:");
    for(i=1;i<=n;i++)
        printf("%4d",R[i]);
    Bubble_Sort(n);
    puts("\nThe sequence after bubble_sort is:");
    for(i=1;i<=n;i++)
        printf("%4d",R[i]);
    puts("\n Press any key to quit...");
    getch();
}
```

}



## 归纳注释

**算法的最好时间复杂度：**若文件的初始状态是正序的，一趟扫描即可完成排序。所需的关键字比较次数 C 和记录移动次数 M 均达到最小值，即  $C_{\min}=n-1$ ,  $M_{\min}=0$ 。冒泡排序最好的时间复杂度为  $O(n)$ 。

**算法的最坏时间复杂度：**若初始文件是反序的，需要进行  $n-1$  趟排序。每趟排序要进行  $n-i$  次关键字的比较 ( $1 \leq i \leq n-1$ )，且每次比较都必须移动记录 3 次。在这种情况下，比较和移动次数均达到最大值，即  $C_{\max}=n(n-1)/2=O(n^2)$ ,  $M_{\max}=3n(n-1)/2=O(n^2)$ 。冒泡排序的最坏时间复杂度为  $O(n^2)$ 。

**算法的平均时间复杂度**为  $O(n^2)$ 。虽然冒泡排序不一定要进行  $n-1$  趟，但由于它的记录移动次数较多，故平均时间性能比直接插入排序要差得多。

**算法稳定性：**冒泡排序是就地排序，且它是稳定的。

**算法改进：**上述的冒泡排序还可做如下的改进，①记住最后一次交换发生位置 lastExchange 的冒泡排序（该位置之前的相邻记录均已有序）。下一趟排序开始时， $R[1..lastExchange-1]$  是有序区， $R[lastExchange..n]$  是无序区。这样，一趟排序可能使当前有序区扩充多个记录，从而减少排序的趟数。②改变扫描方向的冒泡排序。冒泡排序具有不对称性。能一趟扫描完成排序的情况，只有最轻的气泡位于  $R[n]$  的位置，其余的气泡均已排好序，那么也只需一趟扫描就可以完成排序。如对初始关键字序列 12, 18, 42, 44, 45, 67, 94, 10 就仅需一趟扫描。需要  $n-1$  趟扫描完成排序情况，当只有最重的气泡位于  $R[1]$  的位置，其余的气泡均已排好序时，则仍需做  $n-1$  趟扫描才能完成排序。比如对初始关键字序列：94, 10, 12, 18, 42, 44, 45, 67 就需 7 趟扫描。造成不对称性的原因是每趟扫描仅能使最重气泡“下沉”一个位置，因此使位于顶端的最重气泡下沉到底部时，需做  $n-1$  趟扫描。在排序过程中交替改变扫描方向，可改进不对称性。

## 实例 44 快速排序



### 实例说明

用快速排序的方法对数组进行排序。程序运行结果如图 44-1 所示。

```

C:\TC
Please input total element number of the sequence:
15
Please input the elements one by one:
123 89 99 90 89 1 2 0 23 45 67 89 2 12 53
The sequence you input is:
123 89 99 90 89 1 2 0 23 45 67 89 2 12 53
The sequence after quick_sort is:
0 1 2 2 12 23 45 53 67 89 89 89 90 99 123
Press any key to quit...

```

图 44-1 实例 44 快速排序程序运行结果



### 实例解析

#### 快速排序 (QuickSort)

快速排序是一种划分交换排序。它采用了一种分治的策略，通常称其为分治法 (Divide-and-

ConquerMethod)。

(1) 分治法的基本思想，将原问题分解为若干个规模更小但结构与原问题相似的子问题。递归地解这些子问题，然后将这些子问题的解组合为原问题的解。

(2) 快速排序的基本思想

设当前待排序的无序区为  $R[low..high]$ ，利用分治法的基本思想如下。

① 分解。在  $R[low..high]$  中任选一个记录作为基准 (Pivot)，以此基准将当前无序区划分为左、右两个较小的子区间  $R[low..pivotpos-1]$  和  $R[pivotpos+1..high]$ ，并使左边子区间中所有记录的关键字均小于等于基准记录 (不妨记为 pivot) 的关键字 pivot.key，右边的子区间中所有记录的关键字均大于等于 pivot.key，而基准记录 pivot 则位于正确的位置 (pivotpos) 上，无需参加后续的排序。

划分的关键是要求出基准记录所在的位置 pivotpos。划分的结果可以简单地表示为 (注意  $pivot=R[pivotpos]$ )： $R[low..pivotpos-1].keys \leq R[pivotpos].key \leq R[pivotpos+1..high].keys$ ，其中  $low \leq pivotpos \leq high$ 。

② 求解。通过递归调用快速排序对左、右子区间  $R[low..pivotpos-1]$  和  $R[pivotpos+1..high]$  快速排序。

③ 组合。因为当“求解”步骤中的两个递归调用结束时，其左、右两个子区间已有序。对快速排序而言，“组合”步骤无需做什么，可看作是空操作。

### 快速排序算法

```
void QuickSort(SeqList R, int low, int high)
{ //对 R[low..high]快速排序
    int pivotpos; //划分后的基准记录的位置
    if(low<high){ //仅当区间长度大于 1 时才须排序
        pivotpos=Partition(R, low, high); //对 R[low..high]做划分
        QuickSort(R, low, pivotpos-1); //对左区间递归排序
        QuickSort(R, pivotpos+1, high); //对右区间递归排序
    }
} //QuickSort
```

为排序整个文件，调用 QuickSort(R,1,n) 即可完成对  $R[l..n]$  的排序。

### 划分算法 (Partition)

第 1 步，(初始化)设置两个指针 i 和 j，它们的初值分别为区间的下界和上界，即  $i=low, i=high$ ；选取无序区的第一个记录  $R[i]$  (即  $R[low]$ ) 作为基准记录，并将它保存在变量 pivot 中。

第 2 步，令 j 自 high 起向左扫描，直到找到第 1 个关键字小于 pivot.key 的记录  $R[j]$ ，将  $R[j]$  移至 i 所指的位置上，这相当于  $R[j]$  和基准  $R[i]$  (即 pivot) 进行了交换，使关键字小于基准关键字 pivot.key 的记录移到了基准的左边，交换后  $R[j]$  中相当于是 pivot；然后，令 i 指针自  $i+1$  位置开始向右扫描，直至找到第 1 个关键字大于 pivot.key 的记录  $R[i]$ ，将  $R[i]$  移到 i 所指的位置上，这相当于交换了  $R[i]$  和基准  $R[j]$ ，使关键字大于基准关键字的记录移到了基准的右边，交换后  $R[i]$  中又相当于存放了 pivot；接着令指针 j 自位置  $j-1$  开始向左扫描，如此交替改变扫描方向，从两端各自往中间靠拢，直至  $i=j$  时，i 便是基准 pivot 最终的位置，将 pivot 放在此位置上就完成了一次划分。

划分算法如下所示：

```
int Partition(SeqList R, int i, int j)
{ //调用 Partition(R, low, high) 时，对 R[low..high] 做划分，并返回基准记录的位置
    RecType pivot=R[i]; //用区间的第 1 个记录作为基准
    while(i<j){ //从区间两端交替向中间扫描，直至 i=j 为止
        ...
    }
}
```

```

        while(i<j&&R[j].key>=pivot.key) //pivot 相当于在位置 i 上
            j--; //从右向左扫描, 查找第 1 个关键字小于 pivot.key 的记录 R[j]
        if(i<j) //表示找到的 R[j] 的关键字小于 pivot.key
            R[i++]=R[j]; //相当于交换 R[i] 和 R[j], 交换后 i 指针加 1
        while(i<j&&R[i].key<=pivot.key) //pivot 相当于在位置 j 上
            i++; //从左向右扫描, 查找第 1 个关键字大于 pivot.key 的记录 R[i]
        if(i<j) //表示找到了 R[i], 使 R[i].key>pivot.key
            R[j--]=R[i]; //相当于交换 R[i] 和 R[j], 交换后 j 指针减 1
    } //endwhile
    R[i]=pivot; //基准记录已被最后定位
    return i;
} //partition

```



## 程序代码

### 【程序 44】 快速排序

```

#include <stdio.h>
#define MAX 255
int R[MAX];
int Partition(int i,int j)
{/* 调用 Partition(R, low, high) 时, 对 R[low..high] 做划分, 并返回基准记录的位置 */
    int pivot=R[i]; /* 用区间的第 1 条记录作为基准 */
    while(i<j){ /* 从区间两端交替向中间扫描, 直至 i=j 为止 */
        while(i<j&&R[j]>=pivot) /* pivot 相当于在位置 i 上 */
            j--; /* 从右向左扫描, 查找第 1 个关键字小于 pivot.key 的记录 R[j] */
        if(i<j) /* 表示找到的 R[j] 的关键字小于 pivot.key */
            R[i++]=R[j]; /* 相当于交换 R[i] 和 R[j], 交换后 i 指针加 1 */
        while(i<j&&R[i]<=pivot) /* pivot 相当于在位置 j 上 */
            i++; /* 从左向右扫描, 查找第 1 个关键字大于 pivot.key 的记录 R[i] */
        if(i<j) /* 表示找到了 R[i], 使 R[i].key>pivot.key */
            R[j--]=R[i]; /* 相当于交换 R[i] 和 R[j], 交换后 j 指针减 1 */
    } /* endwhile */
    R[i]=pivot; /* 基准记录已被最后定位 */
    return i;
} /* end of partition */

void Quick_Sort(int low,int high)
{ /* 对 R[low..high] 快速排序 */
    int pivotpos; /* 划分后的基准记录的位置 */
    if(low<high){/* 仅当区间长度大于 1 时才需排序 */
        pivotpos=Partition(low,high); /* 对 R[low..high] 做划分 */
        Quick_Sort(low,pivotpos-1); /* 对左区间递归排序 */
        Quick_Sort(pivotpos+1,high); /* 对右区间递归排序 */
    }
} /* end of Quick_Sort */

void main()
{
    int i,n;
    clrscr();
    puts("Please input total element number of the sequence:");
    scanf("%d",&n);
}

```

```

if(n<=0 || n>MAX)
{
    printf("n must more than 0 and less than %d.\n",MAX);
    exit(0);
}
puts("Please input the elements one by one:");
for(i=1;i<=n;i++)
    scanf("%d",&R[i]);
puts("The sequence you input is:");
for(i=1;i<=n;i++)
    printf("%4d",R[i]);
Quick_Sort(1,n);
puts("\nThe sequence after quick_sort is:");
for(i=1;i<=n;i++)
    printf("%4d",R[i]);
puts("\n Press any key to quit...");
getch();
}

```



## 归纳注释

快速排序的时间主要耗费在划分操作上，对长度为  $k$  的区间进行划分，共需  $k-1$  次关键字的比较。

**最坏时间复杂度：**最坏情况是每次划分选取的基准都是当前无序区中关键字最小（或最大）的记录，划分的结果是基准左边的子区间为空（或右边的子区间为空），而划分所得的另一个非空的子区间中记录数目，仅仅比划分前的无序区中记录个数减少一个。因此，快速排序必须做  $n-1$  次划分，第  $i$  次划分开始时区间长度为  $n-i+1$ ，所需的比较次数为  $n-i$  ( $1 \leq i \leq n-1$ )，故总的比较次数达到最大值  $C_{\max} = n(n-1)/2 = O(n^2)$ 。如果按上面给出的划分算法，每次取当前无序区的第 1 个记录为基准，那么当文件的记录已按递增序（或递减序）排列时，每次划分所取的基准就是当前无序区中关键字最小（或最大）的记录，则快速排序所需的比较次数反而最多。

**最好时间复杂度：**在最好情况下，每次划分所取的基准都是当前无序区的“中值”记录，划分的结果与基准的左、右两个无序子区间的长度大致相等。总的关键字比较次数为  $O(nlgn)$ 。

用递归树来分析最好情况下的比较次数更简单。因为每次划分后左、右子区间长度大致相等，故递归树的高度为  $O(lgn)$ ，而递归树每一层上各结点所对应的划分过程中所需要的关键字比较次数总和不超过  $n$ ，故整个排序过程所需要的关键字比较总次数  $C(n) = O(nlgn)$ 。因为快速排序的记录移动次数不大于比较的次数，所以快速排序的最坏时间复杂度应为  $O(n^2)$ ，最好时间复杂度为  $O(nlgn)$ 。

**基准关键字的选取：**在当前无序区中选取划分的基准关键字是决定算法性能的关键。  
① “三者取中”的规则，即在当前区间里，将该区间首、尾和中间位置上的关键字比较，以三者之中值所对应的记录作为基准，在划分开始前将该基准记录和该区的第 1 个记录进行交换，此后的划分过程与上面所给的 Partition 算法完全相同。  
② 取位于 low 和 high 之间的随机数  $k$  ( $low \leq k \leq high$ )，用  $R[k]$  作为基准；选取基准最好的方法是用一个随机函数产生一个位于 low 和 high 之间的随机数  $k$  ( $low \leq k \leq high$ )，用  $R[k]$  作为基准，这相当于强迫  $R[low..high]$  中的记录是随机分布的。用此方法所得到的快速排序一般称为随机的快速排序。随机的快速排序与一般的快速排序算法差别很小。但随机化后，算法的性能大大地提高了，尤其是对初始有序的文件，一般不可能导致最坏情况的发生。算法的随机化不仅仅适用于快速排序，也适用于其他需要数据随机分布的算法。

平均时间复杂度：尽管快速排序的最坏时间为  $O(n^2)$ ，但就平均性能而言，它是基于关键字比较的内部排序算法中速度最快的，快速排序亦因此而得名。它的平均时间复杂度为  $O(n \lg n)$ 。

空间复杂度：快速排序在系统内部需要一个栈来实现递归。若每次划分较为均匀，则其递归树的高度为  $O(\lg n)$ ，故递归后所需栈空间为  $O(\lg n)$ 。最坏情况下，递归树的高度为  $O(n)$ ，所需的栈空间为  $O(n)$ 。

稳定性：快速排序是非稳定的。

## 实例 45 选择排序

### 实例说明

用直接选择排序方法对数组进行排序。程序运行结果如图 45-1 所示。

```
TC
Please input total element number of the sequence:
8
Please input the elements one by one:
12 -23 566 25 89 365 -89 56 5
The sequence you input is:
12 -23 566 25 89 365 -89 56
The sequence after select_sort is:
-89 -23 12 25 56 89 365 566
Press any key to quit...
```

图 45-1 实例 45 程序运行结果

### 实例解析

选择排序（Selection Sort）的基本思想是：每一趟从待排序的记录中选出关键字最小的记录，顺序放在已排好序的子文件的最后，直到全部记录排序完毕。

常用的选择排序方法有直接选择排序和堆排序。

#### 直接选择排序（Straight Selection Sort）

直接选择排序的基本思想是  $n$  个记录的文件的直接选择排序可经过  $n-1$  趟直接选择排序得到有序结果。

① 初始状态：无序区为  $R[1..n]$ ，有序区为空。

② 第 1 趟排序

在无序区  $R[1..n]$  中选出关键字最小的记录  $R[k]$ ，将它与无序区的第 1 个记录  $R[1]$  交换，使  $R[1..1]$  和  $R[2..n]$  分别变为记录个数增加 1 个的新有序区和记录个数减少 1 个的新无序区。

③ 第  $i$  趟排序

第  $i$  趟排序开始时，当前有序区和无序区分别为  $R[1..i-1]$  和  $R[i..n]$  ( $1 \leq i \leq n-1$ )。该趟排序从当前无序区中选出关键字最小的记录  $R[k]$ ，将它与无序区的第 1 个记录  $R[i]$  交换，使  $R[1..i]$  和  $R[i+1..n]$  分别变为记录个数增加 1 个的新有序区和记录个数减少 1 个的新无序区。

这样， $n$  个记录的文件的直接选择排序即可经过  $n-1$  趟直接选择排序得到有序结果。

直接选择排序的具体算法如下：

```
void SelectSort(SeqList R)
```

```

{
    int i, j, k;
    for(i=1;i<n;i++){ //做第 i 趟排序(1≤i≤n-1)
        k=i;
        for(j=i+1;j<=n;j++) //在当前无序区 R[i..n]中选 key 最小的记录 R[k]
            if(R[j].key<R[k].key)
                k=j; //k 记下目前找到的最小关键字所在的位置
            if(k!=i){ //交换 R[i] 和 R[k]
                R[0]=R[i]; R[i]=R[k]; R[k]=R[0]; //R[0]做暂存单元
            } //endif
        } //endfor
    } //SelectSort
}

```

## 程序代码

### 【程序 45】 直接选择排序

```

#include <stdio.h>
#define MAX 255
int R[MAX];
void Select_Sort(int n)
{
    int i,j,k;
    for(i=1;i<n;i++)
    {/* 做第 i 趟排序(1≤i≤n-1) */
        k=i;
        for(j=i+1;j<=n;j++) /* 在当前无序区 R[i..n]中选 key 最小的记录 R[k] */
            if(R[j]<R[k])
                k=j; /* k 记下目前找到的最小关键字所在的位置 */
            if(k!=i)
            { /* 交换 R[i] 和 R[k] */
                R[0]=R[i]; R[i]=R[k]; R[k]=R[0]; /* R[0]做暂存单元 */
            } /* endif */
        } /* endfor */
    } /* end of Select_Sort */

void main()
{
    int i,n;
    clrscr();
    puts("Please input total element number of the sequence:");
    scanf("%d",&n);
    if(n<=0||n>MAX)
    {
        printf("n must more than 0 and less than %d.\n",MAX);
        exit(0);
    }
    puts("Please input the elements one by one:");
    for(i=1;i<=n;i++)
        scanf("%d",&R[i]);
    puts("The sequence you input is:");
    for(i=1;i<=n;i++)
        printf("%4d",R[i]);
    Select_Sort(n);
}

```

```

puts ("\nThe sequence after select_sort is:");
for(i=1;i<=n;i++)
    printf ("%4d",R[i]);
puts ("\n Press any key to quit...");
getch();
}

```



## 归纳注释

**关键字比较次数：**无论文件初始状态如何，在第*i*趟排序中选出最小关键字的记录，需做*n-i*次比较，因此，总的比较次数为  $n(n-1)/2=O(n^2)$ 。

**记录的移动次数：**当初始文件为正序时，移动次数为 0。文件初态为反序时，每趟排序均要执行交换操作，总的移动次数取最大值  $3(n-1)$ 。直接选择排序的平均时间复杂度为  $O(n^2)$ 。

直接选择排序是一个就地排序。

**稳定性分析：**直接选择排序是不稳定的，反例比如[2,2,1]。

# 实例 46 堆排序



## 实例说明

用堆排序的方法对数组进行排序。程序运行结果如图 46-1 所示。

```

ex TC
Please input total element number of the sequence:
15
Please input the elements one by one:
1 2 3 4 5 6 7 8 9 10 33 44 565 785 1001
The sequence you input is:
1 2 3 4 5 6 7 8 9 10 33 44 565 785 1001
The sequence after Big heap_sort is:
1001 785 565 44 33 10 9 8 7 6 5 4 3 2 1
Press any key to quit...

```

图 46-1 实例 46 堆排序程序运行结果



## 实例解析

### 堆排序

**堆排序定义：***n* 个关键字序列  $K_1, K_2, \dots, K_n$  称为堆，当且仅当该序列满足如下性质（简称堆性质）：① $k_i \leq K_{2i}$  且  $k_i \leq K_{2i+1}$  或② $K_i \geq K_{2i}$  且  $K_i \geq K_{2i+1}$  ( $1 \leq i \leq n$ )。

若将此序列所存储的向量  $R[1..n]$  看做是一棵完全二叉树的存储结构，则堆实质上是满足如下性质的完全二叉树：树中任一非叶结点的关键字均不大于（或不小于）其左右子结点（若存在）的关键字。

例如，关键字序列 (10, 15, 56, 25, 30, 70) 和 (70, 56, 30, 25, 15, 10) 分别满足

堆性质①和②，故它们均是堆，其对应的完全二叉树比如小根堆示例和大根堆示例如图 46-2 和 46-3 所示。

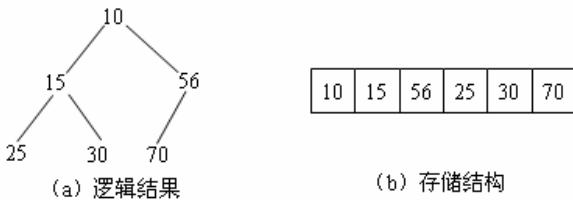


图 46-2 小根堆示例

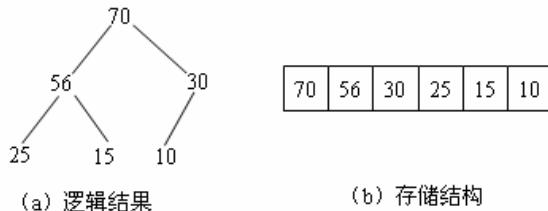


图 46-3 大根堆示例

## 大根堆和小根堆

根结点（亦称为堆顶）的关键字是堆里所有结点关键字中最小者的堆称为小根堆。

根结点（亦称为堆顶）的关键字是堆里所有结点关键字中最大者的堆称为大根堆。

注意：堆中任一子树亦是堆；以上讨论的堆实际上是二叉堆（Binary Heap），类似地可定义 k 叉堆。

## 堆排序特点

堆排序（HeapSort）是一树形选择排序。在排序过程中，将  $R[l..n]$  看成是一棵完全二叉树的顺序存储结构，利用完全二叉树中双亲结点和孩子结点之间的内在关系，在当前无序区中选择关键字最大（或最小）的记录。

## 堆排序与直接插入排序的区别

直接选择排序中，为了从  $R[1..n]$  中选出关键字最小的记录，必须进行  $n-1$  次比较，然后在  $R[2..n]$  中选出关键字最小的记录，又需要做  $n-2$  次比较。事实上，后面的  $n-2$  次比较中，有许多比较可能在前面的  $n-1$  次比较中已经做过，但由于前一趟排序时未保留这些比较结果，所以后一趟排序时又重复执行了这些比较操作。

堆排序可通过树形结构保存部分比较结果，可减少比较次数。

堆排序利用了大根堆（或小根堆）堆顶记录的关键字最大（或最小）这一特征，使得在当前无序区中选取最大（或最小）关键字的记录变得简单。

## 用大根堆排序的基本思想

- ① 先将初始文件  $R[1..n]$  建成一个大根堆，此堆为初始的无序区
- ② 再将关键字最大的记录  $R[1]$ （即堆顶）和无序区的最后一个记录  $R[n]$  交换，由此得到新的无序区  $R[1..n-1]$  和有序区  $R[n]$ ，且满足  $R[1..n-1].keys \leq R[n].key$

③ 由于交换后新的根  $R[1]$  可能违反堆性质，故应将当前无序区  $R[1..n-1]$  调整为堆。然后再次将  $R[1..n-1]$  中关键字最大的记录  $R[1]$  和该区间的最后一个记录  $R[n-1]$  交换，由此得到新的无序区  $R[1..n-2]$  和有序区  $R[n-1..n]$ ，且仍满足关系  $R[1..n-2].keys \leq R[n-1..n].keys$ ，同样要将  $R[1..n-2]$  调整为堆……直到无序区只有一个元素为止。

### 大根堆排序算法的基本操作

- ① 初始化操作：将  $R[1..n]$  构造为初始堆。
- ② 每一趟排序的基本操作：将当前无序区的堆顶记录  $R[1]$  和该区间的最后一个记录交换，然后将新的无序区调整为堆（亦称重建堆）。

注意：只需做  $n-1$  趟排序，选出较大的  $n-1$  个关键字即可使得文件递增有序。用小根堆排序与利用大根堆类似，只不过其排序结果是递减有序的。堆排序和直接选择排序相反，在任何时刻，堆排序中无序区总是在有序区之前，且有序区是在原向量的尾部由后往前逐步扩大至整个向量为止。

堆排序的算法如下：

```
void HeapSort(SeqList R)
{ //对 R[1..n]进行堆排序，用 R[0]做暂存单元
    int i;
    BuildHeap(R); //将 R[1..n]建成初始堆
    for(i=n; i>1; i--){ //对当前无序区 R[1..i]进行堆排序，共做 n-1 趟。
        R[0]=R[1]; R[1]=R[i]; R[i]=R[0]; //将堆顶和堆中最后一个记录交换
        Heapify(R, 1, i-1); //将 R[1..i-1]重新调整为堆，仅有 R[1]可能违反堆性质
    } //endfor
} //HeapSort
```

### BuildHeap 和 Heapify 函数的实现

因为构造初始堆必须使用到调整堆的操作，先讨论 Heapify 的实现。

Heapify 函数思想：每趟排序开始前  $R[1..i]$  是以  $R[1]$  为根的堆，在  $R[1]$  与  $R[i]$  交换后，新的无序区  $R[1..i-1]$  中只有  $R[1]$  的值发生了变化，故除  $R[1]$  可能违反堆性质外，其余任何结点为根的子树均是堆。因此，当被调整区间是  $R[low..high]$  时，只需调整以  $R[low]$  为根的树即可。

“筛选法”调整堆。 $R[low]$  的左、右子树（若存在）均已是堆，这两棵子树的根  $R[2low]$  和  $R[2low+1]$  分别是各自子树中关键字最大的结点。若  $R[low].key$  不小于这两个孩子结点的关键字，则  $R[low]$  未违反堆性质，以  $R[low]$  为根的树已是堆，无需调整；否则必须将  $R[low]$  和它的两个孩子结点中关键字较大者进行交换，即  $R[low]$  与  $R[large]$  ( $R[large].key = \max(R[2low].key, R[2low+1].key)$ ) 交换。交换后又可能使结点  $R[large]$  违反堆性质，同样由于该结点的两棵子树（若存在）仍然是堆，故可重复上述的调整过程，对以  $R[large]$  为根的树进行调整。此过程直至当前被调整的结点已满足堆性质，或者该结点已是叶子为止。上述过程就像过筛子一样，把较小的关键字逐层筛下去，而将较大的关键字逐层选上来。因此，有人将此方法称为“筛选法”。

具体的算法如下：

```
void Heapify(SeqList R, int s, int m)
{ //对 R[1..n]进行堆调整，用 temp 做暂存单元
    int j;
    temp=R[s];
    j=2*s;
    while (j<=m)
    {
        if (R[j]>R[j+1]&&j<m) j++;
        if (temp<R[j]) break;
```

```

        R[s]=R[j];
        s=j;
        j=j*2;
    }
    R[s]=temp;
} //endwhile
} //Heapify

```

BuildHeap 的实现：要将初始文件 R[1..n] 调整为一个大根堆，就必须将它所对应的完全二叉树中以每一结点为根的子树都调整为堆。

显然只有一个结点的树是堆，而在完全二叉树中，所有序号  $i > [n/2]$  的结点都是叶子，因此以这些结点为根的子树均已是堆。这样，只需依次将以序号为  $[n/2], [n/2]-1, \dots, 1$  的结点作为根的子树都调整为堆即可。

具体算法如下：

```

void BuildHeap(SeqList R, int n)
{
    for(int i=n/2;i>0;i--)
        Heapify(R,i,n);
}

```



## 程序代码

### 【程序 46】 堆排序

```

#include <stdio.h>
#define MAX 255
int R[MAX];

void Heapify(int s,int m)
{ /*对 R[1..n]进行堆调整，用 temp 做暂存单元 */
    int j,temp;
    temp=R[s];
    j=2*s;
    while (j<=m)
    {
        if (R[j]>R[j+1]&&j<m) j++;
        if (temp<R[j]) break;
        R[s]=R[j];
        s=j;
        j=j*2;
    }/* end of while */
    R[s]=temp;
} /* end of Heapify */

void BuildHeap(int n)
{ /*由一个无序的序列建成一个堆 */
    int i;
    for(i=n/2;i>0;i--)
        Heapify(i,n);
}

void Heap_Sort(int n)

```

```

{ /* 对 R[1..n]进行堆排序，用 R[0]做暂存单元 */
    int i;
    BuildHeap(n); /* 将 R[1-n]建成初始堆 */
    for(i=n;i>1;i--)
    { /* 对当前无序区 R[1..i]进行堆排序，共做 n-1 趟。 */
        R[0]=R[1]; R[1]=R[i];R[i]=R[0]; /* 将堆顶和堆中最后一个记录交换 */
        Heapify(1,i-1); /* 将 R[1..i-1]重新调整为堆，仅有 R[1]可能违反堆性质 */
    } /* end of for */
} /* end of Heap_Sort */
void main()
{
    int i,n;
    clrscr();
    puts("Please input total element number of the sequence:");
    scanf("%d",&n);
    if(n<=0||n>MAX)
    {
        printf("n must more than 0 and less than %d.\n",MAX);
        exit(0);
    }
    puts("Please input the elements one by one:");
    for(i=1;i<=n;i++)
        scanf("%d",&R[i]);
    puts("The sequence you input is:");
    for(i=1;i<=n;i++)
        printf("%4d",R[i]);
    Heap_Sort(n);
    puts("\nThe sequence after Big heap_sort is:");
    for(i=1;i<=n;i++)
        printf("%4d",R[i]);
    puts("\n Press any key to quit...");
    getch();
}

```



## 归纳注释

堆排序的时间主要由建立初始堆和反复重建堆这两部分的时间开销构成，它们均是通过调用 Heapify 实现的。

堆排序的最坏时间复杂度为  $O(n \lg n)$ 。堆排序的平均性能较接近于最坏性能。

由于建初始堆所需的比较次数较多，所以堆排序不适于记录数较少的文件。

堆排序是就地排序，辅助空间为  $O(1)$ ，堆排序是不稳定的排序方法。

# 实例 47 归并排序



## 实例说明

用归并排序的方法对数组进行排序。程序运行结果如图 47-1 所示。

```

C:\TC
Please input total element number of the sequence:
15
Please input the elements one by one:
99 88 77 66 55 44 33 22 11 0 1 2 77 100 5
The sequence you input is:
 99 88 77 66 55 44 33 22 11 0 1 2 77 100 5
The sequence after merge_sortDC is:
 0 1 2 5 11 22 33 44 55 66 77 77 88 99 100
Press any key to quit...

```

图 47-1 实例 47 归并排序程序运行结果

## 实例解析

归并排序 (Merge Sort) 是利用“归并”技术来进行排序。归并是指将若干已排序的子文件合并成一个有序的文件。

### 两路归并算法

算法基本思路：设两个有序的子文件（相当于输入堆）放在同一向量中相邻的位置上， $R[low..m]$ ,  $R[m+1..high]$ , 先将它们合并到一个局部的暂存向量  $R1$ （相当于输出堆）中，待合并完成后将  $R1$  复制回  $R[low..high]$  中。

① 合并过程，设置  $i$ ,  $j$  和  $p$  3 个指针，其初值分别指向这 3 个记录区的起始位置。合并时依次比较  $R[i]$  和  $R[j]$  的关键字，取关键字较小的记录复制到  $R1[p]$  中，然后将被复制记录的指针  $i$  或  $j$  加 1，以及指向复制位置的指针  $p$  加 1。

重复这一过程直至两个输入的子文件有一个已全部复制完毕（不妨称其为空），此时将另一非空的子文件中剩余记录依次复制到  $R1$  中即可。

② 动态申请  $R1$ ，因为申请的空间可能很大，故需加入申请空间是否成功的处理。

归并算法如下：

```

void Merge(SeqList R, int low, int m, int high)
    {//将两个有序的子文件 R[low..m]和 R[m+1..high]归并成一个有序的子文件 R[low..high]
    int i=low, j=m+1, p=0; //置初始值
    RecType *R1; //R1 是局部向量，若 p 定义为此类型指针速度更快
    R1=(RecType *)malloc((high-low+1)*sizeof(RecType));
    if(! R1) //申请空间失败
        Error("Insufficient memory available!");
    while(i<=m&&j<=high) //两子文件非空时取其小者输出到 R1[p]上
        R1[p++]=(R[i].key<=R[j].key)?R[i++]:R[j++];
    while(i<=m) //若第 1 个子文件非空，则复制剩余记录到 R1 中
        R1[p++]=R[i++];
    while(j<=high) //若第 2 个子文件非空，则复制剩余记录到 R1 中
        R1[p++]=R[j++];
    for(p=0, i=low; i<=high; p++, i++)
        R[i]=R1[p]; //归并完成后将结果复制回 R[low..high]
    } //Merge

```

### 归并排序

归并排序有两种实现方法：自底向上和自顶向下。

(1) 自底向上的基本思想，第 1 趟归并排序时，将待排序的文件  $R[1..n]$  看做是  $n$  个长度为 1 的有序子文件，将这些子文件两两归并，若  $n$  为偶数，则得到  $\lceil \frac{n}{2} \rceil$ （表示大于等于  $\frac{n}{2}$  的最小整数）个

长度为 2 的有序子文件；若  $n$  为奇数，则最后一个子文件轮空（不参与归并）。故本趟归并完成后，前  $\lceil \lg n \rceil$  个有序子文件长度为 2，但最后一个子文件长度仍为 1；第 2 趟归并则是将第 1 趟归并所得到的  $\lceil \lg n \rceil$  个有序的子文件两两归并，如此反复，直到最后得到一个长度为  $n$  的有序文件为止。

上述的每次归并操作，均是将两个有序的子文件合并成一个有序的子文件，故称其为“二路归并排序”。类似地有  $k(k > 2)$  路归并排序。

(2) 一趟归并算法，在某趟归并中，设各子文件长度为  $length$ （最后一个子文件的长度可能小于  $length$ ），则归并前  $R[1..n]$  中共有  $\lceil \lg n \rceil$  个有序的子文件： $R[1..length], R[length+1..2length] \dots$ 。

注意：调用归并操作将相邻的一对子文件进行归并时，必须对子文件的个数（可能是奇数）以及最后一个子文件的长度小于  $length$  这两种特殊情况进行处理：① 若子文件个数为奇数，则最后一个子文件无需和其他子文件归并（即本趟轮空）；② 若子文件个数为偶数，则要注意最后一对子文件中后一子文件的区间上界是  $n$ 。

具体算法如下：

```
void MergePass(SeqList R, int length)
{ //对 R[1..n] 做一趟归并排序
    int i;
    for(i=1; i+2*length-1<=n; i=i+2*length)
        Merge(R, i, i+length-1, i+2*length-1);
        //归并长度为 length 的两个相邻子文件
    if(i+length-1<n) //尚有两个子文件，其中后一个长度小于 length
        Merge(R, i, i+length-1, n); //归并最后两个子文件
    //注意：若 i≤n 且 n≤i+length 时，则剩余一个子文件轮空，无需归并
}
```

二路归并排序算法如下：

```
void MergeSort(SeqList R)
{//采用自底向上的方法，对 R[1..n] 进行二路归并排序
    int length;
    for(length=1; length<n; length*=2) //做 n 趟归并
        MergePass(R, length); //有序段长度≥n 时终止
}
```

注意：自底向上的归并排序算法虽然效率较高，但可读性较差。

(3) 自顶向下的方法采用分治法进行自顶向下的算法设计，形式更为简洁。

设归并排序的当前区间是  $R[low..high]$ ，分治法的 3 个步骤如下。

- ① 分解，将当前区间一分为二，即求分裂点。
- ② 求解，递归地对两个子区间  $R[low..mid]$  和  $R[mid+1..high]$  进行归并排序。
- ③ 组合，将已排序的两个子区间  $R[low..mid]$  和  $R[mid+1..high]$  归并为一个有序的区间  $R[low..high]$ 。

递归的终结条件为子区间长度为 1（一个记录自然有序）。

分治法的具体算法如下。

```
void MergeSortDC(SeqList R, int low, int high)
{//用分治法对 R[low..high] 进行二路归并排序
    int mid;
    if(low<high){ //区间长度大于 1
        mid=(low+high)/2; //分解
        MergeSortDC(R, low, mid); //递归地对 R[low..mid] 排序
        MergeSortDC(R, mid+1, high); //递归地对 R[mid+1..high] 排序
        Merge(R, low, mid, high); //组合，将两个有序区归并为一个有序区
    }
}
```



## 程序代码

### 【程序 47】 归并排序

```
#include <stdio.h>
#define MAX 255
int R[MAX];

void Merge(int low,int m,int high)
{ /* 将两个有序的子文件 R[low..m] 和 R[m+1..high] 归并成一个有序的子文件 R[low..high] */
    int i=low,j=m+1,p=0; /* 置初始值 */
    int *R1; /* R1 是局部向量, 若 p 定义为此类型指针速度更快 */
    R1=(int *)malloc((high-low+1)*sizeof(int));
    if(!R1) /* 申请空间失败 */
    {
        puts("Insufficient memory available!");
        return;
    }
    while(i<=m&&j<=high) /* 两子文件非空时取其小者输出到 R1[p] 上 */
        R1[p++]=(R[i]<=R[j])?R[i++]:R[j++];
    while(i<=m) /* 若第 1 个子文件非空, 则复制剩余记录到 R1 中 */
        R1[p++]=R[i++];
    while(j<=high) /* 若第 2 个子文件非空, 则复制剩余记录到 R1 中 */
        R1[p++]=R[j++];
    for(p=0,i=low;i<=high;p++,i++)
        R[i]=R1[p]; /* 归并完成后将结果复制回 R[low..high] */
} /* end of Merge */

void Merge_SortDC(int low,int high)
{ /* 用分治法对 R[low..high] 进行二路归并排序 */
    int mid;
    if(low<high)
    { /* 区间长度大于 1 */
        mid=(low+high)/2; /* 分解 */
        Merge_SortDC(low,mid); /* 递归地对 R[low..mid] 排序 */
        Merge_SortDC(mid+1,high); /* 递归地对 R[mid+1..high] 排序 */
        Merge(low,mid,high); /* 组合, 将两个有序区归并为一个有序区 */
    }
} /* end of Merge_SortDC */

void main()
{
    int i,n;
    clrscr();
    puts("Please input total element number of the sequence:");
    scanf("%d",&n);
    if(n<=0||n>MAX)
    {
        printf("n must more than 0 and less than %d.\n",MAX);
        exit(0);
    }
}
```

```

puts("Please input the elements one by one:");
for(i=1;i<=n;i++)
    scanf("%d",&R[i]);
puts("The sequence you input is:");
for(i=1;i<=n;i++)
    printf("%4d",R[i]);
Merge_SortDC(1,n);
puts("\nThe sequence after merge_sortDC is:");
for(i=1;i<=n;i++)
    printf("%4d",R[i]);
puts("\n Press any key to quit...");
getch();
}

```



## 归纳注释

**稳定性:** 归并排序是一种稳定的排序。

**存储结构要求:** 可用顺序存储结构, 也易于在链表上实现。

**时间复杂度:** 对长度为  $n$  的文件, 需进行  $\left\lceil \frac{n}{2} \right\rceil$  趟二路归并, 每趟归并的时间为  $O(n)$ , 故其时

间复杂度无论是在最好情况下还是在最坏情况下均是  $O(n \lg n)$ 。

**空间复杂度:** 需要一个辅助向量来暂存有序子文件归并的结果, 故其辅助空间复杂度为  $O(n)$ , 显然它不是就地排序。

若用单链表做存储结构, 则可以很容易给出就地的归并排序。

# 实例 48 基数排序



## 实例说明

用基数排序的方法对数组进行排序。程序运行结果如图 48-1 所示。

```

C:\TC
Input the records ('0' to end input):
123 34 65 7 1 567 6578 8976 34 23 65 78 90 78 0
The sequence you input is:
123 34 65 7 1 567 6578 8976 34 23 65 78 90 78
-----
-----
-----
-----
The sequence after radix_sort is:
1 7 23 34 34 65 65 78 78 90 123 567 6578 8976
Press any key to quit...
-
```

图 48-1 实例 48 基数排序程序运行结果



## 实例解析

**分配排序的基本思想:** 排序过程无需比较关键字, 而是通过“分配”和“收集”过程来实现。

它们的时间复杂度可达到线性阶  $O(n)$ 。

分配排序包括箱排序和基数排序。

### 箱排序 (Bin Sort)

箱排序也称桶排序 (Bucket Sort)，其基本思想是设置若干个箱子，依次扫描待排序的记录  $R[0], R[1], \dots, R[n-1]$ ，把关键字等于  $k$  的记录全都装入到第  $k$  个箱子里 (分配)，然后按序号依次将各非空的箱子首尾连接起来 (收集)。

例如，要将一副混洗的 52 张扑克牌按点数  $A < 2 < \dots < J < Q < K$  排序，需设置 13 个“箱子”，排序时依次将每张牌按点数放入相应的箱子里，然后依次将这些箱子首尾相接，就得到了按点数递增序排列的一副牌。

箱排序中，箱子的个数取决于关键字的取值范围。若  $R[0..n-1]$  中关键字的取值范围是 0 到  $m-1$  的整数，则必须设置  $m$  个箱子。因此箱排序要求关键字的类型是有限类型，否则可能要无限个箱子。

箱子的类型应设计成链表，因为一般情况下每个箱子中存放多少个关键字相同的记录是无法预料的。

为保证排序是稳定的，分配过程中装箱及收集过程中的连接必须按先进先出原则进行。

#### (1) 实现方法一

每个箱子设为一个链队列。当一条记录装入某箱子时，应做入队操作将其插入该箱子尾部；而收集过程则是对箱子做出队操作，依次将出队的记录放到输出序列中。

#### (2) 实现方法二

若输入的待排序记录是以链表形式给出时，出队操作可简化为将整个箱子链表链接到输出链表的尾部。这只需要修改输出链表的尾结点中的指针域，令其指向箱子链表的头，然后修改输出链表的尾指针，令其指向箱子链表的尾即可。

分配过程的时间是  $O(n)$ ；收集过程的时间为  $O(m)$  (采用链表来存储输入的待排序记录) 或  $O(m+n)$ 。因此，箱排序的时间为  $O(m+n)$ 。若箱子个数  $m$  的数量级为  $O(n)$ ，则箱排序的时间是线性的，即  $O(n)$ 。

注意：箱排序实用价值不大，仅适用于作为基数排序的一个中间步骤。

### 基数排序

基数排序 (Radix Sort) 是对箱排序的改进和推广。

文件中任一记录  $R[i]$  的关键字均由  $d$  个分量  $k_i^0 k_i^1 \dots k_i^{d-1}$  构成。若这  $d$  个分量中每个分量都是一个独立的关键字，则文件是多关键字的 (如扑克牌有两个关键字，点数和花色)；否则文件是单关键字的  $k_i^j$  ( $0 \leq j < d$ ) 只不过是关键字中其中的一位 (如字符串、十进制整数等)。

多关键字中的每个关键字的取值范围一般不同。如扑克牌的花色取值只有 4 种，而点数则有 13 种。单关键字中的每位一般取值范围相同。

设单关键字的每个分量的取值范围均是  $C_0 \leq k_j \leq C_{rd-1}$  ( $0 \leq j < rd$ )，则可能的取值个数  $rd$  称为基数。基数的选择和关键字的分解因关键字的类型而异。

(1) 若关键字是十进制整数，则按个、十等位进行分解，基数  $rd=10$ ， $C_0=0$ ， $C_9=9$ ， $d$  为最长整数的位数。

(2) 若关键字是小写的英文字母，则  $rd=26$ ， $C_0='a'$ ， $C_{25}='z'$ ， $d$  为字符串的最大长度。

基数排序的基本思想是：从低位到高位依次对  $K^j$  ( $j=d-1, d-2, \dots, 0$ ) 进行箱排序。在  $d$  趟箱排序中，所需的箱子数就是基数  $rd$ ，这就是“基数排序”名称的由来。

要保证基数排序是正确的，就必须保证除第一趟外各趟箱排序是稳定的。

算法描述如下：

```
/* 基数排序的代码是很简单的、下面的过程假设长度为 n 的数组 A 中的每个元素都有 d 位数字，其中第 1 位是最低的，第 d 位是最高位。 */
void Radix_Sort(SeqList R, int d);
{
    int i;
    for(i=1;i<=d;i++)
        使用一种稳定的排序方法来对数组 R 按数字 i 进行排序;
} /* end of Radix_Sort */
```



## 程序代码

### 【程序 48】 基数排序

```
#include "stdio.h"
#include "conio.h"
#include "stdlib.h"
#define MAX 5
typedef struct node
{
    int k;
    struct node *next;
} *lnode;
lnode my_input(int *d)
{
    lnode head,temp,terminal;
    char s[MAX+1];
    printf("Input the records ('0' to end input):\n");
    scanf("%s",s);
    head=NULL;
    *d=0;
    terminal=NULL;
    while(s[0]!='0')
    {
        temp=(lnode)malloc(sizeof(struct node));
        if (strlen(s)>*d)
            *d=strlen(s);
        temp->k=atoi(s);
        if (head==NULL)
        {
            head=temp;
            terminal=temp;
        }
        else
        {
            terminal->next=temp;
            terminal=temp;
        }
        scanf("%s",s);
    }
    terminal->next=NULL;

    return head;
}
void my_output(lnode h)
```

```

{
    lnode t=h;
    printf("\n");
    while (h!=NULL)
    {
        printf("%d ",h->k);
        h=h->next;
    }
    h=t;
    /* getch(); */
}

lnode Radix_Sort(lnode head,int d)
{
    lnode p,q,h,t;
    int i,j,x,radix=1;
    h=(lnode)malloc(10*sizeof(struct node));
    t=(lnode)malloc(10*sizeof(struct node));
    for (i=d;i>=1;i--)
    {
        for (j=0;j<=9;j++)
        {
            h[j].next=NULL;
            t[j].next=NULL;
        }
        p=head;
        while (p!=NULL)
        {
            x=((p->k)/radix)%10;
            if (h[x].next==NULL)
            {
                h[x].next=p;
                t[x].next=p;
            }
            else
            {
                q=t[x].next;
                q->next=p;
                t[x].next=p;
            }
            p=p->next;
        }

        j=0;
        while (h[j].next==NULL)
        j++;
        head=h[j].next;
        q=t[j].next;
        for (x=j+1;x<=9;x++)
        if (h[x].next!=NULL)
        {
            q->next=h[x].next;
            q=t[x].next;
        }
        q->next=NULL;
        radix*=10;
    }
}

```

```

printf("\n-----\n");
}
return head;
}
void my_free(lnode h)
{
    lnode temp=h;
    while (temp)
    {
        h=temp->next;
        free(temp);
        temp=h;
    }
}
void main()
{
    lnode h;
    int d;
    clrscr();
    h=my_input(&d);
    puts("The sequence you input is:");
    my_output(h);
    h=Radix_Sort(h,d);
    puts("\nThe sequence after radix_sort is:");
    my_output(h);
    my_free(h);
    puts("\n Press any key to quit...");
    getch();
}

```



## 归纳注释

若排序文件不是以数组 R 形式给出，而是以单链表形式给出（此时称为链式的基数排序），则可通过修改出队和入队函数使表示箱子的链队列无需分配结点空间，而使用原链表的结点空间。入队出队操作亦无需移动记录而仅需修改指针。这样以来节省了一定的时间和空间，但算法要复杂得多，且时空复杂度就其数量级而言并未得到改观。

基数排序的时间是线性的（即  $O(n)$ ）。

基数排序所需的辅助存储空间为  $O(n+rd)$ 。

基数排序是稳定的。

# 实例 49 顺序表插入和删除



## 实例说明

通过顺序表的初始化、添加节点、删除节点等操作学习顺序表的用法。程序运行结果如图 49-1 所示。

```
ca tc
Please input the size of the list:
10
Please input the elements of the list one by one:
10 20 30 40 50 60 70 80 90 100
Now the list is:
10 20 30 40 50 60 70 80 90 100
Please input the position of the element you want to delete:
7
Now the list is:
10 20 30 40 50 60 70 90 100
Please input the element you want to insert:
88
Please input the position of the element(<= 8>):7
Now the list is:
10 20 30 40 50 60 70 88 90 100

Press any key to quit...
```

图 49-1 实例 49 程序运行结果

## 实例解析

顺序表是各种数据结构中最简单却是应用很广的一种形式。学生成绩的序列或者按顺序排列的多种产品属性数据都可以看成是顺序表。简而言之，顺序表就是一个广义的一维数组，这个数组的元素可以是包含基本数据类型或者复杂数据类型在内的任何数据。在实际内存存储中顺序表的元素是连续放置的，所以可以根据某个元素在顺序表中的序号来直接访问这个元素，也就是说顺序表是可以随机访问（Random Access）的。

## 程序代码

### 【程序 49】 顺序表的操作

```
#define ListSize 100/* 假定表空间大小为 100 */
#include <stdio.h>
#include <stdlib.h>
void Error(char * message)
{
printf("错误:%s\n",message);
exit(1);
}
struct Seqlist{
int data[ListSize];/* 向量 data 用于存放表结点 */
int length; /* 当前的表长度 */
};
/* 以上为定义表结构 */

/* -----以下为两个主要算法----- */
void InsertList(struct Seqlist *L, int x, int i)
{
/* 将新结点 x 插入 L 所指的顺序表的第 i 个结点 ai 的位置上 */
int j;
if ( i < 0 || i > L->length )
Error("position error");/* 非法位置，退出 */
if ( L->length>=ListSize )
Error("overflow");
for ( j=L->length-1 ; j >= i ; j --)
```

```

L->data[j+1]=L->data [j];
L->data[i]=x ;
L->length++ ;
}

void DeleteList ( struct Seqlist *L, int i )
{/* 从 L 所指的顺序表中删除第 i 个结点 ai */
int j;
if ( i< 0 || i > L-> length-1)
Error( " position error" ) ;
for ( j = i+1 ; j < L-> length ; j++ )
    L->data [ j-1 ]=L->data [ j ]; /* 结点前移 */
L-> length-- ; /* 表长减小 */
}
/* =====以下代码对算法验证===== */
void Initlist(struct Seqlist *L)
{
    L->length=0;
}
/* 显示顺序表的内容 */
void DisplayList(struct Seqlist *L)
{
    int i;
    puts("Now the list is:");
    for(i=0;i<L->length;i++)
        printf("%d ",L->data[i]);
    printf("\n");
    return;
}
void main()
{
    struct Seqlist *SEQA;
    int i,n,t;
    SEQA = (struct Seqlist *)malloc(sizeof(struct Seqlist));
    Initlist(SEQA);
    clrscr();
    puts("Please input the size of the list:");
    scanf("%d",&n);
    puts("Please input the elements of the list one by one:");
    for (i=0;i<n;i++)
    {
        scanf("%d",&t);
        InsertList (SEQA,t,i);
    }
    DisplayList(SEQA);
    puts("Please input the position of the element you want to delete:");
    scanf("%d",&t);
    DeleteList (SEQA,t);
    DisplayList(SEQA);
    puts("Please input the element you want to insert:");
    scanf("%d",&t);
    printf("Please input the position of the element( <= %d):",SEQA->length-1);
    scanf("%d",&i);
    InsertList(SEQA,t,i);
}

```

```
DisplayList(SEQA);
puts("\n Press any key to quit...");
getch();
}
```



## 归纳注释

程序首先调用子函数 InitList()对顺序表进行初始化，要求用户输入表的大小，再依次输入表的结点元素。通过循环调用 InsertList()依次将元素插入到顺序表中，接着调用函数 DeleteList()删去顺序表的一个元素，调用 InsertList()插入一个元素，使用 DisplayList()函数循环地把顺序表中的元素逐个输出。

# 实例 50 链表操作



## 实例说明

通过链表的各种操作学习链表的概念、结构体的应用以及指针在链表中的应用。程序运行结果如图 50-1~图 50-6 所示。

```
ca tc
simple linklist realization of c
=====
[1] create linklist
[2] search
[3] insert
[4] delete
[5] print
[6] exit
if no list exist,create first
=====
Please input your choose<1-6>: _
```

图 50-1 实例 50 程序主界面

```
ca tc
Please input the number of linklist:
5
please input 1 student's name: Kate
please input 2 student's name: Mike
please input 3 student's name: John
please input 4 student's name: Herry
please input 5 student's name: Jenny
Linklist created successfully!
Press any key to return...
```

图 50-2 创建链表

```
ca tc
Input the student's name which you want to find:
Mike
The stud name you want to find is:Mike
Press any key to return...
```

图 50-3 查找记录

```
please input the student's name: Holly
Now the link list is:
Holly Kate Mike John Herry Jenny
Press any key to return....
```

图 50-4 插入记录

```
Now the link list is:
Holly Kate Mike John Herry Jenny
Input the student's name which you want to delete:
John
Now the link list is:
Holly Kate Mike Herry Jenny
Delete successfully! Press any key to return...
```

图 50-5 删除记录

```
Thank you for your using!
Press any key to quit...
```

图 50-6 退出程序

## 实例解析

程序通过结构体数据类型模拟了链表的结构，并使用指针实现了对链表的各项操作。

链接方式存储的线性表简称为链表（Linked List）。链表的存储说明如下。

(1) 用一组任意的存储单元来存放线性表的结点（这组存储单元既可以是连续的，也可以是不连续的）。

(2) 因为链表中结点的逻辑次序和物理次序不一定相同，所以为了能正确表示结点间的逻辑关系，在存储每个结点值的同时，还必须存储指示其后继结点的地址信息。这个地址信息称为指针（pointer）或链（link）。



## 程序代码

### 【程序 50】 链表的操作

```
#include <stdio.h>
#define N 10
typedef struct node
{
    char name[20];
    struct node *link;
}stud;
... /* 其他操作函数见光盘 */
main()
{
    int choose;
    stud *head,*searchpoint,*forepoint;
    char fullname[20];
```

```

while(1)
{
    menu();
    scanf("%d",&choose);
    switch(choose)
    {
        case 1:
            clrscr();
            head=creat();
            puts("Linklist created successfully! \nPress any key to return... ");
            getch();
            break;
        case 2:
            clrscr();
            printf("Input the student's name which you want to find:\n");
            scanf("%s",fullname);
            searchpoint=search(head,fullname);
            printf("The stud name you want to find is:%s",*&searchpoint->name);
            printf("\nPress any key to return... ");
            getchar();
            getchar();
            break;
        case 3:
            clrscr();
            insert(head);
            print(head);
            printf("\nPress any key to return... ");
            getchar();getchar();
            break;
        case 4:
            clrscr();
            print(head);
            printf("\nInput the student's name which you want to delete:\n");
            scanf("%s",fullname);
            searchpoint=search(head,fullname);
            forepoint=search2(head,fullname);
            del(forepoint,searchpoint);
            print(head);
            puts("\nDelete successfully! Press any key to return... ");
            getchar();
            getchar();
            break;
        case 5:print(head);
            printf("\nPress any key to return... ");
            getchar();getchar();
            break;
        case 6:quit();
            break;
        default:
            clrscr();
            printf("Illegal letter! Press any key to return... ");
            menu();
            getchar();
    }
}

```

```
}
```



## 归纳注释

程序运行时，首先调用创建函数 creat()，根据给定的长度建立了一个关于学生信息的链表；然后通过插入函数 InsertList()将指定的学生信息从链表的头部插入到链表中；随后要求输入要从链表中删除的学生信息，调用 search() 函数查询链表中是否有指定的学生信息，如果找到就调用删除子函数 Deletelist() 删除链表中的相关项；为了查看操作的效果，可以随时选择打印输出链表；最后主函数调用退出函数 Quit() 退出整个程序。

# 实例 51 双链表



## 实例说明

本实例实现通过双链表结构来查找学生的信息。程序运行时，要求输入学生信息，按顺序显示出来，并可按姓名查找。程序运行结果如图 51-1 所示。

```
ca tc
Please input the size of the list:
6
Please input the 1 man's name: Helloy
Please input the 2 man's name: Mikey
Please input the 3 man's name: Johny
Please input the 4 man's name: Kate
Please input the 5 man's name: Herry
Please input the 6 man's name: Jenny

Now the double list is:
Helloy Mikey Johny Herry Jenny

Please input the name which you want to find:
Kate
the name you want to find is:Kate

Now the double list is:
Helloy Mikey Johny Herry Jenny

Press any key to quit...
```

图 51-1 实例 51 程序运行结果



## 实例解析

双（向）链表中有两条方向不同的链，即每个结点中除 next 域存放后继结点地址外，还增加一个指向其直接前趋的指针域 prior。这样在查找过程中，当指针处在中间某个结点时不仅可以像单链表那样向后查找，而且可以返回之前找过的结点。这样的结构对于需要无规律反复读取链表元素的操作是很有用的。



## 程序代码

### 【程序 51】 双链表

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

```

#define N 10
typedef struct node
{
    char name[20];
    struct node *llink,*rlink;
}stud; /*双链表的结构定义*/
/*双链表的创建*/
stud * creat(int n)
{
    stud *p,*h,*s;
    int i;
    if((h=(stud *)malloc(sizeof(stud)))==NULL)
    {
        printf("cannot find space!\n");
        exit(0);
    }
    h->name[0]='\0';
    h->llink=NULL;
    h->rlink=NULL;
    p=h;
    for(i=0;i<n;i++)
    {
        if((s= (stud *) malloc(sizeof(stud)))==NULL)
        {
            printf("cannot find space!\n");
            exit(0);
        }
        p->rlink=s;
        printf("Please input the %d man's name: ",i+1);
        scanf("%s",s->name);
        s->llink=p;
        s->rlink=NULL;
        p=s;
    }
    h->llink=s;
    p->rlink=h;
    return(h);
} /* 其他函数见光盘*/

```



## 归纳注释

程序运行后，首先在创建过程中输入链表所需信息；接着调用子函数 `print()` 把双链表的信息打印输出；然后根据用户输入的学生姓名调用子函数 `search()` 在建成的链表中查找对应节点。如果这个节点存在，就调用删除子函数 `delete()` 进行删除；最后再次调用输出子函数 `print()` 将修改后的双链表信息打印输出。

# 实例 52 二叉树遍历



## 实例说明

实现二叉树的建立和遍历。程序运行结果如图 52-1 所示。

```
ca tc
Please initialize the TREE!
Input 1 DATA: a
This Address is:6336, Data is:a, Left Pointer is:0, Right Pointer is: 0
Input 2 DATA: b
This Address is:6356, Data is:b, Left Pointer is:0, Right Pointer is: 0
Input 3 DATA: ^
Input 4 DATA: c
This Address is:6376, Data is:c, Left Pointer is:0, Right Pointer is: 0
Input 5 DATA: ^
Input 6 DATA: d
This Address is:6416, Data is:d, Left Pointer is:0, Right Pointer is: 0
Input 7 DATA: ^
Input 8 DATA: ^
Input 9 DATA: ^

This is TREE Struct:
Address: 6336, Data: a, Left Pointer: 6356, Right Pointer: 0
Address: 6356, Data: b, Left Pointer: 0, Right Pointer: 6376
Address: 6376, Data: c, Left Pointer: 0, Right Pointer: 6416
Address: 6416, Data: d, Left Pointer: 0, Right Pointer: 0
This TREE has 1 leaves.
High of The TREE is: 4

Press any key to quit...
```

图 52-1 实例 52 运行结果



## 实例解析

树结构的形式在客观世界中是大量存在的，例如家谱、行政组织机构都可用树结构形象地表示。

树结构在计算机领域中也有着广泛的应用。例如在编译程序中，可以用树来表示源程序的语法结构；在数据库系统中，可以用树来组织信息；在分析算法的行为时，可以用树来描述其执行过程。

二叉树（Binary Tree）是树形结构的一个特殊类型。许多实际问题抽象出来的数据往往是二叉树的形式，即使是一般的树也能简单地转换为二叉树，而且二叉树的存储结构及其算法都较为简单，因此二叉树显得特别重要。

二叉树是  $n(n \geq 0)$  个结点的一个有限集，它或者是空集( $n=0$ )，或者由一个根结点及两棵互不相交的、分别称做这个根的左子树和右子树的二叉树组成。对二叉树上的节点访问有 3 种顺序，即先序遍历、中序遍历和后序遍历。这 3 种遍历方法都是通过递归实现的，具体的算法可参考有关的书籍。



## 程序代码

### 【程序 52】 二叉树的遍历

```
#include <stdio.h>
typedef struct bitnode
{ /* 定义二叉树 */
    char data;
    struct bitnode *lchild, *rchild;
}bitnode, *bitree;
void visit(e) /* 访问树枝 */
bitnode *e;
{
    printf(" Address: %o, Data: %c, Left Pointer: %o, Right Pointer:
%o\n", e, e->data, e->lchild, e->rchild);
```

```
}

void preordertraverse(t) /* 遍历二叉树 */
bitnode *t;
{   if(t)
    {   visit(t);
        preordertraverse(t->lchild);
        preordertraverse(t->rchild);
        return ;
    }else return ;
}
.../* 其他程序见光盘 */
```



## 归纳注释

在程序中，首先主函数通过调用子函数 creatbitree() 创建了一个二叉树；接着对该二叉树进行了中序遍历（调用子函数 preordertraverse()），输出其各个结点的内容；然后调用子函数 countleaf() 对该二叉树的叶子结点进行了计数，从而实现了对二叉树的基本操作。有兴趣的读者可以模仿此例编写出二叉树的先序遍历和后序遍历程序。

# 实例 53 浮点数转换为字符串



## 实例说明

本实例通过数字金额转换为大写金额的程序来介绍将浮点数转换为字符串的方法。程序运行结果如图 53-1 所示（此程序是用 Visual C++6.0 编译器编译运行的结果）。

The screenshot shows a terminal window titled "E:\meyy\Debug\53.exe". The program output is as follows:

```
*****
*          数字金额转换为大写金额程序 Ver.1.0
*
*          By RZLIN
*
*          请输入要转换的金额:
*          1234567.89
*          您输入的金额为: 1234567.89。
*
*          转换为大写金额是: 壹佰贰拾叁万肆仟伍佰陆拾柒元捌角玖分正
*
*          请按任意键退出...
*****
```

图 53-1 实例 53 运行结果



## 实例解析

首先将用户输入的金额转换成“分”，即扩大 100 倍，然后逐位判断是否每一位为零。在不是零的情况下，找出对应的货币单位和数量，从而生成相应的字符串。



## 程序代码

【程序 53】 浮点数转换为字符串（数字金额转换为大写金额）

//本实例源码参见光盘



## 归纳注释

在本例中，使用了 `sprintf()` 函数将浮点数转换成字符串，在 C 语言中，`sprintf()` 函数的定义如下 “`int sprintf(char *buffer,const char *format [,argument].....)`”，其中，第 1 个参数是转换后的字符串。注意这个字符串一定要事先分配内存，不能是未分配内存的指针，否则将产生运行时错误。第 2 个参数是格式控制参数，其格式与 `printf()` 函数中的格式控制完全一样。在本例中，使用了形如 “`sprintf(je,"% .01f",100*x)`” 这样的语句，就是将 “`100*x`” 这个浮点数按小数点后面保留两位浮点精度的格式转换到字符串 `je` 中去。`sprintf()` 函数的返回值是转换的字符个数。`sprintf()` 会自动在 `buffer` 字符串后面附加字符串结束标记符“`\0`”，但是在返回值计数中不包含这个字符。

在使用 `sprintf()` 函数的时候，另一个需要注意的问题是被转换的字串长度不要超过 `buffer` 的长度，因为在 C 语言标准中没有定义如何处理这种情况，所以可能会发生莫名其妙的错误。

# 实例 54 汉诺塔问题



## 实例说明

汉诺塔问题的原始描述是有 3 个柱子 `a`, `b`, `c`，初始的时候在 `a` 柱上从上到下、从小到大放了 3 个盘子 1, 2, 3，要求经过若干次移动后把盘子按照从上到下、从小到大的顺序放在 `c` 柱上。在移动的过程中要求尺码小的盘子必须放在尺码大的盘子上面，`b` 柱子作为中转的柱子。后来，汉诺塔问题扩展到了 3 个柱子，`n` 个盘子的情况。程序运行时，根据输入汉诺塔的层数，显示移动的方案。程序运行结果如图 54-1 所示。

```
ca\ tc
This is a hanoi program.
Please input number of the plates:
4
The steps of moving plates are:
>> Move Plate No.1 from Stick A to Stick B.
>> Move Plate No.2 from Stick A to Stick C.
>> Move Plate No.1 from Stick B to Stick C.
>> Move Plate No.3 from Stick A to Stick B.
>> Move Plate No.1 from Stick C to Stick A.
>> Move Plate No.2 from Stick C to Stick B.
>> Move Plate No.1 from Stick A to Stick B.
>> Move Plate No.4 from Stick A to Stick C.
>> Move Plate No.1 from Stick B to Stick C.
>> Move Plate No.2 from Stick B to Stick A.
>> Move Plate No.1 from Stick C to Stick A.
>> Move Plate No.3 from Stick B to Stick C.
>> Move Plate No.1 from Stick A to Stick B.
>> Move Plate No.2 from Stick A to Stick C.
>> Move Plate No.1 from Stick B to Stick C.

Press any key to quit...
```

图 54-1 实例 54 汉诺塔程序运行结果



## 实例解析

递归策略是一个过程或函数，在其定义或说明中，又直接或间接调用自身的一种方法，这样通常可以把一个大型复杂的问题层层转化为一个与原问题相似的规模较小的问题来求解，递归策略只需少量的程序代码就可描述出解题过程所需要的多次重复计算，大大减少了程序的代码量。

下面首先分析汉诺塔问题的一般解法。

(1) 将 a 柱子上的 n 个盘子从 a→(b)→c，可以用 b 作为过渡盘。

① a 柱上面的 n-1 个盘子（1 号~n-1 号）从 a→(c)→b。

② 把 a 最下面的最大的那个盘子（第 n 个）从 a→c。

③ 把 n-1 个（1 号~n-1 号）盘子从 b→(a)→c。

(2) 解决步骤 (1) 中的②，把一个盘从一个柱子搬到另一个柱子，很容易解决。

(3) 解决步骤 (1) 中的①。

将 a 柱上的 n-2 个（1 号~n-2 号）盘子从 a→()→c。

把 a 柱此时最下面的最大的盘子（第 n-1 个）从 a→c。

把 n-2 个（1 号~n-2 号）盘子从 c→()→b。

(4) 解决步骤 (1) 中的③。

将 b 柱上的 n-2 个（1 号~n-2 号）盘子从 b→()→a。

把 b 柱此时最下面的最大的盘子（第 n-1 个）从 b→c。

把 n-2 个（1 号~n-2 号）盘子从 a→()→c。

.....

一直到递归的出口的条件——当盘子只有一个的时候。



## 程序代码

### 【程序 54】 汉诺塔问题

```
/*-----*/  
/*           汉诺塔问题           */  
/*-----*/  
#include <stdio.h>  
  
/* hanoil 子程序，实现将 n 个盘子从 a 移动到 c */  
void hanoil(int n,char a,char b, char c)  
{  
    if(n==1) /* 递归调用的出口，n=1 */  
        printf("  >> Move Plate No.%d from Stick %c to Stick %c.\n",n,a,c);  
    else  
    {  
        hanoil(n-1,a,c,b); /* 递归调用 */  
        printf("  >> Move Plate No.%d from Stick %c to Stick %c.\n",n,a,c);  
        hanoil(n-1,b,a,c);  
    }  
}  
  
***** 主程序*****  
void main()  
{
```

```
int n;
char a='A';
char b='B';
char c='C';
clrscr();
printf("This is a hanoil program.\nPlease input number of the plates:\n");
scanf("%d",&n);
if(n<=0)
{
    puts("n must no less than 1!");
    exit(1);
}
puts("The steps of moving plates are:");
hanoil(n,a,b,c);
puts("\n Press any key to quit...");
getch();
}
```



## 归纳注释

在本例中，首先主函数定义了汉诺塔的高度，然后定义了 3 个字符型变量用来模拟汉诺塔问题中的 3 根柱子，接着调用函数 hanoi() 模拟了将 n 个盘子从柱子 a 移到 c 上的过程。

## 实例 55 哈夫曼编码



## 实例说明

本实例实现哈夫曼树和哈夫曼编码的构造。程序运行后，首先生成了一个哈夫曼树，然后用这颗哈夫曼树对所有字符进行了哈夫曼编码，并显示出来。程序运行结果如图 55-1 所示。

```
please input the leaf num of tree:  
8  
please input the weight of every leaf  
1 2 3 4 5 6 7 1  
The Hafman code are:  
      1           1           0           1           0  
      1           1           0           0           0  
      1           0           0           0           0  
      1           0           1           1           1  
      1           1           1           0           0  
      0           0           1           1           1  
      0           1           0           0           0  
      1           1           0           1           1  
Press any key to quit...
```

图 55-1 实例 55 程序运行结果



实例解析

哈夫曼编码是根据字符出现频率对数据进行编码解码，以便对文件进行压缩的一种方法，目前大部分有效的压缩算法（比如 MP3 编码方法）都是基于哈夫曼编码的。具体的算法描述参见相关数据结构书籍。

数据压缩过程称为编码，也就是把文件中的每个字符均转换为一个惟一的二进制位串。数据解压过程称为解码，也就是把给定的二进制位字符串转换为对应的字符。

## 程序代码

### 【程序 55】 哈夫曼编码

```
#include <stdio.h>
#define MAX 1000
#define MAXSYMBS 30
#define MAXNODE 59

typedef struct
{
    int weight;
    int flag;
    int parent;
    int lchild;
    int rchild;
}huffnode;

typedef struct
{
    int bits[MAXSYMBS];
    int start;
}huffcode;

main()
{
    huffnode huff_node[MAXNODE];
    huffcode huff_code[MAXSYMBS],cd;
    int i,j,m1,m2,x1,x2,n,c,p;
/*    char symbs[MAXSYMBS],symb; */
    /*数组 huff_node 初始化*/
    clrscr();
    printf("please input the leaf num of tree:\n");
    scanf("%d",&n);
    for(i=0;i<2*n-1;i++)
    {
        huff_node[i].weight=0;
        huff_node[i].parent=0;
        huff_node[i].flag=0;
        huff_node[i].lchild=-1;
        huff_node[i].rchild=-1;
    }

    printf("please input the weight of every leaf\n");
    for(i=0;i<n;i++)
        scanf("%d",&huff_node[i].weight);
/*构造哈夫曼树*/
    for(i=0;i<n-1;i++)
    {
        m1=m2=MAX;
```

```

x1=x2=0;
for(j=0;j<n+i;j++)
{
    if (huff_node[j].weight<m1&&huff_node[j].flag==0)
    {
        m2=m1;
        x2=x1;
        m1=huff_node[j].weight;
        x1=j;
    }
    else if (huff_node[j].weight<m2&&huff_node[j].flag==0)
    {
        m2=huff_node[j].weight;
        x2=j;
    }
}
huff_node[x1].parent=n+i; /*将找出的两棵子树合并为一棵子树*/
huff_node[x2].parent=n+i;
huff_node[x1].flag=1;
huff_node[x2].flag=1;
huff_node[n+i].weight=huff_node[x1].weight+huff_node[x2].weight;
huff_node[n+i].lchild=x1;
huff_node[n+i].rchild=x2;
}
/*求字符的哈夫曼编码*/
for(i=0;i<n;i++)
{
    cd.start=n;
    c=i;
    p=huff_node[c].parent;
    while(p!=0)
    {
        if(huff_node[p].lchild==c)

            cd.bits[cd.start]=0;
        else
            cd.bits[cd.start]=1;
        cd.start=cd.start-1;
        c=p;
        p=huff_node[p].parent;
    }
    cd.start++;
    for(j=cd.start;j<=n;j++)
        huff_code[i].bits[j]=cd.bits[j];
    huff_code[i].start=cd.start;
}
/*输出字符的哈夫曼编码*/
puts("The Hafman code are:");
for(i=0;i<n;i++)
{
    for(j=huff_code[i].start;j<=n;j++)
        printf("%10d",huff_code[i].bits[j]);
}

```

```

    printf("\n");
}

puts("Press any key to quit...");
getch();
}

```



## 归纳注释

程序首先定义了哈夫曼树叶子节点的结构 huffnode 以及存放哈夫曼编码的结构体 huffcode；然后调用系统函数 printf()、scanf() 并结合 for 语句实现了叶子节点的初始化；接着使用 C 语言的基本语句实现了构造哈夫曼树的算法；之后在哈夫曼编码算法的基础上，用 C 语言对常用字符构造了哈夫曼编码；最后调用系统函数 printf() 进行哈夫曼编码的输出。

# 实例 56 图的深度优先遍历



## 实例说明

本实例实现对图的深度优先遍历。程序运行时，在屏幕上显示出图的邻接表结构，并按照深度遍历的顺序输出图的元素。程序运行结果如图 56-1 所示。

```

C:\ Command Prompt - exit
Content of the graph's Adlist is:
vertex1 -> 2 3 4
vertex2 -> 1 5 6
vertex3 -> 1 7
vertex4 -> 1 7 4
vertex5 -> 2 8
vertex6 -> 2 7
vertex7 -> 3 6 8
vertex8 -> 5 7

The end of the dfs are:
vertex[1]
vertex[2]
vertex[5]
vertex[8]
vertex[7]
vertex[3]
vertex[6]
vertex[4]

Press any key to quit...

```

图 56-1 实例 56 程序运行结果



## 实例解析

图 G 由两个集合 V 和 E 组成，记为  $G=(V,E)$ ，其中，V 是顶点的有穷非空集合，E 是 V 中顶点偶对（称为边）的有穷集。通常，也将图 G 的顶点集和边集分别记为  $V(G)$  和  $E(G)$ 。 $E(G)$  可以是空集。如果  $E(G)$  为空，则图 G 只有顶点而没有边。

如果图 G 中的每条边都是有方向的，则称 G 为有向图 (Digraph)；反之，如果图 G 中的每条边都是没有方向的，则称 G 为无向图 (Undigraph)。

图的深度优先遍历的基本思想：假定从图中某个顶点  $v_1$  为出发点，首先访问出发点  $v_1$ ，然

后任选一个 v1 的访问过的邻接点 v2，以 v2 为新的出发点继续递归进行深度优先搜索，直到图中所有顶点被访问过。



## 程序代码

### 【程序 56】 图的深度优先遍历

```
/*////////////////////////////////////////////////////////////////*/  
/*          图的深度优先遍历          */  
/*////////////////////////////////////////////////////////////////*/  
  
#include <stdlib.h>  
#include <stdio.h>  
struct node           /* 图顶点结构定义 */  
{  
    int vertex;        /* 顶点数据信息 */  
    struct node *nextnode; /* 指向下一顶点的指针 */  
};  
typedef struct node *graph; /* 图形的结构 */  
struct node head[9];       /* 图形顶点数组 */  
int visited[9];            /* 遍历标记数组 */  
.../*其他程序见光盘*/  
***** 图的深度优先搜寻法*****  
void dfs(int current)  
{  
    graph ptr;  
    visited[current] = 1; /* 记录已遍历过 */  
    printf("vertex[%d]\n",current); /* 输出遍历顶点值 */  
    ptr = head[current].nextnode; /* 顶点位置 */  
    while (ptr != NULL) /* 遍历至链表尾 */  
    {  
        if (visited[ptr->vertex] == 0) /* 如果没遍历过 */  
            dfs(ptr->vertex);  
        ptr = ptr->nextnode; /* 下一个顶点 */  
    }  
}
```



## 归纳注释

本实例先用一个二维数组 node[20][2]存放图的信息，再调用函数 creategraph(int node[20][2],int num)将它转化成邻接表的形式，接着调用函数 dfs(int current)对图形进行深度优先遍历；函数 dfs()在遍历的同时把整个邻接表中各个节点的信息输出到屏幕。

## 实例 57 图的广度优先遍历



## 实例说明

本实例实现图的广度优先遍历，程序先在屏幕上显示了整个图的邻接表结构，然后输出图按照广度优先顺序遍历的结果。程序运行结果如图 57-1 所示。

```
ca Command Prompt - exit
This is an example of Width Preferred Traverse of Graph.
The content of the graph's allist is:
vertex1 => 2 5 7
vertex2 => 1 4 8
vertex3 => 6 7
vertex4 => 2 8
vertex5 => 1 8
vertex6 => 3
vertex7 => 3 1 8
vertex8 => 4 5 2 7
The contents of BFS are:
Vertex[1]
Vertex[2]
Vertex[5]
Vertex[7]
Vertex[4]
Vertex[8]
Vertex[3]
Vertex[6]

Press any key to quit...
```

图 57-1 实例 57 程序运行结果

## 实例解析

图的广度优先遍历的基本思想：从图中某个顶点 v1 出发，访问了 v1 之后依次访问 v1 的所有邻接点；然后分别从这些邻接点出发按深度优先搜索递归访问图的其他顶点，直到所有顶点都被访问到。它类似于树的按层次遍历，其特点是尽可能优先对横向搜索，故称之为广度优先搜索。



## 程序代码

### 【程序 57】 图的广度优先遍历（只给出了核心代码）

```
*****图的广度优先搜寻法*****
*****图的广度优先遍历*****
void bfs(int current)
{
    graph ptr;

    /* 处理第一个顶点 */
    enqueue(current);           /* 将顶点存入队列 */
    visited[current] = 1;        /* 已遍历过记录标志置 1 */
    printf(" Vertex[%d]\n",current); /* 打印输出遍历顶点值 */
    while ( front != rear )      /* 队列是否为空 */
    {
        current = dequeue();     /* 将顶点从队列取出 */
        ptr = head[current].nextnode; /* 顶点位置 */
        while ( ptr != NULL )      /* 遍历至链表尾 */
        {
            if ( visited[ptr->vertex] == 0 ) /* 顶点没有遍历过 */
            {
                enqueue(ptr->vertex); /* 将定点放入队列 */
                visited[ptr->vertex] = 1; /* 置遍历标记为 1 */
            }
        }
    }
}
```

```

        printf(" Vertex[%d]\n",ptr->vertex);/* 打印遍历顶点值 */
    }
    ptr = ptr->nextnode;      /* 下一个顶点 */
}
}
}

```



## 归纳注释

与实例 56 相似，程序先利用一个二维数组 node[20][2]存放图形的信息，再通过调用函数 creategraph(int node[20][2],int num)把它转化成邻接表的形式；接着调用函数 bfs(int current)对图形进行广度优先遍历；函数 bfs()在遍历的同时将邻接表的信息输出到屏幕。

# 实例 58 求解最优交通路径



## 实例说明

程序中预置了部分城市间的距离，然后由用户选择两个城市，计算出两城市之间的最短距离以及相应的路线。程序运行结果如图 58-1 所示。此程序可在 Visual C++ 中编译（见光盘）。

```

F:\zhongzi\WorkDir\CExample\VC\Debug\58.exe

*****欢迎使用最优交通路径程序!*****

城市列表如下：

< 0>乌鲁木齐 < 1>西宁      < 2>兰州      < 3>呼和浩特
< 4>北京          < 5>天津      < 6>沈阳      < 7>长春
< 8>哈尔滨        < 9>大连      <10>西安      <11>郑州
<12>徐州          <13>成都      <14>武汉      <15>上海
<16>昆明          <17>贵州      <18>株洲      <19>南昌
<20>福州          <21>柳州      <22>南宁      <23>广州
<24>深圳

请选择起点城市 (0~24) :
15
请选择终点城市 (0~24) :
4

从上海到北京的最短路径是<最短距离为 1462km. >
上海-->徐州-->天津-->北京

请按任意键退出...

```

图 58-1 实例 58 程序运行结果



## 实例解析

运行该程序后，首先调用 CreateUDN() 函数生成一个拥有 25 个城市，30 条公路的交通图。然后调用 narrate() 函数输出提示信息，提示用户输入出发城市和终到城市的序号，再调用 ShortestPath() 函数求出最短路径，由 Output() 函数输出结果，最终退出 main() 函数，程序结束。

CreateUDN() 函数根据用户输入的顶点数和边数生成一个相应的交通图，其中顶点对应于城市，边对应于城市间的直接通路，为了便于 ShortestPath() 函数的计算，在生成的交通图中所有的通路都是双向的，即如果 A 城市到 B 城市有直接通路，且里程为 K 千米，则 B 城市到 A 城市也有直接通路，并且里程同样为 K 千米。

narrate()函数把能够进行计算的城市列表按简单的格式进行输出。

ShortestPath()利用了图论中的最短路径算法，有兴趣的读者可以参考离散数学或数据结构等书，查阅其中的 Dijkstra 算法和 Floyd 算法。

Output()函数把计算的结果格式化输出。

## 程序代码

### 【程序 58】 求解最优交通路径（只给出核心代码）

```
/*交通图最短路径程序*/\n\n#include "string.h"\n#include "stdio.h"\n\ntypedef struct ArcCell{\n    int adj; /*相邻接的城市序号*/\n}ArcCell; /*定义边的类型*/\n\ntypedef struct VertexType{\n    int number; /*城市序号*/\n    char *city; /*城市名称*/\n}VertexType; /*定义顶点的类型*/\n\ntypedef struct{\n    VertexType vex[25]; /*图中的顶点，即为城市*/\n    ArcCell arcs[25][25]; /*图中的边，即为城市间的距离*/\n    int vexnum,arcnum; /*顶点数，边数*/\n}MGraph; /*定义图的类型*/\nMGraph G; /*把图定义为全局变量*/\nint P[25][25];\nlong int D[25];\n/* 其他函数见光盘 */\nvoid ShortestPath(num) /*最短路径函数*/\nint num;\n{\n    int v,w,i,t;\n    int final[25];\n    int min;\n    for(v=0;v<25;++v)\n    {\n        final[v]=0;D[v]=G.arcs[num][v].adj;\n        for(w=0;w<25;++w) P[v][w]=0;\n        if(D[v]<20000) {P[v][num]=1;P[v][v]=1;}\n    }\n    D[num]=0;final[num]=1;\n    for(i=0;i<25;++i)\n    {\n        min=20000;\n        for(w=0;w<25;++w)\n            if(!final[w])\n                if(D[w]<min){v=w;min=D[w];}\n        final[v]=1;\n        for(w=0;w<25;++w)
```

```

if(!final[w]&&((min+G.arcs[v][w].adj)<D[w]))
{
    D[w]=min+G.arcs[v][w].adj;
    for(t=0;t<25;t++) P[w][t]=P[v][t];
    P[w][w]=1;
}
}

```



## 归纳注释

在本例中，使用到了结构（struct）的概念。C 语言是一种可扩展性极强的语言，可以支持用户自定义的数据类型。结构是其中最重要的一种（另外还有联合（union）和枚举（enum））。在结构中，可以声明一个或多个数据域，零个或多个函数域。在使用的时候，结构可以用两种方式访问。如果声明的结构变量是实例（形如“struct StructName a;”），则用 a.data 来访问 data 域；如果声明的是结构指针“struct StructName\*p;”，则用 p->data 来访问 data 域。在本例中还用到了匿名结构的声明方法，即使用“typedef struct{……}ObjName;”这种声明方法。在匿名结构中，只能声明一个实例，以后再也不能声明该类型的实例。

# 实例 59 八皇后问题



## 实例说明

八皇后问题是指出解如何在国际象棋  $8 \times 8$  棋盘上无冲突地放置八个皇后棋子。因为在国际象棋里，皇后的移动方式是横竖交叉，所以在任意一个皇后所在位置的水平、竖直和斜 45 度线上都不能有其他皇后棋子的存在。一个完整无冲突的八皇后棋子分布称为八皇后问题的一个解。本实例实现了八枚皇后棋子在  $8 \times 8$  棋盘上无冲突的放置。程序运行结果如图 59-1 所示。此程序可在 Visual C++ 中编译（见光盘）。

```

C:\> "F:\zhongzi\WorkDir\CEexample\VC\59\Debug\59.exe"
<0,5><1,2><2,4><3,6><4,0><5,3><6,1><7,7>
<0,5><1,2><2,4><3,7><4,0><5,3><6,1><7,6>
<0,5><1,2><2,6><3,1><4,3><5,7><6,0><7,4>
<0,5><1,2><2,6><3,1><4,7><5,4><6,0><7,3>
<0,5><1,2><2,6><3,3><4,0><5,7><6,1><7,4>
<0,5><1,3><2,0><3,4><4,7><5,1><6,6><7,2>
<0,5><1,3><2,1><3,7><4,4><5,6><6,0><7,2>
<0,5><1,3><2,6><3,0><4,2><5,4><6,1><7,7>
<0,5><1,3><2,6><3,0><4,7><5,1><6,4><7,2>
<0,5><1,7><2,1><3,3><4,0><5,6><6,4><7,2>
<0,6><1,0><2,2><3,7><4,5><5,3><6,1><7,4>
<0,6><1,1><2,3><3,0><4,7><5,4><6,2><7,5>
<0,6><1,1><2,5><3,2><4,0><5,3><6,7><7,4>
<0,6><1,2><2,0><3,5><4,7><5,4><6,1><7,3>
<0,6><1,2><2,7><3,1><4,4><5,0><6,5><7,3>
<0,6><1,3><2,1><3,4><4,7><5,0><6,2><7,5>
<0,6><1,3><2,1><3,7><4,5><5,0><6,2><7,4>
<0,6><1,4><2,2><3,0><4,5><5,7><6,1><7,3>
<0,7><1,1><2,3><3,0><4,6><5,4><6,2><7,5>
<0,7><1,1><2,4><3,2><4,0><5,6><6,3><7,3>
<0,7><1,2><2,0><3,5><4,1><5,4><6,6><7,3>
<0,7><1,3><2,0><3,2><4,5><5,1><6,6><7,4>

Press any key to quit...

```

图 59-1 实例 59 程序运行结果



## 实例解析

本实例使用了回溯的方法来解决八皇后问题，也就是逐次试探的方法。这个方法通过函数putchess()对自身的递归调用来实现。运行程序后，主函数调用putchess()函数在棋盘第一行第一列上放置棋子，开始向下一行递归。每一步递归中，首先检测待放位置有没有冲突出现。如果没有冲突就放下棋子，并进入下一行递归，否则检测该行的下一个位置。如果整个一行中都没有可以放置的位置，就退回上一层递归。最后，如果本次放置成功，并且递归调用深度为7，就打印输出结果。



## 程序代码

### 【程序 59】八皇后问题

```
#include <math.h>
#include <stdio.h>
#define MAX 8 /* 棋子数及棋盘大小 MAX×MAX */
int board[MAX];

void show_result()
{
    int i;
    for(i=0;i<MAX;i++)
        printf("(%d,%d)", i, board[i]);
    printf("\n");
}

/* 检查在同一直线上是否有其他棋子 */
int check_cross(int n)
{
    int i;
    for(i=0;i<n;i++){
        if(board[i]==board[n] || (n-i)==abs(board[i]-board[n]))return 1;
    }
    return 0;
}

/* 放棋子到棋盘上 */
void put_chess(int n)
{
    int i;
    for(i=0;i<MAX;i++){
        board[n]=i;
        if(!check_cross(n)){
            if(n==MAX-1) show_result(); /* 找到其中一种方法打印出结果 */
            else put_chess(n+1);
        }
    }
}

void main()
{
    clrscr();
}
```

```

    puts("The possible placements are:");
    put_chess(0);
    puts("\n Press any key to quit...");
    getch();
    return;
}

```



## 归纳注释

这是一个古老的具有代表性的问题，用计算机求解时的算法也很多，这里仅介绍一种方法，即采用一维数组来进行处理。数组的下标  $i$  表示棋盘上的第  $i$  列， $a[i]$  的值表示皇后在第  $i$  列所放的位置。例如  $a[1]=5$ ，表示在棋盘的第 1 列的第 5 行放一个皇后。程序中首先假定  $a[1]=1$ ，表示第一个皇后放在棋盘的第 1 列的第 1 行的位置上，然后试探第 2 列中皇后可能的位置，找到合适的位置后，再处理后续的各列，这样通过各列的反复试探，可以最终找出皇后的全部摆放方法。

# 实例 60 骑士巡游



## 实例说明

骑士巡游的问题简述如下：在国际象棋盘上某一位置放置一个马的棋子，然后采用象棋中“马走日字”规则，要求这匹马能不重复地走完 25 个格子。本实例用枚举方法求解骑士巡游问题。程序可自定义棋盘的大小，先输出标志矩阵，然后输入骑士在棋盘的初始位置，即可给出其中的一种解法。程序运行结果如图 60-1 所示。此程序可在 Visual C++ 中编译（见光盘）。

```

C:\> T:\zhongzi\WorkDir\CEExample\VC\60\Debug\60.exe
Please input size of the chessboard: 4
The sign matrix is:
0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0
0 0 0 0 0 0 0 1 1 0 1 0 0 0 0 0
0 0 0 0 1 0 0 0 0 1 0 1 0 0 0 0
0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0
0 0 1 0 0 0 0 0 0 0 1 0 0 1 0 0
0 0 0 1 0 0 0 0 0 0 0 0 1 1 0 1 0
1 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 1
0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0
0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0
1 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 1
0 1 0 1 1 0 0 0 0 0 0 0 0 1 0 0 0
0 0 1 0 0 1 0 0 0 0 0 0 0 0 1 0 0
0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0
0 0 0 0 0 1 0 1 0 0 0 0 1 0 0 0 0
0 0 0 0 0 1 0 1 1 0 0 0 0 0 0 0
0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0
Please input the knight's position <i,j>: 1 1
The travel steps are:
    1   6   13   0
    12   9   2   5
    7   4   11   14
    10   15   8   3
Please input the knight's position <i,j>: 0 0
Press any key to quit...

```

图 60-1 实例 60 程序运行结果



## 实例解析

本实例首先使用基本输入输出语句要求用户输入棋盘的大小（小于等于 11），然后调用子函数 creatadjm() 创建标志矩阵，并将该矩阵打印输出；接着输入棋子的初始位置 (I,j)，调用巡游函数 travel() 开始巡游；最后将棋盘矩阵输出。



## 程序代码

### 【程序 60】骑士巡游

```
/////////////////////////////骑士巡游问题/////////////////////////////  
/*  
 * 骑士巡游问题  
 */  
  
#include <stdio.h>  
int f[11][11] ; /*定义一个矩阵来模拟棋盘*/  
int adjm[121][121];/*标志矩阵，即对于上述棋盘，依次进行编号 1~121(行优先) */  
  
void creatadjm(void); /*创建标志矩阵函数声明*/  
void mark(int,int,int,int); /*将标志矩阵相应位置置 1*/  
void travel(int,int); /*巡游函数声明*/  
int n,m; /*定义矩阵大小及标志矩阵的大小*/  
  
*****主函数*****  
int main()  
{  
    int i,j,k,l;  
    printf("Please input size of the chessboard: "); /*输入矩阵的大小值*/  
    scanf("%d",&n);  
    m=n*n;  
    creatadjm(); /*创建标志矩阵*/  
    puts("The sign matrix is:");  
    for(i=1;i<=m;i++) /*打印输出标志矩阵*/  
    {  
        for(j=1;j<=m;j++)  
            printf("%2d",adjm[i][j]);  
        printf("\n");  
    }  
  
    printf("Please input the knight's position (i,j): "); /*输入骑士的初始位置*/  
    scanf("%d %d",&i,&j);  
    l=(i-1)*n+j; /*骑士当前位置对应的标志矩阵的横坐标*/  
    while ((i>0)|| (j>0)) /*对骑士位置进行判断*/  
    {  
        for(i=1;i<=n;i++) /*棋盘矩阵初始化*/  
            for(j=1;j<=n;j++)  
                f[i][j]=0;  
        k=0; /*所跳步数计数*/  
        travel(l,k); /*从 i,j 出发开始巡游*/  
        puts("The travel steps are:");
```

```

        for(i=1;i<=n;i++)
            /*巡游完成后输出巡游过程*/
        {
            for(j=1;j<=n;j++)
                printf("%4d",f[i][j]);
            printf("\n");
        }

printf("Please input the knight's position (i,j): ");/*为再次巡游输入起始位置*/
scanf("%d %d",&i,&j);
l=(i-1)*n+j;
}
puts("\n Press any key to quit... ");
getch();
return 0;
}

/*********************创建标志矩阵子函数********************/
void creatadjm()
{
    int i,j;
    for(i=1;i<=n;i++)                                /*巡游矩阵初始化*/
        for(j=1;j<=n;j++)
            f[i][j]=0;
    for(i=1;i<=m;i++)                                /*标志矩阵初始化*/
        for(j=1;j<=m;j++)
            adjm[i][j]=0;
    for(i=1;i<=n;i++)
        for(j=1;j<=n;j++)
            if(f[i][j]==0)                            /*对所有符合条件的标志矩阵中的元素置 1*/
            {
                f[i][j]=1;
                if((i+2<=n)&&(j+1<=n)) mark(i,j,i+2,j+1);
                if((i+2<=n)&&(j-1>=1)) mark(i,j,i+2,j-1);
                if((i-2>=1)&&(j+1<=n)) mark(i,j,i-2,j+1);
                if((i-2>=1)&&(j-1>=1)) mark(i,j,i-2,j-1);
                if((j+2<=n)&&(i+1<=n)) mark(i,j,i+1,j+2);
                if((j+2<=n)&&(i-1>=1)) mark(i,j,i-1,j+2);
                if((j-2>=1)&&(i+1<=n)) mark(i,j,i+1,j-2);
                if((j-2>=1)&&(i-1>=1)) mark(i,j,i-1,j-2);
            }
    return;
}

/*********************巡游子函数********************/
void travel(int p,int r)
{
    int i,j,q;
    for(i=1;i<=n;i++)
        for(j=1;j<=n;j++)
            if(f[i][j]>r) f[i][j]=0;                  /*棋盘矩阵的置大于 r 时，置 0*/
            r=r+1;                                     /*跳步计数加 1*/
}

```

```

i=((p-1)/n)+1;                                /*还原棋盘矩阵的横坐标*/
j=((p-1)%n)+1;                                /*还原棋盘矩阵的纵坐标*/
f[i][j]=r;                                     /*将 f[i][j]做为第 r 跳步的目的地*/



for(q=1;q<=m;q++)           /*从所有可能的情况出发，开始进行试探式巡游*/
{
    i=((q-1)/n)+1;
    j=((q-1)%n)+1;
    if((adjm[p][q]==1)&&(f[i][j]==0))
        travel(q,r);                         /*递归调用自身*/
}
return;
}

/******************赋值子函数********************/
void mark(int i1,int j1,int i2,int j2)
{
    adjm[(i1-1)*n+j1][(i2-1)*n+j2]=1;
    adjm[(i2-1)*n+j2][(i1-1)*n+j1]=1;
    return;
}

```

# 03

## 第三部分 数值计算 与趣味数学篇

### 精彩导读

- 验证歌德巴赫猜想
- 求 $\pi$ 的近似值
- 堆栈四则运算
- 绘制彩色抛物线
- 求解线性/非线性方程
- 求定积分

# 实例 61 绘制余弦曲线和直线的迭加

## 实例说明

在屏幕上显示余弦曲线和直线的迭加图形。程序运行效果如图 61-1 所示。

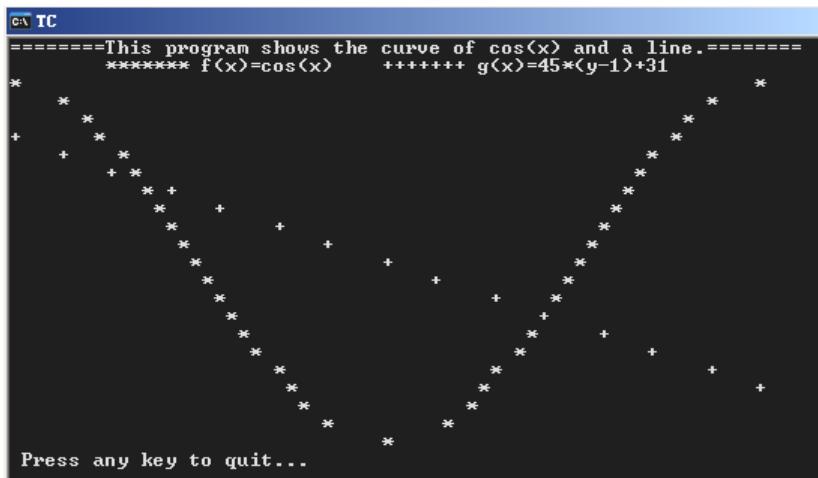


图 61-1 实例 61 程序运行效果

## 实例解析

在屏幕上显示  $0 \sim 360^\circ$  的  $\cos(x)$  曲线与直线  $f(x)=45 \times (y-1)+31$  的迭加图形。其中  $\cos(x)$  图形用“\*”表示， $f(x)$  用“+”表示，在两个图形相交的点上则用  $f(x)$  图形的符号。

如果在程序中使用数组，这个问题则变得十分简单。但若规定不能使用数组，问题就变得不容易解决了。问题的关键在于余弦曲线在  $0 \sim 360^\circ$  的区间内，一行中要显示两个点，而对一般的显示器来说，只能按行输出，即输出第一行信息后，只能向下一行输出，不能再返回到上一行。为了获得要求的图形就必须在一行中一次输出两个“\*”。

为了同时得到余弦函数  $\cos(x)$  图形在一行上的两个点，考虑利用  $\cos(x)$  的左右对称性。将屏幕的行方向定义为  $x$ ，列方向定义为  $y$ ，则  $0 \sim 180^\circ$  的图形与  $180 \sim 360^\circ$  的图形是左右对称的，若定义图形的总宽度为 62 列，计算出  $x$  行  $0 \sim 180^\circ$  时  $y$  点的坐标  $m$ ，那么在同一行与之对称的  $180 \sim 360^\circ$  度的  $y$  点的坐标就应为  $62-m$ 。程序中利用反余弦函数  $\arccos$  计算坐标  $(x,y)$  的对应关系。使用这种方法编出的程序短小精炼，体现了一定的技巧。

## 程序代码

### 【程序 61】 绘制余弦曲线与直线的迭加

//本实例源码参见光盘

## 归纳注释

图形迭加的关键是分别计算出同一行中两个图形的列方向点坐标后，正确判断相互的位置关系。为此，可以先判断图形的交点，再分别控制打印两个不同的图形。

## 实例 62 计算高次方数的尾数



计算形如  $13^{13}$  高次方的计算结果的最后 3 个尾数。程序运行效果如图 62-1 所示。

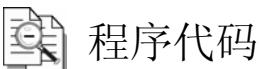
```
TC
*****
* This is a program to calculate the last 3 digits of *
* high order value, e.g., 13^15. *
*****
>> Input X and Y<X^Y>: 15 89
>> The last 3 digits of 15^89 is: 375
>> Press any key to quit...
```

图 62-1 实例 62 程序运行效果



解本题最直接的方法是将 13 累乘 13 次，截取最后 3 位即可。但是由于计算机所能表示的整数范围有限，用这种“正确”的算法不可能得到结果。事实上，题目仅要求最后 3 位的值，完全没有必要求 13 的 13 次方的完整结果。

研究乘法的规律发现，乘积的最后 3 位的值只与乘数和被乘数的后 3 位有关，与乘数和被乘数的高位无关。利用这一规律，可以大大简化程序。



### 【程序 62】 计算高次方数的尾数

//本实例源码参见光盘



在进行数学运算时必须注意计算机所能表示的各种类型数的范围，超出该范围，计算结果将会有很大变化，甚至是错误的。

## 实例 63 打鱼还是晒网



中国有句俗语叫“三天打鱼两天晒网”。某人从 1990 年 1 月 1 日起开始“三天打鱼两天晒网”，问这个人在以后的某一天中是“打鱼”还是“晒网”。程序运行效果如图 63-1 所示。此程序可在 Visual C++ 中编译（见光盘）。



图 63-1 实例 63 程序运行效果

## 实例解析

根据题意可以将解题过程分为 3 步：

- (1) 计算从 1990 年 1 月 1 日开始至指定日期共有多少天；
  - (2) 由于“打鱼”和“晒网”的周期为 5 天，所以将计算出的天数用 5 去除；
  - (3) 根据余数判断他是在“打鱼”还是在“晒网”；

若余数为 1, 2, 3, 则他是在“打鱼”, 否则是在“晒网”。

在这 3 步中，关键是第 1 步。求从 1990 年 1 月 1 日至指定日期有多少天，要判断所经历的年份中是否有闰年。闰年的判定方法可以用伪语句描述如下：

如果 ((年能被 4 除尽 且 不能被 100 除尽)或 能被 400 除尽)  
则 该年是闰年;  
否则 不是闰年。

C语言中判断能否整除可以使用求余运算（即求模）。

# 程序代码

### 【程序 63】 打鱼还是晒网

```

puts("◇ 打鱼还是晒网 ◇");
puts("◇ 中国有句俗语叫【三天打鱼两天晒网】。 ◇");
puts("◇ 某人 20 岁从 1990 年 1 月 1 日起开始【三天打鱼两天晒网】， ◇");
puts("◇ 问这个人在以后的某一天中是【打鱼】还是【晒网】？ ◇");
puts("◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇◇\n");
while(1)
{
    printf(" >> 请输入年/月/日【输入 1990 1 1 退出】: ");
    scanf("%d%d%d", &today.year, &today.month, &today.day); /* 输入日期 */
    if(today.year<1990)
    {
        if(today.year<1970)
            puts(" >> 对不起，那一年那还没出生呢！按任意键继续... ");
        else
            puts(" >> 对不起，那一年他还没开始打鱼呢！按任意键继续... ");
        getch();
        continue;
    }
    if(today.year==1990&&today.month==1&&today.day==1)
        break;
    term.month=12; /* 设置变量的初始值：月 */
    term.day=31; /* 设置变量的初始值：日 */
    for(yearday=0,year=1990;year<today.year;year++)
    {
        term.year=year;
        yearday+=days(term); /* 计算从 1990 年至指定年的前一年共有多少天 */
    }
    yearday+=days(today); /* 加上指定年中到指定日期的天数 */
    day=yearday%5; /* 求余数 */
    if(day>0&&day<4)
printf(" >> %d 年%d 月%d 日，他正在打鱼。\\n", today.year, today.month, today.day);
/* 打印结果 */
    else
printf(" >> %d 年%d 月%d 日，他正在晒网。\\n", today.year, today.month, today.day);

    }
    puts("\n >> 请按任意键退出... ");
    getch();
}

int days(struct date day)
{
    static int day_tab[2][13]=
        {{0,31,28,31,30,31,30,31,31,30,31,30,31}, /* 平均每月的天数 */
         {0,31,29,31,30,31,30,31,31,30,31,30,31},
    };
    int i,lp;
    lp=day.year%4==0&&day.year%100!=0||day.year%400==0;
    /* 判定 year 为闰年还是平年，lp=0 为平年，非 0 为闰年 */
    for(i=1;i<day.month;i++) /* 计算本年中自 1 月 1 日起的天数 */
        day.day+=day_tab[lp][i];
    return day.day;
}

```



## 归纳注释

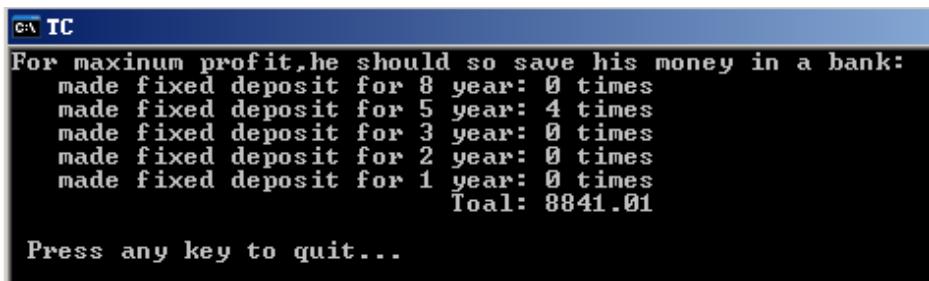
在计算天数，或者实现简单的日历、时钟时，通常都需要考虑闰年问题，可以在本实例的基础上进行改进。

# 实例 64 怎样存钱以获取最大利息



## 实例说明

通过计算利息来决定怎样存钱以便能够获取最大的利息。程序运行效果如图 64-1 所示。



```
For maximum profit, he should save his money in a bank:
made fixed deposit for 8 year: 0 times
made fixed deposit for 5 year: 4 times
made fixed deposit for 3 year: 0 times
made fixed deposit for 2 year: 0 times
made fixed deposit for 1 year: 0 times
Total: 8841.01

Press any key to quit...
```

图 64-1 实例 64 程序运行效果



## 实例解析

假设银行整存整取存款不同期限的月息利率分别为：

0.63%	期限=1 年
0.66%	期限=2 年
0.69%	期限=3 年
0.75%	期限=5 年
0.84%	期限=8 年

其中利息=本金×月息利率×12×存款年限。

现在某人手中有 2000 元钱，请通过计算选择一种存钱方案，使得钱存入银行 20 年后得到的利息最多（假定银行对超过存款期限的那一部分时间不付利息）。

根据题意，为了得到最多的利息，存入银行的钱应在到期时马上取出来，然后立刻将原来的本金和利息加起来再作为新的本金存入银行，这样不断地滚动直到满 20 年为止，由于存款的利率不同，所以不同的存款方法（年限）存 20 年得到的利息是不一样的。

分析题意，设 2000 元存 20 年，其中 1 年存  $i_1$  次，2 年存  $i_2$  次，3 年存  $i_3$  次，5 年存  $i_5$  次，8 年存  $i_8$  次，则到期时存款人应得到的本利合计为：

$$2000 \times (1+rate1)^{i1} \times (1+rate2)^{i2} \times (1+rate3)^{i3} \times (1+rate5)^{i5} \times (1+rate8)^{i8}$$

其中  $rateN$  为对应存款年限的利率。根据题意还可得到以下限制条件：

$$0 \leq i_8 \leq 2$$

$$0 \leq i_5 \leq (20 - 8 \times i_8) / 5$$

$$0 \leq i_3 \leq (20 - 8 \times i_8 - 5 \times i_5) / 3$$

$$0 \leq i_2 \leq (20 - 8 \times i_8 - 5 \times i_5 - 3 \times i_3) / 2$$
$$0 \leq i_1 = 20 - 8 \times i_8 - 5 \times i_5 - 3 \times i_3 - 2 \times i_2$$

可以用穷举法穷举出所有的  $i_8$ 、 $i_5$ 、 $i_3$ 、 $i_2$  和  $i_1$  的组合，代入求本利的公式计算出最大值，即可得到最佳存款方案。



## 程序代码

### 【程序 64】 怎样存钱以获得最大利息

//本实例源码参见光盘



## 归纳注释

在解决这类问题时可以利用穷举法将所有的情况都计算一遍，以便确定哪种是最优的。

# 实例 65 阿姆斯特朗数



## 实例说明

如果一个正整数等于其各个数字的立方和，则称该数为阿姆斯特朗数（亦称为自恋性数）。如  $407=4^3+0^3+7^3$  就是一个阿姆斯特朗数。试编程求 1000 以内的所有阿姆斯特朗数。程序运行效果如图 65-1 所示。

```
ca Command Prompt - exit
    This program will find the Armstrong number.

>> Please input the range you want to find <2~n>:
>> 9000
>> There are following Armstrong number smaller than 9000:
    153  370  371  407
Press any key to quit...
```

图 65-1 实例 65 程序运行效果



## 实例解析

可采用穷举法，依次取 1000 以内的各数（设为  $i$ ），将  $i$  的各位数字分解后，据阿姆斯特朗数的性质进行计算和判断。



## 程序代码

### 【程序 65】 阿姆斯特朗数

//本实例源码参见光盘



## 归纳注释

在求取各种性质的数时，通常采用穷举法，将某个范围内的数一一根据该数的特定性质进行判断，符合的就输出。

# 实例 66 亲密数



## 实例说明

如果整数 A 的全部因子（包括 1，不包括 A 本身）之和等于 B；且整数 B 的全部因子（包括 1，不包括 B 本身）之和等于 A，则将整数 A 和 B 称为亲密数。求 3000 以内的全部亲密数。程序运行结果如图 66-1 所示。

```
Command Prompt - exit
=====
This is a program to find friendly numbers pair.
Which means the sum of integer A's all factors (except A)
equals to the sum of integer B's all factors (except B).
< e.g. sum of integer 6's all factors are:1+2+3=6 >
=====
Please input the scale you want to find n: 10000
There are following friendly--numbers pair smaller than 10000:
220.. 284    1184..1210    2620..2924    5020..5564    6232..6368
Press any key to quit...
```

图 66-1 实例 66 程序运行结果



## 实例解析

按照亲密数定义，要判断数 a 是否有亲密数，只要计算出 a 的全部因子的累加和为 b，再计算 b 的全部因子的累加和为 n，若 n 等于 a 则可判定 a 和 b 是亲密数。



## 程序代码

### 【程序 66】 亲密数

//本实例源码参见光盘



## 归纳注释

计算数 a 的各因子的算法是用 a 依次对  $i(i=1 \sim a/2)$  进行模运算，若模运算结果等于 0，则 i 为 a 的一个因子；否则 i 就不是 a 的因子。

# 实例 67 自守数

## 实例说明

本实例求取 200 000 以内的自守数（即一个数的平方的尾数等于该数自身的自然数）。程序运行结果如图 67-1 所示。

```
Command Prompt - exit
=====
|| This program will find the automorphic numbers. ||
|| The definition of a automorphic number is: the mantissa ||
|| of a natural number's square equals to itself. ||
|| e.g., 25^2=625, 76^2=5776, 9376^2=87909376. ||
=====
>> Please input the scale n you want to find : 200000
>> The automorphic numbers small than 200000 are:
0   1   5   6   25   76   376   625   9376   90625   109376
>> Press any key to quit...
```

图 67-1 实例 67 程序运行结果

## 实例解析

自守数例如：

$$25^2=625 \quad 76^2=5776 \quad 9376^2=87909376$$

若采用“求出一个数的平方后再截取最后相应位数”的方法显然是不可取的，因为计算机无法表示过大的整数。

分析手工方式下整数平方（乘法）的计算过程，以 376 为例：

376	被乘数
× 376	乘数
—	—
2256	第一个部分积=被乘数*乘数的倒数第一位
2632	第二个部分积=被乘数*乘数的倒数第二位
1128	第三个部分积=被乘数*乘数的倒数第三位
—	—
141376	积

本问题所关心的是积的最后 3 位。分析产生积的后 3 位的过程，可以看出，在每一次的部分积中，并不是它的每一位都会对积的后 3 位产生影响。总结规律可以得到：在 3 位数乘法中，对积的后 3 位产生影响的部分积分别为：

第一个部分积中：被乘数最后三位\*乘数的倒数第一位

第二个部分积中：被乘数最后二位\*乘数的倒数第二位

第三个部分积中：被乘数最后一位\*乘数的倒数第三位

将以上的部分积的后 3 位求和后截取后 3 位就是 3 位数乘积的后 3 位。这样的规律可以推广到同样问题的不同位数乘积。

按照手工计算的过程可以设计算法编写程序。



## 程序代码

### 【程序 67】 自守数

//本实例源码参见光盘



## 归纳注释

由于计算机整数大小的限制，本实例采用手工计算平方的方法进行计算求取结果的末尾若干位数。

# 实例 68 具有 $abcd=(ab+cd)^2$ 性质的数



## 实例说明

本实例求取具有  $abcd=(ab+cd)^2$  性质的全部 4 位数。程序运行结果如图 68-1 所示。

```
c:\ Command Prompt - exit
=====
|| This program will find the four figures which have ||
|| the characteristic as follows: abcd=(ab+cd)^2. ||
|| e.g., 3025=(30+25)*(30+25). ||
=====
>> There are following numbers with satisfied condition:
2025 3025 9801
>> Press any key to quit...
```

图 68-1 实例 68 程序运行结果



## 实例解析

3025 这个数具有一种独特的性质：将它平分为二段，即 30 和 25，使之相加后求平方，即  $(30+25)^2$ ，恰好等于 3025 本身。

具有这种性质的 4 位数没有分布规律，可以采用穷举法，对所有 4 位数进行判断，从而筛选出符合这种性质的 4 位数。具体算法实现，可任取一个 4 位数，将其截为两部分，前两位为 a，后两位为 b，然后套用公式计算并判断。



## 程序代码

### 【程序 68】 具有 $abcd=(ab+cd)^2$ 性质的数

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

```

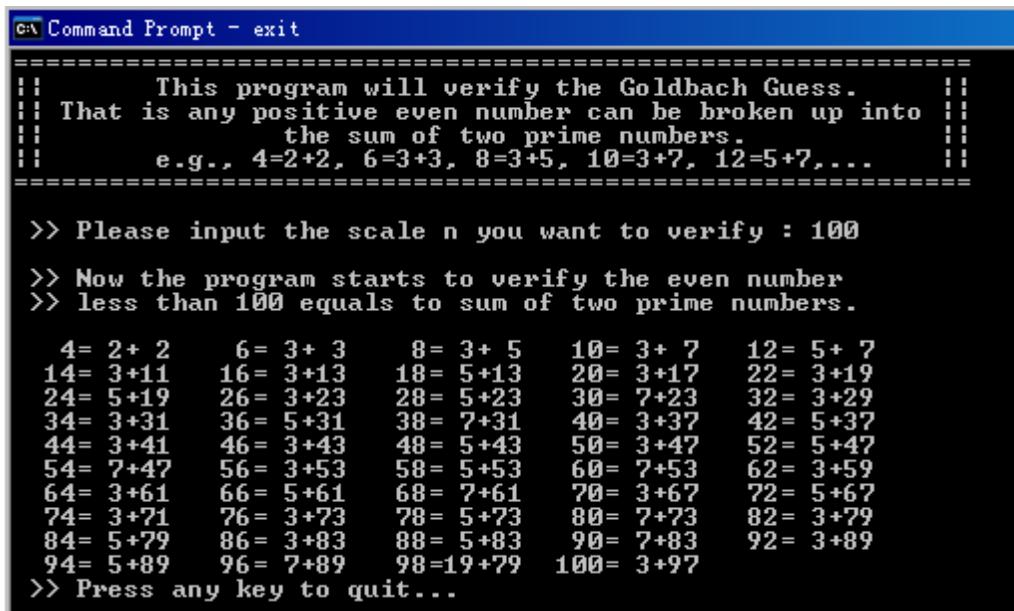
void main()
{
    int n,a,b;
    clrscr();
    puts("=====");
    puts("|| This program will find the four figures which have ||");
    puts("|| the characteristic as follows: abcd=(ab+cd)^2. ||");
    puts("|| e.g., 3025=(30+25)*(30+25). ||");
    puts("=====");
    printf("\n >> There are following numbers with satisfied condition:\n\n");
    for(n=1000;n<10000;n++) /*4位数N的取值范围1000~9999*/
    {
        a=n/100;           /*截取N的前两位数存于a*/
        b=n%100;          /*截取N的后两位存于b*/
        if((a+b)*(a+b)==n) /*判断N是否为符合题目所规定性质的4位数*/
            printf(" %d ",n);
    }
    puts("\n\n >> Press any key to quit... ");
    getch();
}

```

## 实例 69 验证歌德巴赫猜想

### 实例说明

验证 2000 以内的正偶数都能够分解为两个素数之和（即验证歌德巴赫猜想对 2000 以内的正偶数成立）。程序运行结果如图 69-1 所示。



```

c:\ Command Prompt - exit
=====
|| This program will verify the Goldbach Guess.
|| That is any positive even number can be broken up into
||       the sum of two prime numbers.
|| e.g., 4=2+2, 6=3+3, 8=3+5, 10=3+7, 12=5+7, ...
||

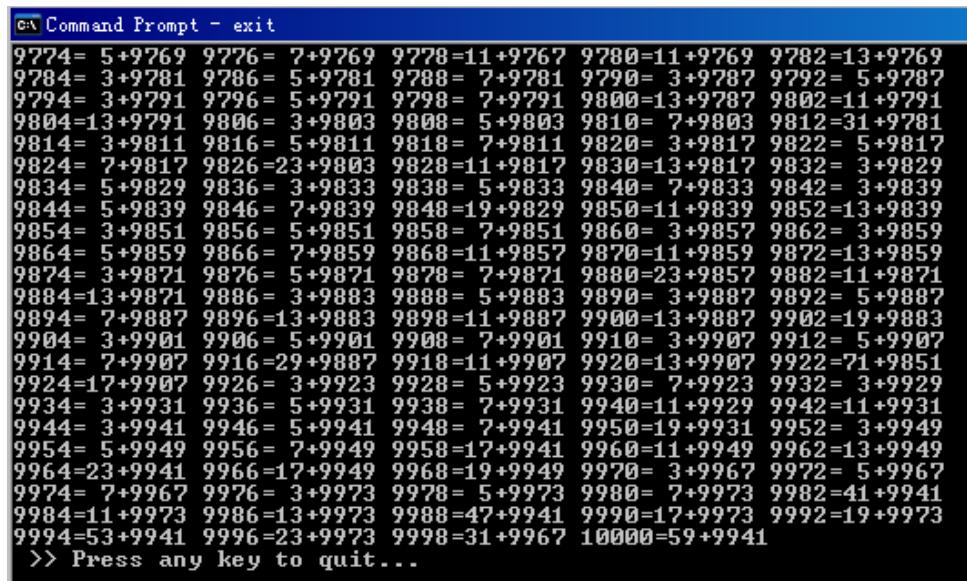
>> Please input the scale n you want to verify : 100
>> Now the program starts to verify the even number
>> less than 100 equals to sum of two prime numbers.

 4= 2+ 2      6= 3+ 3      8= 3+ 5      10= 3+ 7     12= 5+ 7
 14= 3+11     16= 3+13     18= 5+13     20= 3+17     22= 3+19
 24= 5+19     26= 3+23     28= 5+23     30= 7+23     32= 3+29
 34= 3+31     36= 5+31     38= 7+31     40= 3+37     42= 5+37
 44= 3+41     46= 3+43     48= 5+43     50= 3+47     52= 5+47
 54= 7+47     56= 3+53     58= 5+53     60= 7+53     62= 3+59
 64= 3+61     66= 5+61     68= 7+61     70= 3+67     72= 5+67
 74= 3+71     76= 3+73     78= 5+73     80= 7+73     82= 3+79
 84= 5+79     86= 3+83     88= 5+83     90= 7+83     92= 3+89
 94= 5+89     96= 7+89     98=19+79   100= 3+97

>> Press any key to quit...

```

图 69-1 实例 69 程序运行结果



```

C:\ Command Prompt - exit
9774= 5+9769 9776= 7+9769 9778=11+9767 9780=11+9769 9782=13+9769
9784= 3+9781 9786= 5+9781 9788= 7+9781 9790= 3+9787 9792= 5+9787
9794= 3+9791 9796= 5+9791 9798= 7+9791 9800=13+9787 9802=11+9791
9804=13+9791 9806= 3+9803 9808= 5+9803 9810= 7+9803 9812=31+9781
9814= 3+9811 9816= 5+9811 9818= 7+9811 9820= 3+9817 9822= 5+9817
9824= 7+9817 9826=23+9803 9828=11+9817 9830=13+9817 9832= 3+9829
9834= 5+9829 9836= 3+9833 9838= 5+9833 9840= 7+9833 9842= 3+9839
9844= 5+9839 9846= 7+9839 9848=19+9829 9850=11+9839 9852=13+9839
9854= 3+9851 9856= 5+9851 9858= 7+9851 9860= 3+9857 9862= 3+9859
9864= 5+9859 9866= 7+9859 9868=11+9857 9870=11+9859 9872=13+9859
9874= 3+9871 9876= 5+9871 9878= 7+9871 9880=23+9857 9882=11+9871
9884=13+9871 9886= 3+9883 9888= 5+9883 9890= 3+9887 9892= 5+9887
9894= 7+9887 9896=13+9883 9898=11+9887 9900=13+9887 9902=19+9883
9904= 3+9901 9906= 5+9901 9908= 7+9901 9910= 3+9907 9912= 5+9907
9914= 7+9907 9916=29+9887 9918=11+9907 9920=13+9907 9922=71+9851
9924=17+9907 9926= 3+9923 9928= 5+9923 9930= 7+9923 9932= 3+9929
9934= 3+9931 9936= 5+9931 9938= 7+9931 9940=11+9929 9942=11+9931
9944= 3+9941 9946= 5+9941 9948= 7+9941 9950=19+9931 9952= 3+9949
9954= 5+9949 9956= 7+9949 9958=17+9941 9960=11+9949 9962=13+9949
9964=23+9941 9966=17+9949 9968=19+9949 9970= 3+9967 9972= 5+9967
9974= 7+9967 9976= 3+9973 9978= 5+9973 9980= 7+9973 9982=41+9941
9984=11+9973 9986=13+9973 9988=47+9941 9990=17+9973 9992=19+9973
9994=53+9941 9996=23+9973 9998=31+9967 10000=59+9941
>> Press any key to quit...

```

图 69-2 实例 69 程序运行结果



## 实例解析

为了验证歌德巴赫猜想对 2000 以内的正偶数都是成立的，要将整数分解为两部分，然后判断分解出的两个整数是否均为素数。若是，则满足题意；否则重新进行分解和判断。



## 程序代码

### 【程序 69】 验证歌德巴赫猜想

```

#include<stdio.h>
#include<math.h>
int fflag(int n);
void main()
{
    int i,j,n;
    long max;
    clrscr();
    puts("=====");
    puts("|| This program will verify the Goldbach Guess. ||");
    puts("|| That is any positive even number can be broken up into ||");
    puts("|| the sum of two prime numbers. ||");
    puts("|| e.g., 4=2+2, 6=3+3, 8=3+5, 10=3+7, 12=5+7,... ||");
    puts("=====");
    printf("\n >> Please input the scale n you want to verify : ");
    scanf("%ld",&max);
    printf("\n >> Now the program starts to verify the even number\n");
    printf(" >> less than %ld equals to sum of two prime numbers.\n\n",max);
    for(i=4,j=0;i<=max;i+=2)
    {
        for(n=2;n<i;n++) /*将偶数 i 分解为两个整数*/
            if(fflag(n)) /*分别判断两个整数是否均为素数*/
                if(fflag(i-n))
                {
                    printf("%4d=%2d+%2d ",i,n,i-n); /*若均是素数则输出*/
                    j++;
                }
    }
}

```

```

    if( j==5 )
    {
        printf( "\n" );
        j=0;
    }
    break;
}
if(n==i) printf("error %d\n",i);
puts("\n > Press any key to quit...");
getch();
}

int fflag(int i) /*判断是否为素数*/
{
    int j;
    if(i<=1) return 0;
    if(i==2) return 1;
    if(!(i%2))return 0; /*if no,return 0*/
    for(j=3;j<=(int)(sqrt((double)i)+1);j+=2)
        if(!(i%j))return 0;
    return 1; /*if yes,return 1*/
}

```



## 归纳注释

程序中对判断是否为素数的算法进行了改进，对整数判断“用从 2 开始到该整数的一半”改为“2 开始到该整数的平方根”。

# 实例 70 素数幻方



## 实例说明

求四阶的素数幻方。即在一个  $4 \times 4$  的矩阵中，每一个格填入一个数字，使每一行、每一列和两条对角线上的 4 个数字所组成的 4 位数，均为可逆素数。程序运行结果如图 70-1 所示。

```

C:\TC
7 0 2 7
3 1 9 1
No.133
9 1 7 3
1 2 5 9
3 8 2 1
3 3 9 1
No.134
9 1 7 3
1 2 5 9
7 8 4 1
3 3 7 1
No.135
9 1 7 3
1 2 8 3
7 5 4 7
3 9 1 1
No.136
9 1 7 3
1 5 5 9
3 8 2 1
3 3 9 1
Press any key to quit...

```

图 70-1 实例 70 程序运行结果



## 实例解析

最简单的算法是穷举法，设定 $4\times 4$ 矩阵中每一个元素的值后，判断每一行、每一列和两条对角线上的4个数字组成的4位数是否都是可逆素数，若是，则求出了满足题意的一个解。

这种算法在原理是对的，也一定可以求出满足题意的全部解。但是，按照这一思路编出的程序效率很低，在计算机上运行几个小时也不会结束。这一算法致命的缺陷是要穷举和判断的情况过多。

充分利用题目中的“每一个4位数都是可逆素数”这一条件，可以放弃对矩阵中每个元素进行的穷举的算法，先求出全部的4位可逆素数共(204个)，以矩阵的行为单位，在4位可逆素数的范围内进行穷举，然后将穷举的4位整数分解为数字后，再进行列和对角线方向的条件判断，改进的算法与最初的算法相比，大大地减少了穷举的次数。

考虑矩阵的第一行和最后一行数字，它们分别是列方向4位数的第一个数字和最后一个数字，由于这些4位数也必须是可逆素数，所以矩阵的每一行和最后一行中的各个数字都不能为偶数或5。这样穷举矩阵的第一行和最后一行时，它们的取值范围是：所有位的数字均不是偶数或5的4位可逆数。由于符合这一条件的4位可逆素数很少，所以这一限制又一次减少了穷举的次数。

对算法的进一步研究会发现：当设定了第一和第二行的值后，就已经可以判断出当前的这种组合是否一定是错误的(尚不能肯定该组合一定是正确的)。若按列方向上的4个两位数与4位可逆数的前两位矛盾(不是其中的一种组合)，则第一、二行的取值一定是错误的。同理在设定了前3行数据后，可以立刻判断出当前的这种组合是否一定是错误的，若判断出矛盾情况，则可以立刻设置新的一组数据。这样就可以避免将4个数据全部设定好以后再进行判断所造成的低效。

根据以上分析，可以用伪语言描述以上改进的算法：

```
开始
    找出全部4位的可逆素数;
    确定全部出现在第一和最后一行的4位可逆素数;
    在指定范围内穷举第一行
        在指定范围内穷举第二行
            若第一、第二、第三行已出现矛盾，则继续穷举下一个数;
            在指定范围内穷举第4行
                判断列和对角方向是否符合题意
                    若符合题意，则输出矩阵;
                    否则继续穷举下一个数;
    结束
```



## 程序代码

### 【程序 70】 素数幻方

//本实例源码参见光盘



## 归纳注释

本实例中采用了很多程序设计技巧，设置若干辅助数组，其目的就是要最大限度的提高程序的执行效率，缩短运行时间。因此，该程序运行效率是比较高的。

# 实例 71 百钱百鸡问题

## 实例说明

中国古代数学家张丘建在他的《算经》中提出了著名的“百钱买百鸡问题”: 鸡翁一, 值钱五, 鸡母一, 值钱三, 鸡雏三, 值钱一, 百钱买百鸡, 问翁、母、雏各几何? 程序运行效果如图 71-1 所示。

```
*****  
* This program is to solve Problem of *  
*      Hundred Yuan Hundred Fowls.    *  
* Which is presented by Zhang Qiujiang, *  
* a Chinese ancient mathematician, in his work *  
* Bible of Calculation: 5 Yuan can buy 1 cock, *  
* 3 Yuan can buy 1 hen, 1 Yuan buy 3 chickens, *  
* now one has 100 Yuan to buy 100 fowls, the *  
* question is how many cocks, hens, chickens *  
* to buy?  
*****  
  
The possible plans to buy 100 fowls with 100 Yuan are:  
1: cock= 0 hen=25 chicken=75  
2: cock= 4 hen=18 chicken=78  
3: cock= 8 hen=11 chicken=81  
4: cock=12 hen= 4 chicken=84  
  
Press any key to quit...
```

图 71-1 实例 71 程序运行效果

## 实例解析

设鸡翁、鸡母、鸡雏的个数分别为  $x, y, z$ , 题意给定共 100 钱要买百鸡, 若全买公鸡最多买 20 只, 显然  $x$  的值在 0~20 之间; 同理,  $y$  的取值范围在 0~33 之间, 可得到下面的不定方程:

$$5x + 3y + z/3 = 100$$

$$x + y + z = 100$$

所以此问题可归结为求不定方程的整数解。

由程序设计实现不定方程的求解与手工计算不同。在分析确定方程中未知数变化范围的前提下, 可通过对未知数可变范围的穷举, 验证方程在什么情况下成立, 从而得到相应的解。

## 程序代码

### 【程序 71】 百钱百鸡问题

```
#include<stdio.h>  
void main()  
{  
    int x,y,z,j=0;  
    clrscr();  
    puts("*****");  
    puts("*      This program is to solve Problem of      *");  
    puts("*      Hundred Yuan Hundred Fowls.      *");  
    puts("* Which is presented by Zhang Qiujiang,      *");  
    puts("* a Chinese ancient mathematician, in his work *");  
    puts("* Bible of Calculation: 5 Yuan can buy 1 cock, *");
```

```

puts(" * 3 Yuan can buy 1 hen, 1 Yuan buy 3 chickens, *");
puts(" * now one has 100 Yuan to buy 100 fowls, the *");
puts(" * question is how many cocks, hens, chickens *");
puts(" * to buy? *");
puts("*****");
printf("\n The possible plans to buy 100 fowls with 100 Yuan are:\n\n");
for(x=0;x<=20;x++)           /*外层循环控制鸡翁数*/
    for(y=0;y<=33;y++)        /*内层循环控制鸡母数,y在0~33变化*/
    {
        z=100-x-y;           /*内外层循环控制下, 鸡雏数 z 的值受x,y 的值的制约*/
        if(z%3==0&&5*x+3*y+z/3==100)
            /*验证取 z 值的合理性及得到一组解的合理性*/
        printf ("%2d: cock=%2d hen=%2d chicken=%2d\n",++j,x,y,z);
    }
puts("\n Press any key to quit...");
getch();
}

```



## 归纳注释

百钱百鸡问题是经典的数学问题，在用计算机求解这类问题时，可设定相应变量的范围，并采用穷举法一一验证，找出符合条件的解。

# 实例 72 爱因斯坦的数学题



## 实例说明

爱因斯坦出了一道这样的数学题：有一条长阶梯，若每步跨 2 阶，则最后剩 1 阶，若每步跨 3 阶，则最后剩 2 阶，若每步跨 5 阶，则最后剩 4 阶，若每步跨 6 阶则最后剩 5 阶。只有每次跨 7 阶，最后才正好一阶不剩。请问这条阶梯共有多少阶？（求取得最小解）程序运行效果如图 72-1 所示。

```

*****  

* This program is to solve *  

* Einstein's interesting math question, *  

* which is presented by Albert Einstein, *  

* a famous theoretical physicist. *  

* The Problem is as follows: there is a long *  

* ladder, if one step strides 2 stages, 1 stages *  

* left, if one step strides 3 stages, 2 stages *  

* left, if one step strides 5 stages, 4 stages *  

* left, if one step strides 6 stages, 5 stages *  

* left, the question is, how many stages has *  

* the ladder?  

*****  

>> The ladder has 119 stages.  

Press any key to quit...
-
```

图 72-1 实例 72 程序运行效果



## 实例解析

根据题意，阶梯数满足下面一组同余式：

```
x≡1 (mod2)
x≡2 (mod3)
x≡4 (mod5)
x≡5 (mod6)
x≡0 (mod7)
```

通过程序对大于 1 的自然数一一进行判别，即可找到最小的解。



## 程序代码

### 【程序 72】 爱因斯坦的数学题

//本实例源码参见光盘



## 归纳注释

该题的解可能不止一个，程序所求取的只是最小的一个解。例如，可采用如下语句求取小于 10000 的解：

```
for(i=0;i<10000;i++)
    if((i%2==1)&&(i%3==2)&&(i%5==4)&&(i%6==5)&&(i%7==0))
        printf("%d ",i);
```

求出的解有：119，329，539，749，959 等。

# 实例 73 三色球问题



## 实例说明

若一个口袋中放有 12 个球，其中有 3 个红的，3 个白的和 6 个黑的，问从中任取 8 个共有多少种不同的颜色搭配？程序运行效果如图 73-1 所示。

```
*****  
* This program is to solve Problem of Three Color Ball. *  
* The Problem is as follows: There are 12 balls in the pocket. *  
* Among them, 3 balls are red, 3 balls are white and 6 balls *  
* are black. Now take out any 8 balls from the pocket, how *  
* many color combinations are there?  
*****  
>> The solutions are:  
No. RED BALL WHITE BALL BLACK BALL  
1 0 2 6  
2 0 3 5  
3 1 1 6  
4 1 2 5  
5 1 3 4  
6 2 0 6  
7 2 1 5  
8 2 2 4  
9 2 3 3  
10 3 0 5  
11 3 1 4  
12 3 2 3  
13 3 3 2  
Press any key to quit...
```

图 73-1 实例 73 程序运行效果



## 实例解析

设任取的红球个数为  $i$ , 白球个数为  $j$ , 则黑球个数为  $8-i-j$ , 根据题意红球和白球个数的取值范围是 0~3, 在红球和白球个数确定的条件下, 黑球个数取值应为  $8-i-j \leq 6$ 。



## 程序代码

### 【程序 73】三色球问题

```
#include<stdio.h>
void main()
{
    int i,j,count=0;
    clrscr();
    puts("*****");
    puts("*      This program is to solve Problem of Three Color Ball.      *");
    puts("* The Problem is as follows: There are 12 balls in the pocket. *");
    puts("* Among them, 3 balls are red,3 balls are white and 6 balls   *");
    puts("* are black. Now take out any 8 balls from the pocket, how     *");
    puts("* many color combinations are there?                         *");
    puts("*****");
    puts(" >> The solutions are:");
    printf(" No.      RED BALL  WHITE BALL  BLACK BALL\n");
    printf("-----\n");
    for(i=0;i<=3;i++)           /*循环控制变量 i 控制任取红球个数 0~3*/
        for(j=0;j<=3;j++)       /*循环控制变量 j 控制任取白球个数 0~3*/
            if((8-i-j)<=6)
                printf(" %2d | %d | %d | %d\n",++count,i,j,8-i-j);

    printf("-----\n");
    printf(" Press any key to quit... ");
    getch();
}
```



## 归纳注释

类似于三色球问题是数学中的排列(取出的先后顺序有区别)、组合(取出的先后顺序无区别)问题, 排列问题(从  $n$  个中取出  $m$  个, 顺序有区别)的解是  $P_n^m = n(n-1)K(n-m+1)$  个, 组合问题(从  $n$  个中取出  $m$  个, 顺序无区别)的解是  $C_n^m = n(n-1)K(n-m+1)/m!$  个。

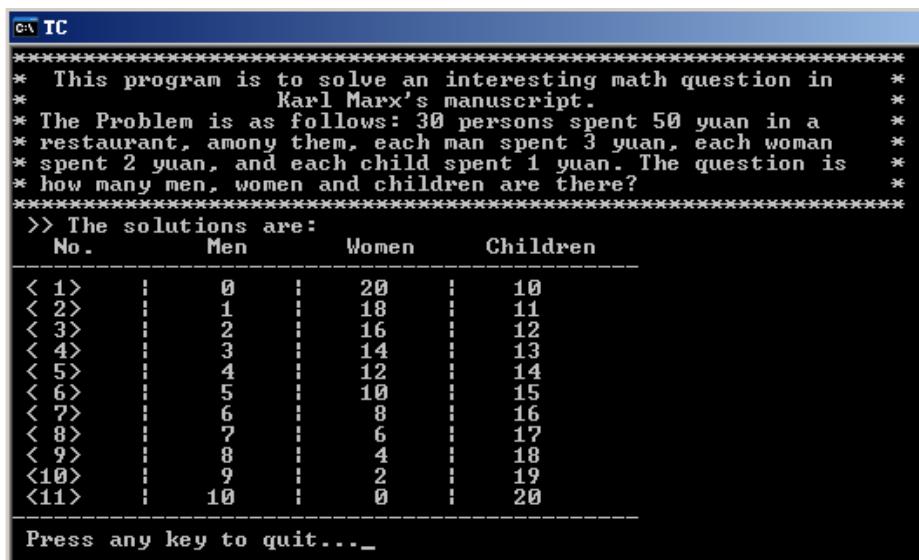
## 实例 74 马克思手稿中的数学题



## 实例说明

马克思手稿中有一道趣味数学问题: 有 30 个人, 其中有男人、女人和小孩, 在一家饭馆吃饭

花了 50 先令；每个男人花 3 先令，每个女人花 2 先令，每个小孩花 1 先令。问男人、女人和小孩各有几人？程序运行效果如图 74-1 所示。



```
*****  
* This program is to solve an interesting math question in  
* Karl Marx's manuscript.  
* The Problem is as follows: 30 persons spent 50 yuan in a  
* restaurant, among them, each man spent 3 yuan, each woman  
* spent 2 yuan, and each child spent 1 yuan. The question is  
* how many men, women and children are there?  
*****  
>> The solutions are:  
No. Men Women Children  
< 1> | 0 | 20 | 10  
< 2> | 1 | 18 | 11  
< 3> | 2 | 16 | 12  
< 4> | 3 | 14 | 13  
< 5> | 4 | 12 | 14  
< 6> | 5 | 10 | 15  
< ?> | 6 | 8 | 16  
< 8> | 7 | 6 | 17  
< 9> | 8 | 4 | 18  
<10> | 9 | 2 | 19  
<11> | 10 | 0 | 20  
Press any key to quit....
```

图 74-1 实例 74 程序运行效果

## 实例解析

设  $x$ ,  $y$ ,  $z$  分别代表男人、女人和小孩。按题目的要求，可得到下面的方程：

$$x+y+z=30 \quad (1)$$

$$3x+2y+z=50 \quad (2)$$

用程序求此不定方程的非负整数解，可先通过  $(2) - (1)$  式得：

$$2x+y=20 \quad (3)$$

由 (3) 式可知， $x$  变化范围是 0~10。

## 程序代码

### 【程序 74】 马克思手稿中的数学题

//本实例源码参见光盘

## 归纳注释

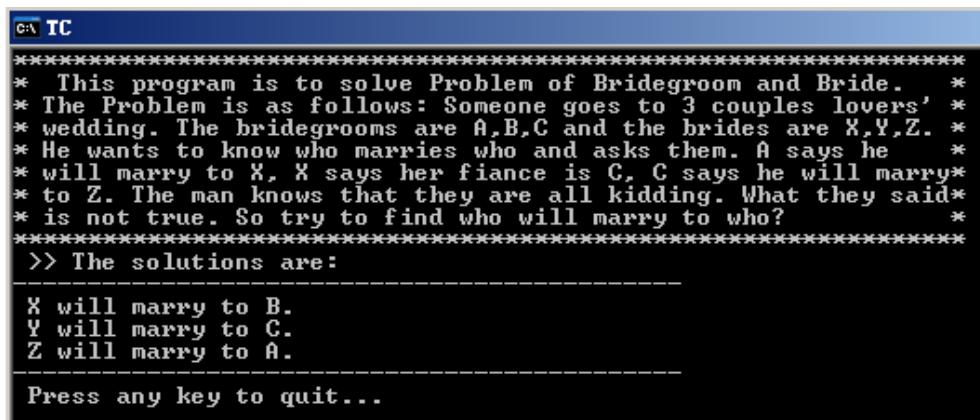
该实例相当于求解具有边界条件的线性不定方程组。

## 实例 75 配对新郎和新娘

## 实例说明

3 对情侣参加婚礼，3 个新郎为 A、B、C，3 个新娘为 X、Y、Z。有人不知道谁和谁结婚，

于是询问了 6 位新人中的 3 位，但听到的回答是这样的：A 说他将和 X 结婚；X 说她的未婚夫是 C；C 说他将和 Z 结婚。这人听后知道他们在开玩笑，全是假话。请编程找出谁将和谁结婚。程序运行效果如图 75-1 所示。



```
*****  
* This program is to solve Problem of Bridegroom and Bride. *  
* The Problem is as follows: Someone goes to 3 couples lovers' *  
* wedding. The bridegrooms are A,B,C and the brides are X,Y,Z. *  
* He wants to know who marries who and asks them. A says he *  
* will marry to X, X says her fiance is C, C says he will marry*  
* to Z. The man knows that they are all kidding. What they said*  
* is not true. So try to find who will marry to who? *  
*****  
>> The solutions are:  
-----  
X will marry to B.  
Y will marry to C.  
Z will marry to A.  
-----  
Press any key to quit...
```

图 75-1 实例 75 程序运行效果

## 实例解析

将 A、B、C 3 人用 1, 2, 3 表示，将 X 和 A 结婚表示为“X=1”，将 Y 不与 A 结婚表示为“Y!=1”。按照题目中的叙述可以写出表达式：

X!=1            A 不与 X 结婚  
X!=3            X 的未婚夫不是 C  
Z!=3            C 不与 Z 结婚

题意还隐含着 X、Y、Z 3 个新娘不能结为配偶，则有：

X!=Y 且 X!=Z 且 Y!=Z

穷举以上所有可能的情况，代入上述表达式中进行推理运算，若假设的情况使上述表达式的结果均为真，则假设情况就是正确的结果。

## 程序代码

### 【程序 75】 新郎和新娘

// 本实例源码参见光盘

## 归纳注释

这类问题通常是通过逻辑变量（非真即假）的运算来求取的。

## 实例 76 约瑟夫问题

## 实例说明

这是 17 世纪的法国数学家加斯帕在《数目的游戏问题》中讲的一个故事：15 个教徒和 15 个

非教徒在深海上遇险，必须将一半的人投入海中，其余的人才能幸免于难，于是想了一个办法：30个人围成一圆圈，从第一个人开始依次报数，每数到第九个人就将他扔入大海，如此循环进行直到仅余15个人为止。问怎样排法，才能使每次投入大海的都是非教徒。程序运行效果如图76-1所示。

图 76-1 实例 76 程序运行效果



## 实例解析

约瑟夫问题并不难，但求解的方法很多，题目的变化形式也很多。这里只给出一种实现方法。

题目中 30 个人围成一圈，因而启发我们用一个循环的链来表示。可以使用结构数组来构成一个循环链。结构中有两个成员，其一为指向下一个人的指针，以构成环形的链；其二为该人是否被扔下海的标记，为 1 表示还在船上。从第一个人开始对还未扔下海的人进行计数，每数到 9 时，将结构中的标记改为 0，表示该人已被扔下海了。这样循环计数直到有 15 个人被扔下海为止。



## 程序代码

### 【程序 76】约瑟夫问题

//本实例源码参见光盘



归纳注释

本程序采用链表环来求解。这类问题也是经典问题，类似的还有海盗分金子等。

## 实例 77 邮票组合



实例说明

某人有 4 张 3 分的邮票和 3 张 5 分的邮票，用这些邮票中的一张或若干张可以得到多少种不同的邮资？程序运行效果如图 77-1 所示。

```
c:\> TC
*****
* This program is to solve Problem of Stamp Combination. *
* The Problem is as follows. John has 4 stamps with value of 3 *
* cents and 3 stamps with value of 5 cents. Use one or more of *
* these stamps, how many kinds of postages can John provide? *
*****
>> The solution is:
-----
>> There are 19 kinds of postages:
5 10 15 3 8 13 18 6 11 16 21 9 14 19 24 12 17 22 27
-----
Press any key to quit....
```

图 77-1 实例 77 程序运行效果



## 实例解析

将问题进行数学分析，不同张数和面值的邮票组成的邮资可用下列公式计算：

$$S=3 \times i + 5 \times j$$

其中  $i$  为 3 分邮票的张数， $j$  为 5 分的张数

按题目的要求，3 分的邮票可以取 0、1、2、3 共 4 张，5 分的邮票可以取 0、1、2 共 3 张。采用穷举方法进行组合，可以求出不同面值不同张数的邮票组合后的邮资。



## 程序代码

### 【程序 77】 邮票组合

//本实例源码参见光盘



## 归纳注释

在用穷举法求解这类问题时，必须注意其中可能有相同的解，不要重复列出。

# 实例 78 分糖果



## 实例说明

10 个小孩围成一圈分糖果，老师分给第一个小孩 10 块，第二个小孩 2 块，第三个小孩 8 块，第四个小孩 22 块，第五个小孩 16 块，第六个小孩 4 块，第七个小孩 10 块，第八个小孩 6 块，第九个小孩 14 块，第十个小孩 20 块。然后所有的小孩同时将手中的糖分一半给右边的小孩；糖块数为奇数的人可向老师要一块。问经过这样几次后，大家手中的糖的块数将一样多，每人各有多少块糖？程序运行效果如图 78-1 所示。

Child No.	1	2	3	4	5	6	7	8	9	10	
Round No.:	< 0>	10	2	8	22	16	4	10	6	14	20
	< 1>	15	6	5	15	19	10	7	8	10	17
	< 2>	17	11	6	11	18	15	9	8	9	14
	< 3>	16	15	9	9	15	17	13	9	9	12
	< 4>	14	16	13	10	13	17	16	12	10	11
	< 5>	13	15	15	12	12	16	17	14	11	11
	< 6>	13	15	16	14	12	14	17	16	13	12
	< 7>	13	15	16	15	13	13	16	17	15	13
	< 8>	14	15	16	16	15	14	15	17	17	15
	< 9>	15	15	16	16	16	15	15	17	18	17
	< 10>	17	16	16	16	16	16	17	18	18	18
	< 11>	18	17	16	16	16	16	17	18	18	18
	< 12>	18	18	17	16	16	16	17	18	18	18
	< 13>	18	18	18	17	16	16	16	17	18	18
	< 14>	18	18	18	18	17	16	16	17	18	18
	< 15>	18	18	18	18	18	17	16	17	18	18
	< 16>	18	18	18	18	18	18	17	17	18	18
	< 17>	18	18	18	18	18	18	18	18	18	18

Press any key to quit...

图 78-1 实例 78 程序运行效果

## 实例解析

分析题目可知，题目描述的分糖过程是一个机械的重复过程，算法完全可以按照描述的过程进行模拟。

## 程序代码

### 【程序 78】 分糖果

```
#include<stdio.h>
void print(int s[]);
int judge(int c[]);
int j=0;
void main()
{
    static int sweet[10]={10,2,8,22,16,4,10,6,14,20}; /*初始化数组数据*/
    int i,t[10],l;
    clrscr();
    printf(" Child No.    1    2    3    4    5    6    7    8    9    10\n");
    printf("-----\n");
    printf(" Round No. |\n");
    print(sweet);      /*输出每个人手中糖的块数*/
    while(judge(sweet)) /*若不满足要求则继续进行循环*/
    {
        for(i=0;i<10;i++) /*将每个人手中的糖分成一半*/
            if(sweet[i]%2==0) /*若为偶数则直接分出一半*/
                t[i]=sweet[i]=sweet[i]/2;
            else /*若为奇数则加 1 后再分出一半*/
                t[i]=sweet[i]=(sweet[i]+1)/2;
        for(l=0;l<9;l++) /*将分出的一半糖给右(后)边的孩子*/
            sweet[l+1]=sweet[l+1]+t[l];
        sweet[0]+=t[9];
        print(sweet);      /*输出当前每个孩子中手中的糖数*/
    }
    printf("-----\n");
}
```

```

printf("\n Press any key to quit...");  

getch();  

}  

int judge(int c[])  

{  

    int i;  

    for(i=0;i<10;i++)           /*判断每个孩子手中的糖是否相同*/  

        if(c[0]!=c[i]) return 1;      /*不相同则返回 1*/  

    return 0;  

}  

void print(int s[])      /*输出数组中每个元素的值*/  

{  

    int k;  

    printf("      <%2d> | ",j++);  

    for(k=0;k<10;k++)  printf("%4d",s[k]);  

    printf("\n");  

}

```



## 归纳注释

本实例根据题目中分糖果的过程直接给出了实现的代码。在求解比较简单、直观的问题时，可采用这种方法。

# 实例 79 波瓦松的分酒趣题



## 实例说明

法国著名数学家波瓦松（Poisson）在青年时代研究过一个有趣的数学问题：某人有 12 品脱的啤酒一瓶，想从中倒出 6 品脱，但他没有 6 品脱的容器，仅有一个 8 品脱和 5 品脱的容器，怎样倒才能将啤酒分为两个 6 品脱呢？程序运行效果如图 79-1 所示。

Step No.	a<12>	b<8>	c<5>
<0>	12	0	0
<1>	4	8	0
<2>	4	3	5
<3>	9	3	0
<4>	9	0	3
<5>	1	8	3
<6>	1	6	5
<7>	6	6	0

图 79-1 实例 79 程序运行效果



## 实例解析

将 12 品脱酒用 8 品脱和 5 品脱的空瓶平分，可以抽象为解不定方程：

$$8x - 5y = 6$$

其意义是：从 12 品脱的瓶中向 8 品脱的瓶中倒  $x$  次，并且将 5 品脱瓶中的酒向 12 品脱的瓶中倒  $y$  次，最后在 12 品脱的瓶中剩余 6 品脱的酒。

分别用  $a$ ,  $b$ ,  $c$  代表 12 品脱、8 品脱和 5 品脱的瓶子，求出不定方程的整数解，按照不定方程的意义则倒酒法为：

$$\begin{array}{c} a \rightarrow b \rightarrow c \rightarrow a \\ x \qquad y \end{array}$$

倒酒的规则如下：

- (1) 按  $a \rightarrow b \rightarrow c \rightarrow a$  的顺序；
- (2)  $b$  倒空后才能从  $a$  中取；
- (3)  $c$  装满后才能向  $a$  中倒。



## 程序代码

### 【程序 79】 波瓦松的分酒趣题

//本实例源码参见光盘



## 归纳注释

这类约束分配问题采用递归求解比较方便。

# 实例 80 求 $\pi$ 的近似值



## 实例说明

用两种方法编程实现求取  $\pi$  的近似值。程序运行效果如图 80-1 所示。

```
CA IC
*****
* This program is to calculate PI approximatively *
* in two methods. *
* One method is Regular Polygon Approximating, *
* the other is Random Number Method. *
*****
>> Result of Regular Polygon Approximating:
>> pi=3.141592653589793
>> The number of edges of required polygon:100663296
-----
>> Result of Random Number Method:
>> pi=3.139600
-----
Press any key to quit...
```

图 80-1 实例 80 程序运行效果



## 实例解析

### 利用“正多边形逼近”的方法求出 $\pi$ 的近似值

利用“正多边形逼近”的方法求出 $\pi$ 值在很早以前就存在，我们的先人祖冲之就是用这种方法在世界上第一个得到精确度达小数点后第6位的 $\pi$ 值的。

利用圆内接正六边形边长等于半径的特点将边数翻番，作出正十二边形，求出边长，重复这一过程，就可获得所需精度的 $\pi$ 的近似值。

假设单位圆内接多边形的边长为 $2b$ ，边数为*i*，则边数加倍后新的正多边形的边长为：

$$x = \frac{\sqrt{2 - 2 \times \sqrt{1 - b \times b}}}{2}$$

周长为：

$y=2ix$  其中*i*为加倍前的正多边形的边数。

### 利用随机数法求 $\pi$ 的近似值

随机数法求 $\pi$ 的近似值的思路：在一个单位边长的正方形中，以边长为半径，以一个顶点为圆心，在正方形上做四分之一圆。随机的向正方形内扔点，若落入四分之一圆内则计数。重复向正方形内扔足够多的点，将落入四分之一圆内的计数除以总的点数，其值就是 $\pi$ 值四分之一的近似值。

按此方法可直接进行编程。



## 程序代码

### 【程序 80】 求 $\pi$ 的近似值

```
#include<stdio.h>
#include<math.h>
#include<time.h>
#include<stdlib.h>
#define N 30000
void main()
{
    double e=0.1,b=0.5,c,d;
    long int i; /*i: 正多边形边数*/
    float x,y;
    int c2=0,d2=0;
    clrscr();
    puts("*****");
    puts("*      This program is to calculate PI approximatively      *");
    puts("*                  in two methods.                          *");
    puts("*      One method is Regular Polygon Approximating,     *");
    puts("*      the other is Random Number Method.                 *");
    puts("*****");
    puts("\n >> Result of Regular Polygon Approximating:");
    for(i=6;;i*=2) /*正多边形边数加倍*/
    {
        d=1.0-sqrt(1.0-b*b); /*计算圆内接正多边形的边长*/
    }
}
```

```

b=0.5*sqrt(b*b+d*d);
if(2*i*b-i*e<1e-15) break; /*精度达 1e-15 则停止计算*/
e=b; /*保存本次正多边形的边长作为下一次精度控制的依据*/
}
printf("-----\n");
printf(" >> pi=%.15lf\n",2*i*b); /*输出π值和正多边形的边数*/
printf(" >> The number of edges of required polygon:%ld\n",i);
printf("-----\n");
randomize();
while(c2++<=N)
{
    x=random(101); /*x 坐标, 产生 0~100 之间共 101 个的随机数*/
    y=random(101); /*y 坐标, 产生 0~100 之间共 101 个的随机数*/
    if(x*x+y*y<=10000) /*利用圆方程判断点是否落在圆内*/
        d2++;
}
puts("\n >> Result of Random Number Method:");
printf("-----\n");
printf(" >> pi=%f\n",4.*d2/N); /*输出求出的π值*/
printf("-----\n");
puts("\n Press any key to quit..."); getch();
}

```



## 归纳注释

从程序运行结果可看出, 利用“正多边形逼近法”求取的 $\pi$ 值比利用“随机数法”求出的 $\pi$ 值要更接近其真实值, 这是因为, “随机数法”只有统计次数足够多时才可能准确。

# 实例 81 奇数平方的有趣性质



## 实例说明

编程验证大于 1000 的奇数其平方与 1 的差是 8 的倍数。程序运行效果如图 81-1 所示。

```

*****>>> This program is to verify << ****
*>>> odd number's characteristic. << ****
* That is square of an odd number larger than 1000 minus ****
* 1 can be divided exactly by 8. ****
* For example, 2001^2-1=4004000=500500*8. ****
*****
>> Please input the range you want to verify: 1020
>> Now start to verify:
1001:(1001*1001-1)/8=125250+0
1003:(1003*1003-1)/8=125751+0
1005:(1005*1005-1)/8=126253+0
1007:(1007*1007-1)/8=126756+0
1009:(1009*1009-1)/8=127260+0
1011:(1011*1011-1)/8=127765+0
1013:(1013*1013-1)/8=128271+0
1015:(1015*1015-1)/8=128778+0
1017:(1017*1017-1)/8=129286+0
1019:(1019*1019-1)/8=129795+0

Press any key to quit...

```

图 81-1 实例 81 程序运行效果



## 实例解析

本题是一个很容易证明的数学定理，我们可以编写程序验证它。

题目中给出的处理过程很清楚，算法不需要特殊设计。可以按照题目的叙述直接进行验证（程序中验证的范围可由用户输入）。



## 程序代码

### 【程序 81】 奇数平方的有趣性质

//本实例源码参见光盘



## 归纳注释

由于本例中的运算结果可能超出 int 整型数的范围，因此采用长整型 long 来定义变量。

# 实例 82 角谷猜想



## 实例说明

日本一位中学生发现一个奇妙的“定理”，请角谷教授证明，而教授无能为力，于是产生角谷猜想。猜想的内容是：任意给一个自然数，若为偶数则除以 2，若为奇数则乘 3 加 1，得到一个新的自然数，然后按照上面的法则继续演算，若干次后得到的结果必然为 1。请编程验证。程序运行效果如图 82-1 所示。

```
*****>> This program is to verify Jiaogu Guess << ****
* That is given any natural number, if it is an even, *
* divides 2, if it is an odd, multiple 3 and add 1, the *
* result continues to be calculated analogously. After *
* some times, the result is always 1. *
*****
>> Please input a number to verify(0 to quit): 6
>> ----- Results of verification: -----
>> Step No.1: 6/2=3
>> Step No.2: 3*3+1=10
>> Step No.3: 10/2=5
>> Step No.4: 5*3+1=16
>> Step No.5: 16/2=8
>> Step No.6: 8/2=4
>> Step No.7: 4/2=2
>> Step No.8: 2/2=1
>>
>> Please input a number to verify(0 to quit): 0
Press any key to quit...
```

图 82-1 实例 82 程序运行效果



## 实例解析

本题是一个还未获得一般证明的猜想，但屡试不爽，可以用程序验证。

题目中给出的处理过程很清楚，算法不需特殊设计，可按照题目的叙述直接进行验证。



## 程序代码

### 【程序 82】 角谷猜想

//本实例源码参见光盘



## 归纳注释

本实例程序在 while 循环中对输入的数进行验证，在输入为 0 时，则退出程序。

# 实例 83 四方定理



## 实例说明

数论中著名的“四方定理”讲的是：所有自然数至多只要用 4 个数的平方和就可以表示。请编程验证此定理。程序运行效果如图 83-1 所示。

```
*****  
* This program is to Theorem of Four Squares. *  
* That is all natural numbers can be represented as *  
* sum of no more than 4 squares of the numbers. *  
*****  
>> Please input a number to verify<0 to quit>: 900  
>> ----- Results of verification: -----  
>> 900=15*15+15*15+15*15+15*15  
>>  
>> Please input a number to verify<0 to quit>: 800  
>> ----- Results of verification: -----  
>> 800=16*16+16*16+12*12+12*12  
>>  
>> Please input a number to verify<0 to quit>: 700  
>> ----- Results of verification: -----  
>> 700=15*15+15*15+13*13+9*9  
>>  
>> Please input a number to verify<0 to quit>: 2000  
>> ----- Results of verification: -----  
>> 2000=26*26+26*26+18*18+18*18  
>>  
>> Please input a number to verify<0 to quit>: 0  
Press any key to quit...
```

图 83-1 实例 83 程序运行效果



## 实例解析

对 4 个变量采用试探的方法进行计算，满足要求时输出计算结果。



## 程序代码

### 【程序 83】 四方定理

//本实例源码参见光盘



## 归纳注释

程序对任意输入的不为 0 (为 0 则程序结束) 的数进行试探, 满足定理则输出结果。

# 实例 84 卡布列克常数



## 实例说明

任意一个 4 位数, 只要它们各个位上的数字是不全相同的, 就有如下的规律:

- (1) 将组成该 4 位数的 4 个数字由大到小排列, 形成由这 4 个数字构成的最大的 4 位数;
- (2) 将组成该 4 位数的 4 个数字由小到大排列, 形成由这 4 个数字构成的最小的 4 位 (如果 4 个数中含有 0, 则得到的数不足 4 位);
- (3) 求两个数的差, 得到一个新的 4 位数 (高位零保留)。

重复以上过程, 最后得到的结果是 6174, 这个数被称为卡布列克数。请编程验证卡布列克常数。程序运行效果如图 84-1 所示。

```
GW TC
*****
*      This program is to verify Comgrich Content.      *
* That is any 4-digit number whose digits are not the same *
* has the rule: (1) range the 4 digits to get the maximum   *
* 4-digit number, (2) range the 4 digits to get the minimum   *
* 4-digit number, (3) get the difference of these two numbers*
* that is a new 4-digit number. Continue to calculate with   *
* (1)-(3), the result in the end is 6174, the Comgrich Content. *
*****
>> Please input a 4-digit number to verify(0 to quit): 4312
>> ----- Results of verification: -----
>> Step No.1: 4321-1234=3087
>> Step No.2: 8730-378=8352
>> Step No.3: 8532-2358=6174
>>
>> Please input a 4-digit number to verify(0 to quit): 9643
>> ----- Results of verification: -----
>> Step No.1: 9643-3469=6174
>>
>> Please input a 4-digit number to verify(0 to quit): 0
Press any key to quit...
```

图 84-1 实例 84 程序运行效果



## 实例解析

题目中给出的处理过程很清楚, 算法不需要特殊设计, 可按照题目的叙述直接进行验证。



## 程序代码

### 【程序 84】 卡布列克常数

//本实例源码参见光盘



## 归纳注释

程序通过 vr6174 函数的递归调用进行验证，通过 parse\_sort 函数将 NUM 分解为数字，函数 max\_min 将分解的数字组成最大整数和最小整数。

# 实例 85 尼科彻斯定理



## 实例说明

验证尼科彻斯定理，即任何一个整数的立方都可以写成一串连续奇数的和。程序运行效果如图 85-1 所示。

```

C:\TC
*****
* This program is to verify Theorem of Nicoqish. *
* That is the cube of any integer can be represented *
* as the sum of some continue odd numbers. *
* For example, 8^3=512=57+59+59+61+63+65+67+69+71. *
*****
>> Please input a integer to verify<0 to quit>: 8
>> ----- Results of verification: -----
>> 8*8*8=512=57+59+59+61+63+65+67+69+71 Satisfy!
>>
>> Please input a integer to verify<0 to quit>: 10
>> ----- Results of verification: -----
>> 10*10*10=1000=91+93+95+97+99+101+103+105+107+109 Satisfy!
>>
>> Please input a integer to verify<0 to quit>: 0
Press any key to quit...

```

图 85-1 实例 85 程序运行效果



## 实例解析

本题是一个定理，我们先来证明它是成立的。

对于任一正整数  $a$ ，不论  $a$  是奇数还是偶数，整数  $(axa-a+1)$  必然为奇数。

构造一个等差数列，数列的首项为  $(axa-a+1)$ ，等差数列的差值为 2（奇数数列），则前  $a$  项的和为：

$$\begin{aligned} &a \times ((axa-a+1))+2 \times a(a-1)/2 \\ &= axaxa - axa + a + axa - a \\ &= axaxa \end{aligned}$$

定理成立。

通过定理的证明过程可知所要求的奇数数列的首项为  $(axa-a+1)$ ，长度为  $a$ 。编程的算法不需要特殊设计，可按照定理的证明过程直接进行验证。



## 程序代码

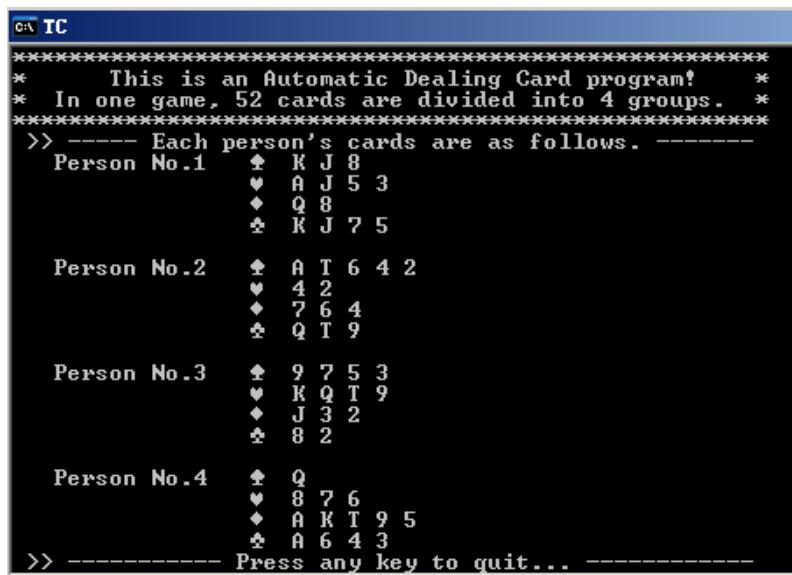
### 【程序 85】 尼科彻斯定理

// 本实例源码参见光盘

# 实例 86 扑克牌自动发牌

## 实例说明

一副扑克有 52 张牌，打桥牌时应将牌分给 4 个人。请设计一个程序完成自动发牌的工作。要求：黑桃用 S (Spaces) 表示；红桃用 H (Hearts) 表示；方块用 D (Diamonds) 表示；梅花用 C (Clubs) 表示。程序运行效果如图 86-1 所示 (T 代表 10)。



The screenshot shows a terminal window titled 'TC' with the following output:

```
*****  
* This is an Automatic Dealing Card program! *  
* In one game, 52 cards are divided into 4 groups. *  
*****  
>> ----- Each person's cards are as follows. -----  
Person No.1 ♦ K J 8  
          ♦ A J 5 3  
          ♦ Q 8  
          ♦ K J 7 5  
  
Person No.2 ♦ A T 6 4 2  
          ♦ 4 2  
          ♦ 7 6 4  
          ♦ Q T 9  
  
Person No.3 ♦ 9 7 5 3  
          ♦ K Q T 9  
          ♦ J 3 2  
          ♦ 8 2  
  
Person No.4 ♦ Q  
          ♦ 8 7 6  
          ♦ A K T 9 5  
          ♦ A 6 4 3  
  
>> ----- Press any key to quit... -----
```

图 86-1 实例 86 程序运行效果

## 实例解析

按照打桥牌的规定，每人应当有 13 张牌。在人工发牌时，先进行洗牌，然后将洗好的牌按一定的顺序发给每一个人。为了便于计算机模拟，可将人工方式的发牌过程加以修改：先确定好发牌顺序 1、2、3、4；将 52 张牌顺序编号：黑桃 2 对应数字 0，红桃 2 对应数字 1，方块 2 对应数字 2，梅花 2 对应数字 3，黑桃 3 对应数字 4，红桃 3 对应数字 5……然后从 52 张牌中随机的为每个人抽牌。

## 程序代码

### 【程序 86】 扑克牌自动发牌

// 本实例源码参见光盘

## 归纳注释

本实例采用 C 语言库函数的随机函数，生成 0~51 之间的共 52 个随机数，以产生洗牌后发牌的效果。

# 实例 87 常胜将军

## 实例说明

现有 21 根火柴，两人轮流取，每人每次可以取走 1~4 根，不可多取，也不能不取，谁取最后一根火柴则谁输。请编写一个程序进行人机对弈，要求人先取，计算机后取；计算机一方为“常胜将军”。程序运行效果如图 87-1 所示。此程序可在 Visual C++ 中编译（见光盘）。

```
ca "F:\zhongzi\WorkDir\CExample\VC\87\Debug\87.exe"
*****
*          This is a Matchstick Taken Game.      *
* There are 21 machsticks, two persons take them in *
* turn. Each one each time takes 1 to 4 sticks. The *
* one who takes the last stick will lose the game.  *
*****
>> ----- Game Begin -----
>> How many sticks do you wish to take<1~4>?4
>> 17 stick left in the pile.
>> Compute take 1 stick.
>> 16 stick left in the pile.
>> How many sticks do you wish to take<1~4>?4
>> 12 stick left in the pile.
>> Compute take 1 stick.
>> 11 stick left in the pile.
>> How many sticks do you wish to take<1~4>?4
>> 7 stick left in the pile.
>> Compute take 1 stick.
>> 6 stick left in the pile.
>> How many sticks do you wish to take<1~4>?3
>> 3 stick left in the pile.
>> Compute take 2 stick.
>> 1 stick left in the pile.
>> How many sticks do you wish to take<1~1>?1
>> You have taken the last stick.
>> ***** You lose! *****
>> ----- Game Over! ----

Press any key to quit...  

```

图 87-1 实例 87 程序运行效果

## 实例解析

在计算机后走的情况下，要想使计算机成为“常胜将军”，必须找出取法的关键。根据本题的要求加以总结，后走一方取子的数量与对方刚才一步取子的数量之和等于 5，就可以保证最后一个子是留给先取子的那个人的。

据此分析进行算法设计就是很简单的工作，编程实现也十分容易。

## 程序代码

### 【程序 87】 常胜将军

// 本实例源码参见光盘



## 归纳注释

本程序的关键在于后取一方的取法，即与先取一方所取的数目之和为 5，即可保证常胜。

# 实例 88 搬山游戏



## 实例说明

设有 n 座山，计算机与人为比赛的双方，轮流搬山。规定每次搬山的数止不能超过 k 座，谁搬最后一座谁输。游戏开始时。计算机请人输入山的总数 (n) 和每次允许搬山的最大数 (k)。然后请人先开始，等人输入了需要搬走的山的数目后，计算机马上打印出它要搬多少座山，并提示尚余多少座山。双方轮流搬山直到最后一座山搬完为止。计算机会显示谁是赢家，并问人是否要继续比赛。若人不想玩了，计算机便会统计出共玩了几局，双方胜负如何。程序运行效果如图 88-1 所示。此程序可在 Visual C++ 中编译（见光盘）。

```
CD "F:\zhongzi\WorkDir\CExample\VC\88\Debug\88.exe"
*****
*          This is a Mountain Moveing Game.      *
* There are n mountains, two persons move them in    *
* turn. Each one each time moves 1 to k mountains, the*
* one who takes the last stick will lose the game.   *
*****
>> ----- Game Begin -----
>> No. 1 game
>>
>> How many mountains are there? 10
>> How many mountains are allowed to each time? 3
>> How many mountains do you wish move ? 3
>> There are 7 mountains left now.
>> Computer move 2 mountains away.
>> There are 5 mountains left now.
>> How many mountains do you wish move ? 2
>> There are 3 mountains left now.
>> Computer move 2 mountains away.
>> There are 1 mountains left now.
>> How many mountains do you wish move ? 1
>> There are 0 mountains left now.
>> ---- I win. You are failure.-----

>> No. 2 game
>>
>> How many mountains are there? 0
>> Games in total have been played 1.
>> You score is win 0,lose 1.
>> My score is win 1,lose 0.
>> ----- Game Over! ----

Press any key to quit..._
```

图 88-1 实例 88 程序运行效果



## 实例解析

计算机参加游戏时应遵循下列原则：

(1) 当剩余山数目 $-1 \leq k$ 时，计算机要移 $(n-1)$ 座，以便将最后一座山留给人。

(2) 对于任意正整数 $x, y$ ，一定有：

$$0 \leq x \% (y+1) \leq y$$

在有 $n$ 座山的情况下，计算机为了将最后一座山留给人，而且又要控制每次搬山的数目不超过最大数 $k$ ，它应搬山的数目要满足下列关系：

$$(n-1) \% (k+1)$$

如果算出结果为0，即整除无余数，则规定只搬1座山，以防止冒进后发生问题。



## 程序代码

### 【程序 88】 搬山游戏

//本实例源码参见光盘

## 实例 89 兔子产子（菲波那契数列）



## 实例说明

从前有一对长寿兔子，它们每一个月生一对兔子，新生的小兔子两个月就长大了，在第二个月的月底开始生它们的下一代小兔子，这样一代一代生下去，求解兔子增长数量的数列。程序运行效果如图 89-1 所示。

```
CA TC
*****
* This is a program to Calculate Rabbits Numbers. *
* There is a rabbit couple procreats 2 rabbits 1 month,*
* and the young rabbits group can procreats in the *
* end of the second month. In this way, how many rabbits*
* are there after n generations? *
*****
>> Please input number of generations <n>2: 23
>> The numbers of rabbits in first 23 generation are as follows:
    1      1      2      3      5      8      13     21
    34     55     89    144    233    377    610    987
   1597   2584   4181   6765  10946  17711  28657
Press any key to quit....
```

图 89-1 实例 89 程序运行效果



## 实例解析

问题可以抽象成下列数学公式：

$$U_n = U_{n-1} + U_{n-2}$$

其中 $n$ 是项数( $n \geq 3$ )。

这就是著名的菲波那契数列，该数列的前几个数：1, 1, 2, 3, 5, 8, 13, 21…

菲波那契数列在程序中可以用多种方法进行处理。按照其通项递推公式，利用最基本的循环

控制就可以实现题目的要求。



## 程序代码

### 【程序 89】 兔子产子

//本实例源码参见光盘



## 归纳注释

在求解类似问题时，可将其归纳为通项公式，再利用通项公式求解会方便很多。

# 实例 90 数字移动



## 实例说明

在如图 90-1 中的 9 个点上，空出正中间的点，其余的点上任意填入数字 1~8。1 的位置固定不动，然后移动其余的数字，使 1 到 8 顺时针从小到大排列。移动的规律是只能将数字沿线移向空白的点。请编程显示数字移动过程。程序运行效果如图 90-2 所示。此程序可在 Visual C++ 中编译（见光盘）。

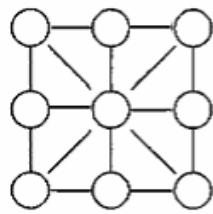


图 90-1 九宫图

```
CD "F:\zhongzi\WorkDir\CEexample\VC\90\Debug\90.exe"
*****
*          This is a program to Move Numbers.      *
*****
>> Input original order of digits 1~8: 8 5 6 3 2 1 7 4
>> The sorting process is as follows:
>> Step No. 0  8 5 6  4 0 3  7 1 2
>> Step No. 1  8 5 6  4 2 3  7 1 0
>> Step No. 2  8 5 6  4 2 0  7 1 3
>> Step No. 3  8 5 0  4 2 6  7 1 3
>> Step No. 4  8 0 5  4 2 6  7 1 3
>> Step No. 5  0 8 5  4 2 6  7 1 3
>> Step No. 6  4 8 5  0 2 6  7 1 3
>> Step No. 7  4 8 5  7 2 6  0 1 3
>> Step No. 8  4 8 5  7 0 6  2 1 3
>> Step No. 9  4 8 5  7 3 6  2 1 0
>> Step No.10  4 8 5  7 3 0  2 1 6
>> Step No.11  4 8 0  7 3 5  2 1 6
>> Step No.12  4 0 8  7 3 5  2 1 6
>> Step No.13  0 4 8  7 3 5  2 1 6
>> Step No.14  7 4 8  0 3 5  2 1 6
>> Step No.15  7 4 8  3 0 5  2 1 6
>> Step No.16  7 0 8  3 4 5  2 1 6
>> Step No.17  0 7 8  3 4 5  2 1 6
>> Step No.18  4 7 8  3 0 5  2 1 6
>> Step No.19  4 7 8  3 5 0  2 1 6
>> Step No.20  4 7 0  3 5 8  2 1 6
>> Step No.21  4 0 7  3 5 8  2 1 6
>> Step No.22  4 5 7  3 0 8  2 1 6
>> Step No.23  4 5 7  3 6 8  2 1 0
>> Step No.24  4 5 7  3 6 0  2 1 8
>> Step No.25  4 5 0  3 6 7  2 1 8
>> Step No.26  4 5 6  3 0 7  2 1 8

Press any key to quit....
```

图 90-2 实例 90 程序运行效果



## 实例解析

分析题目中的条件，要求利用中间的空白格将数字顺时针方向排列，且排列过程中只能借空的点来移动数字。问题的实质就是将矩阵外面的 8 个格看成一个环，8 个数字在环内进行排序，同于受题目要求的限制——只能将数字沿线移向空的点，所以要利用中间的空格进行排序，这样要求的排序算法与其他的不同。

观察中间的点，它是惟一一个与其他 8 个点有连线的点，即它是中心点。中心点活动的空间最大，它可以向 8 个方向移动，充分利用中心点这个特性是算法设计成功的关键。

在找到 1 所在的位置后，其余各个数字的正确位置就是固定的。可以按照下列算法从数字 2 开始，一个一个地来调整各个数字的位置。

(1) 确定数字 i 应处的位置。

(2) 从数字 i 应处的位置开始，向后查找数字 i 现在的位置。

(3) 若数字 i 现在的位置不正确，则将数字 i 从现在的位臵（沿连线）移向中间的空格，而将原有位臵空出；依次将现有空格前的所有元素向后移动……直到将 i 应处的位臵空出，把它移入空出的格。

从数字 2 开始使用以上过程，就可以完成全部数字的移动排序。

编程时要将矩阵的外边 8 个格看成一个环，且环的首元素是不定的，如果算法设计得不好，程序中就要花很多精力来处理环中元素的前后顺序问题。将题目中的  $3 \times 3$  矩阵用一个一维数组表示，中间的元素（第 4 号）刚好为空格。设计另一个指针数组，专门记录指针外 8 个格构成环时的连接关系。指针数组的每个元素依次记录环中数字在原来数组中对应的元素下标。这样通过指针数组将原来矩阵中复杂的环型关系表示成了简单的线性关系，从而大大地简化了程序设计。



## 程序代码

### 【程序 90】 数字移动

```
#include<stdio.h>
int a[]={0,1,2,5,8,7,6,3};      /*指针数组，依次存入矩阵中构成环的元素下标*/
int b[9];                      /*表示  $3 \times 3$  矩阵，b[4]为空格*/
int c[9];                      /*确定 1 所在的位置后，对环进行调整的指针数组*/
int count=0;                    /*数字移动步数计数器*/
void main()
{
    int i,j,k,t;
    void print();
    clrscr();
    puts("*****");
    puts("*          This is a program to Move Numbers.          *");
    puts("*****");
    printf(" >> Input original order of digits 1~8: ");
    for(i=0;i<8;i++)
        scanf("%d",&b[a[i]]);
    /*顺序输入矩阵外边的 8 个数字，矩阵元素的顺序由指针数组的元素 a[i] 控制*/
    printf(" >> The sorting process is as follows:\n");
    print();
}
```

```

for(t=-1,j=0;j<8&&t===-1;j++)           /*确定数字 1 所在的位置*/
    if(b[a[j]]==1) t=j;                   /*t 记录数字 1 所在的位置*/
for(j=0;j<8;j++)      /*调整环的指针数组,将数字 1 所在的位置定为环的首*/
    c[j]=a[(j+t)%8];
for(i=2;i<9;i++)          /*从 2 开始依次调整数字的位置*/
    /*i 为正在处理的数字,i 对应于环中的正确位置就是 i-1*/
    for(j=i-1;j<8;j++)      /*从 i 应处的正确位置开始顺序查找*/
        if(b[c[j]]==i&&j!=i-1)      /*若 i 不在正确的位置*/
    {
        b[4]=i;             /*将 i 移到中心的空格中*/
        b[c[j]]=0;print();    /*空出 i 原来所在的位置,输出*/
        for(k=j;k!=i-1;k--)   /*将空格以前到 i 的正确位置之间的数字依次向后移动一格*/
        {
            b[c[k]]=b[c[k-1]]; /*数字向后移动*/
            b[c[k-1]]=0;
            print();
        }
        b[c[k]]=i;           /*将中间的数字 i 移入正确的位置*/
        b[4]=0;             /*空出中间的空格*/
        print();
        break;
    }
    else if(b[c[j]]==i) break;      /*数字 i 在正确的位置*/
printf("\n Press any key to quit..."); 
getch();
}
void print(void)          /*按格式要求输出矩阵*/
{
    int c;
    printf(" >> Step No.%2d ",count++);
    for(c=0;c<9;c++)
        if(c%3==2) printf("%2d ",b[c]);
        else printf("%2d",b[c]);
    printf("\n");
}

```



## 归纳注释

本实例程序采用循环从 2 开始依次调整数字的位置，调整完毕，利用 print 函数实现输出。

# 实例 91 多项式乘法



## 实例说明

本实例实现多项式乘法。程序运行效果如图 91-1 所示。

```

TC
This is a polynomial multiplication program.
It calculate the product of two polynomials: P(x) and Q(x)
P(x)=Pm-1*x^(m-1)+Pm-2*x^(m-2)+...+P1*x+P0
Q(x)=Qn-1*x^(n-1)+Qn-2*x^(n-2)+...+Q1*x+Q0

>> Please input m (>=1): 7
>> Please input P6, P5, ... P1, P0 one by one:
9 -3 5 6 3 0 3

>> Please input n (>=1): 6
>> Please input Q5, Q4, ... Q1, Q0 one by one:
6 4 9 2 4 -4

The product of two polynomials R(x) is :
<12.000000*x^11> + <-10.000000*x^10> + <-23.000000*x^9> +
<43.000000*x^8> + <-22.000000*x^7> + <11.000000*x^6> +
<25.000000*x^5> + <-20.000000*x^4> + <-10.000000*x^3> +
<3.000000*x^2> + <0.000000*x^1> + <0.000000*x^0> +

Input the value of x: 5.7
The value of the R(5.700000) is: 27982541.0557899
Press any key to quit...

```

图 91-1 实例 91 程序运行效果



## 实例解析

求两个多项式：

$$P(x) = p_{m-1}x^{m-1} + p_{m-2}x^{m-2} + \dots + p_1x + p_0$$

$$Q(x) = q_{n-1}x^{n-1} + q_{n-2}x^{n-2} + \dots + q_1x + q_0$$

$$\text{的乘积多项式 } S(x) = s_{m+n-2}x^{m+n-2} + \dots + s_1x + s_0$$

可以按照如下的迭加公式计算结果的各项系数：

$$s_{i+j} = s_{i+j} + p_i q_j, \quad i = 0, 1, \dots, m-1, \quad j = 0, 1, \dots, n-1$$

在程序中，先定义两个一维数组  $p$ ,  $q$ ，从高到低来存储两个多项式的系数，并定义一维数组  $result$  来存储结果；然后调用  $npmul()$  子函数按照上面的公式所示算法逐项来完成对结果多项式各项系数的计算，最后在屏幕上计算结果。



## 程序代码

### 【程序 91】 多项式乘法

```

#include <stdio.h>
#include <math.h>
#define MAX 50
/* 下面的两个数组可以根据具体要求解的多项式来决定其值 */
static double p[6]={4,-6,3,1,-1,5}; /* 表示多项式 4x^5 - 6x^4 + 3x^3 + x^2 - x + 5 */
static double q[4]={3,2,-5,1}; /* 表示多项式 3x^3 + 2x^2 - 5x + 1 */
static double result[9]={0,0,0,0,0,0,0,0,0}; /* 存放乘积多项式 */
void npmul(p,m,q,n,s)
int m,n;
double p[],q[],s[];

```

```

{
    int i,j;
    for (i=0; i<=m-1; i++)
        for (j=0; j<=n-1; j++)
            s[i+j]=s[i+j]+p[i]*q[j];      /*迭代计算各项系数*/
    return;
}
double compute(s,k,x)                  /*计算所给多项式的值*/
double s[];
int k;
float x;
{
    int i;
    float multip = 1;
    double sum = 0;
    for (i=0;i<k;i++)
        multip = multip * x;           /*先求出x的最高次项的值*/
    for (i=k-1;i>=0;i--)
    {
        sum = sum + s[i] * multip;   /*依次从高到低求出相对应项的值*/
        if (x!=0)
            multip = multip / x; }
    return sum;
}
void main()
{
    int i,j,m,n;
    double px[MAX],qx[MAX],rx[MAX];
    float x;
    clrscr();
    for(i=0;i<MAX;i++)
        rx[i]=0;
    puts("      This is a polynomial multiplication program.");
    puts("It calculate the product of two polynomials: P(x) and Q(x)");
    puts("      P(x)=Pm-1*x^(m-1)+Pm-2*x^(m-2)+...+P1*x+P0");
    puts("      Q(x)=Qn-1*x^(n-1)+Qn-2*x^(n-2)+...+Q1*x+Q0");
    printf("\n >> Please input m (>=1): ");
    scanf("%d",&m);
    printf(" >> Please input P%d, P%d, ... P1, P0 one by one:\n",m-1,m-2);
    for(i=0;i<m;i++)
        scanf("%f",&px[i]);
    printf("\n >> Please input n (>=1): ");
    scanf("%d",&n);
    printf(" >> Please input Q%d, Q%d, ... Q1, Q0 one by one:\n",n-1,n-2);
    for(i=0;i<n;i++)
        scanf("%f",&qx[i]);
    npmul(p,m,q,n,rx);
    printf("\nThe product of two polynomials R(x) is :\n");
    for (i=m+n-1,j=0;i>=1;i--)          /*逐行逐项打印出结果多项式*/
    {
        printf(" (%f*x^%d) + ",rx[m+n-1-i],i-1);
        if(j==2)
        {printf("\n");
         j=0;      }
    }
}

```

```

        else j++;
    }
printf("\n");
printf("Input the value of x: ");
scanf("%f",&x);
printf("\nThe value of the R(%f) is: %13.7f",x,compute(rx,m+n-1,x));
puts("\n Press any key to quit...");
getch();
}

```



## 归纳注释

本程序采用迭代法计算两个多项式乘积，并求出了最终结果。

# 实例 92 产生随机数



## 实例说明

编程实现产生随机数。程序运行效果如图 92-1 所示。

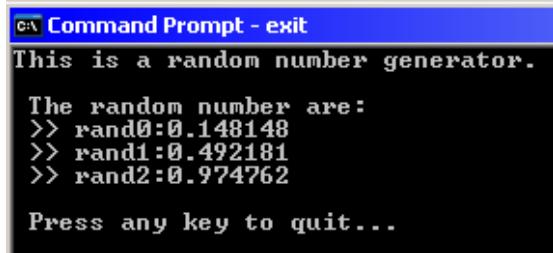


图 92-1 实例 92 程序运行效果



## 实例解析

众所周知，随机数在软件开发中是非常有用的。然而，在 DOS 系统中的很多编程语言中都不能得出令人满意的随机数，这些随机数都有以下几个缺点：值域范围小，易重复，故随机度不高；没有经过归一化，使用不便；其随机序列是固定的。

因此，虽然这些随机序列从内部看起来具有随机性，但就其整个序列而言却不是随机的，即每次调用程序生成的随机序列结果都是惟一的，这样的随机数又称为伪随机数。

本实例讲解一种简单的方法，可以生成每次不同的随机序列。迭代方程大都具有随机性，考虑如下迭代法：

$$f(n+1) = af(n)[1 - f(n)] \quad f(n) \text{ 在 } (0,1) \text{ 之间, } a \text{ 在 } (1,4) \text{ 之间}$$

上面的方程是从  $f(n)$  到  $f(n+1)$  上的映射。方程虽然看起来比较简单，但是当参数  $a$  发生变化时，迭代的结果却表现出“倍周期分叉”的复杂特性。利用这一特性，很容易产生高随机度的随机序列。因为方程对初值非常敏感，不同的初值对迭代的结果有很大的影响，所以可以让计算机自动产生一个变化范围较大的初值，直接赋给方程参数  $a$ 。这样处理后，不光是随机序列内部，而且随机序列本身也具有了随机性。由此，即可得到高随机度的随机数序列。



## 程序代码

### 【程序 92】 产生随机数

//本实例源码参见光盘



## 归纳注释

本实例程序中的初值是利用 DOS 的系统时钟产生的，并经过处理而得到。初值和产生随机序列的函数都被定义为 double 型。

# 实例 93 堆栈四则运算



## 实例说明

利用堆栈实现四则运算。程序运行效果如图 93-1 所示。

The screenshot shows a terminal window titled "F:\WorkDir\CEexample\VC\63\Debug\63.exe". The window displays the following text:

```
=====  
*          四则运算表达式求值程序          *  
*****  
* 使用方法:请从键盘上直接输入表达式,以回车键结束.如45*(12-2)[回车]*  
* 注0:不输入任何数而直接按[回车]键,将退出程序.  
* 注1:本程序暂时不接受除数字键及四则运算符之外的任何其它键盘输入.  
* 注2:本程序暂时只能处理正确的表达式,不支持输入负数.  
*****  
  
表达式1:6748*43+432*45-34*2+43*(225-432+3432)=448211  
表达式2:45342*2-45*3+22*45*(253-22)=319239  
表达式3:  
  
*****  
感谢使用!  
请按任意键退出...
```

图 93-1 实例 93 程序运行效果



## 实例解析

运行程序，首先由 main() 函数调用 Message() 函数显示提示信息。然后调用 EvaluateExpression() 函数计算四则运算表达式。

EvaluateExpression() 函数首先调用 InitStack() 函数建立两个初始栈，一个用来存放操作符串，另一个用来存放操作数串。然后循环调用 GetInput() 函数得到用户输入并计算表达式的值。过程如下：首先得到一个数，将这个数压入操作数栈。然后循环调用 GetInput() 函数，如果输入是运算符，而且优先级比操作符栈栈顶的优先级低，则将运算符压入操作符栈；如果是运算符，但优先级比操作符栈顶的优先级高，则将操作数栈顶的两个数出栈，用该操作符进行运算，把得到的结果压入操作数栈顶。当用户输入完后按回车键时，操作数栈栈顶的数就是最终的运算结果。



## 程序代码

### 【程序 93】 堆栈四则运算

```
//#include "stdafx.h"
#include "stdio.h"
#include "string.h"
#include "stdlib.h"
#include "conio.h"

#define TRUE 1
#define FALSE 0
#define STACK_INIT_SIZE 100/*存储空间初始分配量*/
#define STACKINCREMENT 20/*存储空间分配增量*/

typedef struct
{
    int *pBase; /*在构造之前和销毁之后,base 的值为 NULL*/
    int *pTop; /*栈顶指针*/
    int StackSize; /*当前已分配的存储空间,以元素为单位*/
}Stack;

typedef int BOOLEAN;

char Operator[8]="+-*()#"; /*合法的操作符存储在字符串中*/
char Optr; /*操作符*/
int Opnd=-1; /*操作符*/
int Result; /*操作结果*/

/*算符间的优先关系*/
char PriorityTable[7][7]=
{
    {'>', '>', '<', '<', '<', '>', '>', '>'},
    {'>', '>', '<', '<', '<', '>', '>', '>'},
    {'>', '>', '>', '>', '<', '>', '>', '>'},
    {'>', '>', '>', '>', '<', '>', '>', '>'},
    {'<', '<', '<', '<', '<', '=' , 'o' },
    {'>', '>', '>', '>', 'o', '>', '>' },
    {'<', '<', '<', '<', '<', 'o', '=' }
};

//数据对象的操作方法
//构造一个空栈,如果返回值为 0,则表示初始化失败
Stack InitStack()/*这是个效率低的方法*/
{
    Stack S;
    S.pBase=(int*)malloc(STACK_INIT_SIZE*sizeof(int));
    if(!S.pBase)
        /*内存分配失败*/
        printf("内存分配失败,程序中止运行\n");
        exit(-1);
}
else
```

```

{
    S.pTop=S.pBase;
    S.StackSize=STACK_INIT_SIZE;
}
return S;
}
//销毁栈 S
void DestoryStack(Stack *S)
{
    if(S->pBase)
    {
        free(S->pBase);
        S->pTop=S->pBase=NULL;
    }
}
//若栈不空,则用 e 返回 S 的栈顶元素
//注:由于应用的特殊,可以不检查栈是否为空
int GetTop(Stack S)
{
    return *(S.pTop-1);
}
//插入元素 e 为新的栈顶元素,如果成功则返回 1,否则返回 0
int Push(Stack *S,int e)
{
    if(S->pTop-S->pBase==S->StackSize)
    {//栈满,追加存储空间

        S->pBase=(int*)realloc(S->pBase,S->StackSize+STACKINCREMENT*sizeof(int));
        if(!S->pBase)
            return 0;//存储分配失败
        S->pTop=S->pBase+S->StackSize;
        S->StackSize+=STACKINCREMENT;
    }
    *(S->pTop++)=e;
    return 1;
}

int Pop(Stack *S,int *e)
{//若栈不空,则删除 S 的栈顶元素,用 e 返回其值,并返回 1,否则返回 0
    if(S->pTop==S->pBase)
        return 0;
    *e=*(--(S->pTop));
    return 1;
}

//主函数及其他函数的实现
//比较两个数学符号 operator_1,operator_2 的计算优先权
//在算符优先关系表中查找相应的关系并返回'<', '=' 或'>'
char CheckPriority(char operator_1,char operator_2)
{
    int i,j;//用来查询算符间优先关系表的下标
    //char *ptr;
    i=strchr(Operator,operator_1)-Operator;

```

```

    //找到传入操作符在字符串 Operators 中的相对位置
    j=strchr(Operator,operator_2)-Operator;
    //返回算符优先关系表中相应值
    return PriorityTable[i][j];
}

BOOLEAN IsOperator(char ch)
{//判断一个字符是否为操作符
    if(strchr(Operator,ch))
        return TRUE;
    else
        return FALSE;
}

//从键盘获得输入
void GetInput(void)
{
    char Buffer[20];//键盘输入缓冲区,用来处理输入多位数的情况
    char ch;//存放键盘输入
    int index;//存放 Buffer 的下标
    index=0;
    ch=getch();//从键盘读入一个字符
    while(ch!=13&&!IsOperator(ch))
    {//如果输入的字符是回车符或是操作符,循环结束
        if(ch>='0'&&ch<='9')
        {//将字符回显到屏幕
            printf("%c",ch);
            Buffer[index]=ch;
            index++;
        }
        ch=getch();
    }
    if(ch==13)
        Optr='#';//输入的表达式以回车符结束
    else
    {
        Optr=ch;
        printf("%c",ch);
    }
    if(index>0)
    {
        Buffer[index]='\0';
        Opnd=atoi((Buffer));
    }
    else
        Opnd=-1;//程序不支持输入负数,当 Opnd 为负数时,表示输入的字符为操作符
}
//计算形如 a+b 的表达式, theta 为操作符, a、b 为操作数
int Calc(int a,char theta,int b)
{
    switch(theta)
    {

```

```

case '+':
    return a+b;
case '-':
    return a-b;
case '*':
    return a*b;
default:
    if(b==0)//除数为零的情况
    {
        printf("除数不能为");
        return 0;//返回0用以显示
    }
    else
        return a/b;
}
/*
表达式求值*/
BOOLEAN EvaluateExpression()
{
    int temp;//临时变量
    char theta;//存放操作符的变量
    int itheta;//存放出栈的操作符的变量
    int a,b;//存放表达式运算时的中间值
    int topOpnd;//栈顶操作数
    char topOptr;//栈顶操作符

    Stack OPTR=InitStack();//操作符栈
    Stack OPND=InitStack();//操作数栈

    if(!Push(&OPTR,'#'))//操作符栈中的第一个为#字符
        return FALSE;

    GetInput();//从键盘获得输入

    while(Opdr != '#' || GetTop(OPTR) != '#')
    {//如果 Opdr 大于等于 0, 表示有操作数输入
        if(Opnd>=0)Push(&OPND,Opnd);
        switch(CheckPriority(GetTop(OPTR),Opdr))
        {
            case '<'://栈顶元素优先权低
                if(!Push(&OPTR,Opdr))return FALSE;
                GetInput();
                break;
            case '='://去掉括号并接收键盘输入
                Pop(&OPTR,&temp);GetInput();
                break;
            case '>'://退栈并将运算结果入栈
                //先用 itheta 得到操作符再赋给 theta
                Pop(&OPTR,&iTheta);
                Pop(&OPND,&b);
                Pop(&OPND,&a);
                theta = (char)(itheta);
                Push(&OPND,Calc(a,itheta,b));
                Opnd=-1;
                break;
        }
    }
}

```

```

    }

}

//本算法中，当输入只有一个操作数时就输入回车符
//OPND.pTop==OPND.pBase
//如果 OPND.pTop==OPND.pBase 并且 Opnd<0
//则说明用户未输入任何操作和操作符而直接输入[回车]
//程序直接退出运行
if(OPND.pTop==OPND.pBase&&Opnd<0)
{
    printf("\n\n 感谢使用!\n");
    exit(1);

}

else if(OPND.pTop==OPND.pBase)
    Result=Opnd;
else
{
    Result=GetTop(OPND);
    DestoryStack(&OPND);
    DestoryStack(&OPTR);
}
return TRUE;

}

void Message(void)
{
    printf("\n 四则运算表达式求值演示\n");
    printf("-----\n");
    printf(" 使用方法:请从键盘上直接输入表达式,以回车键结束.如 45*(12-2)[回车]\n");
    printf(" 注 0:不输入任何数而直接按[回车]键,将退出程序.\n");
    printf(" 注 1:本程序暂时不接受除数字键及四则运算符之外的任何其他键盘输入.\n");
    printf(" 注 2:本程序暂时只能处理正确的表达式,不支持输入负数.\n");
    printf("-----\n\n");
}

void main(void)
{
    int i;//用来一些说明性信息
    Message();
    for(i=1;;i++)
    {
        printf("表达式%d:",i);
        if(EvaluateExpression())
            printf("=%d\n",Result);
        else
            printf("计算中遇到错误\n");
    }
}

```



## 归纳注释

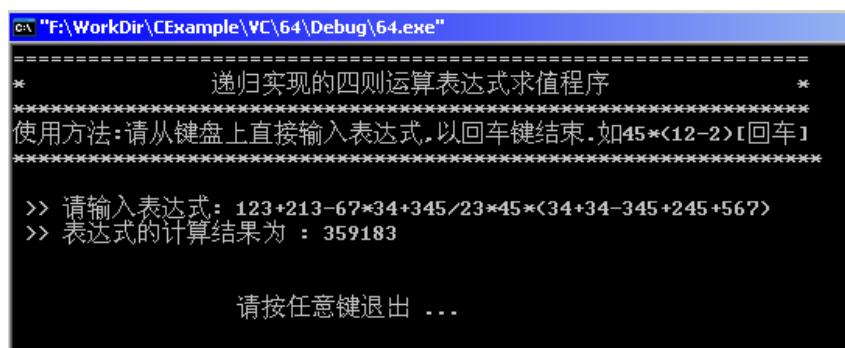
堆栈是一种先进后出的数据结构，对堆栈的操作有 pop 和 push 两种。pop 是将堆栈栈顶的数取

出来，同时栈中元素减少一个，栈顶元素的下一个成为新的栈顶元素。`push` 是将一个元素压入堆栈中，该元素成为新的栈顶元素，原来栈顶元素成为栈顶元素的下一个元素，栈中元素增加一个。

## 实例 94 递归整数四则运算

### 实例说明

利用递归方法实现整数的四则运算。程序运行效果如图 94-1 所示。



The screenshot shows a terminal window titled "F:\WorkDir\CEExample\VC\64\Debug\64.exe". The program displays a welcome message and usage instructions. It then prompts the user to enter an expression and shows the result. Finally, it asks the user to press any key to exit.

```
CD "F:\WorkDir\CEExample\VC\64\Debug\64.exe"
=====
*          递归实现的四则运算表达式求值程序      *
*****
使用方法:请从键盘上直接输入表达式,以回车键结束.如45*(12-2)[回车]
*****
>> 请输入表达式: 123+213-67*34+345/23*45*(34+34-345+245+567)
>> 表达式的计算结果为 : 359183

请按任意键退出 ...
```

图 94-1 实例 94 程序运行效果

### 实例解析

本程序实现了带括号的四则运算的功能，可以把它看作一个简单的计算器。

编程的基本思想如下：把四则运算的优先级做排列，最低的优先级是加减运算，较高的是乘除运算。为了方便起见，可以把运算数看作是和括号一样拥有最高优先级的运算符。在这里，用`<exp>`表示一个可以计算加减运算的表达式，`<term>`表示一个可以计算乘除的表达式，而`<factor>`则是括号表达式或一个数字。然后可以得到下面的表达式（其中“→”表示左边由右边构成，“|”表示或者）：

```
<exp> → <term> + | - <term>
<term> → <factor> * | / <factor>
<factor> → (<exp>) | Number
```

根据上面的表达式，可以很容易地写出相应的程序。

### 程序代码

#### 【程序 94】 递归整数四则运算

```
/*
对四则混合运算所提取的形式化表达式(生成式)
<exp> -> <term> { <addop> <term> }
<addop> -> + | -
<term> -> <factor> { <mulop> <factor> }
<mulop> -> * | /
<factor> -> ( <exp> ) | Number
```

```

*/



#include <stdio.h>
#include <stdlib.h>

char token; /*全局标志变量*/

/*递归调用的函数原型*/
int exp( void );
int term( void );
int factor( void );

void error( void ) /*报告出错信息的函数*/
{
    fprintf( stderr, "错误\n");
    exit( 1 );
}

void match( char expectedToken ) /*对当前的标志进行匹配*/
{
    if( token == expectedToken ) token = getchar(); /*匹配成功，获取下一个标志*/
    else error(); /*匹配不成功，报告错误*/
}
void Message(void)
{
    printf( "=====*\n" );
    printf("          递归实现的四则运算表达式求值程序          *\n" );
    printf( "*****\n" );
    printf( "使用方法:请从键盘上直接输入表达式,以回车键结束.如 45*(12-2)[回车]\n" );
    printf( "*****\n" );
}
main()
{
    int result; /*运算的结果*/
    Message();
    printf(" >> 请输入表达式: ");
    token = getchar(); /*载入第一个符号*/

    result = exp(); /*进行计算*/
    if( token == '\n' ) /*是否一行结束*/
        printf( " >> 表达式的计算结果为 : %d\n", result );
    else error(); /*出现了例外的字符*/
    puts( "\n\n          请按任意键退出 ... \n" );
    getch();
    return 0;
}

int exp( void )
{
    int temp = term(); /*计算比加减运算优先级别高的部分*/
    while(( token == '+' ) || ( token == '-' ))
        switch( token ) {
            case '+': match('+'); /*加法*/
                temp += term();

```

```

        break;
    case '-': match('-');
        temp -= term(); /*减法*/
        break;
    }
    return temp;
}

int term( void )
{
    int div; /*除数*/
    int temp = factor(); /*计算比乘除运算优先级别高的部分*/
    while(( token == '*' ) || ( token == '/' ))
    {
        switch( token ) {
        case '*': match('*'); /*乘法*/
            temp *= factor();
            break;
        case '/': match('/'); /*除法*/
            div = factor();
            if( div == 0 ) /*需要判断除数是否为0*/
            {
                fprintf( stderr, "除数为 0.\n" );
                exit(1);
            }
            temp /= div;
            break;
        }
    }
    return temp;
}

int factor( void )
{
    int temp;
    if( token == '(' ) /*带有括号的运算*/
    {
        match( '(' );
        temp = exp();
        match(')');
    }
    else if ( isdigit( token ) ) /*实际的数字*/
    {
        ungetc( token, stdin ); /*将读入的字符退还给输入流*/
        scanf( "%d", &temp ); /*读出数字*/
        token = getchar(); /*读出当前的标志*/
    }
    else error(); /*不是括号也不是数字*/
    return temp;
}

```



## 归纳注释

本实例采用的方法叫做递归下降法。所谓下降，就是从最宏观的部分开始，逐步细化。

程序执行的时候，由用户输入一个表达式，然后主函数调用 `exp()` 函数来计算表达式的值，最

终输出结果。本程序只是一个最简单的实例，没有处理表达式中有空格的情况和非整数的情况。如果读者理解了这种编程的思想，增加这些功能是轻而易举的。

## 实例 95 复平面作图

### 实例说明

DOS 文件模式下的标准窗口为 25 行，每行 80 个字符。在本例中，将使用文本模式，在复平面上打印输出给定的复数点集。程序运行效果如图 95-1 所示。

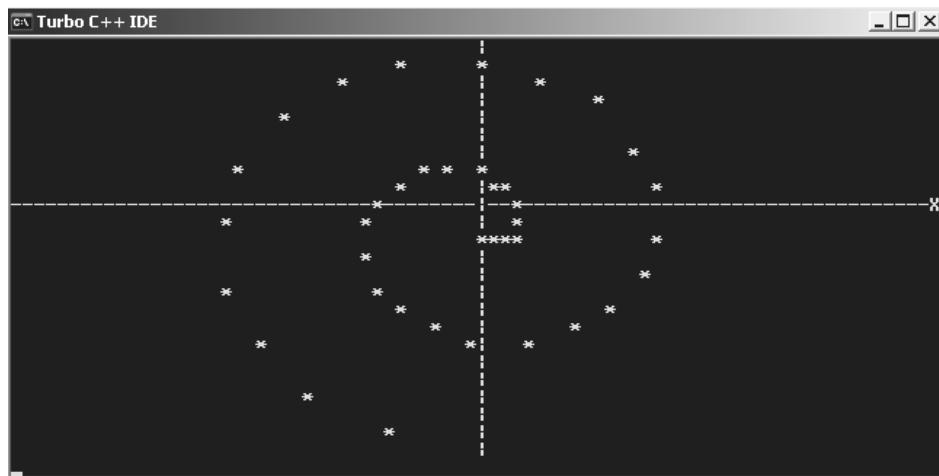


图 95-1 实例 95 程序运行效果

### 实例解析

设给定了  $n$  个复数

$$z_k = x_k + y_k \quad i, k = 0, 1, \dots, n-1$$

其中  $i = \sqrt{-1}$ 。

首先在屏幕上建立一个直角坐标系  $xOy$ ，其  $x$  值与  $y$  值的范围分别为  $[-12, 12]$  和  $[-40, 40]$ 。注意这里是将  $x$  轴作为纵轴， $y$  作为横轴。即所有的复数点均打印在宽 25，长 80 的屏幕上，坐标原点在屏幕中心。

本例中要绘制的图形为  $\rho = 2 + \theta e^{i\theta}$ ，在绘制的时候，用 “-”，“|” 表示坐标轴，“\*” 表示函数点。为了能以熟悉的直角坐标系来描绘复平面，在这里使用隶莫佛定理：

$$re^{i\theta} = r \cos \theta + ir \sin \theta$$

当使用  $x$  表示实部， $y$  表示虚部的时候，就得到了直角坐标与复坐标之间的关系：

$$\begin{cases} x = 2 + \theta \cos \theta \\ y = \theta \sin \theta \end{cases}$$



## 程序代码

### 【程序 95】 复平面作图

//本实例源码参见光盘



## 归纳注释

在程序中，为了实现在屏幕上的输出，定义了二维数组 `screen[25][80]`，用来对应屏幕上的点，并且使用 `memset(screen," ",25*80)` 这个函数将所有的字符都设置成空格，这样在屏幕上就是空白显示，只要将需要显示出来的点设置成可视字符即可。

# 实例 96 绘制彩色抛物线



## 实例说明

本程序实现了根据给定的起点及旋转的角度，用指定的颜色绘制抛物线的功能。程序运行结果如图 96-1 所示。

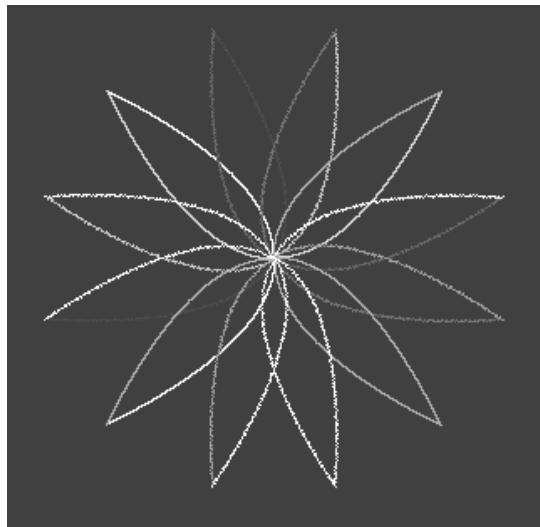


图 96-1 实例 96 程序运行结果



## 实例解析

抛物线插补法的过程简述如下（有兴趣的读者可参阅“数值分析方法”类书籍）：

假设抛物线方程是  $y = ax^2$ ，要画出从起点  $(x_1, y_1)$  到终点  $(-x_1, y_1)$  的部分。其中  $(x_1, y_1)$  为相对于抛物线顶点的坐标。

首先设置插补的初始信息：当  $x_1 < 0$  时水平方向的移动增量  $dx=1$ ，当  $x_1 > 0$  时水平方向的移动增量  $dx=-1$ ；同样  $y$  也类似。

其他插值信息如下：

$$\begin{aligned}fx &= -(2x_1\Delta x + 1.0) \times y_1, \\rx &= -2y_1, \\fy &= x_1^2 \Delta y, \\f &= 0.\end{aligned}$$

进行插补时，首先显示起点 $(x_1, y_1)$ ，并令 $x=x_1$ ,  $y=y_1$ 。如果 $f \geq 0$ , 则 $x=x+\Delta x$ , 就显示像点 $(x, y)$ , 并且令 $f=f-|fx|$ ,  $fx=fx+rx$ 。此时如果 $fx \times (fx-rx) \leq 0$ , 则 $\Delta y=-dy$ ,  $fy=-fy$ ,  $f=-f$ 。如果 $f < 0$ , 则 $y=y+\Delta y$ , 显示像点 $(x, y)$ , 且 $f=f+|fy|$ 。

重复进行以上过程，直到到达终点结束。

绘制旋转的抛物线时，要根据旋转角度 $t$ 对像点进行调整，经过调整后的实际显示的像点方程为 $x_2 = x \cos t - y \sin t$ ,  $y_2 = x \sin t + y \cos t$

运行该程序后，首先由主程序调用绘制抛物线函数依次画 4 个互成 $90^\circ$  的抛物线，然后依次画出互成 $30^\circ$  的各色抛物线 12 个，最后退出主函数，程序结束。



## 程序代码

### 【程序 96】 绘制彩色抛物线

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

/*画抛物线的子函数 spara()*/
/*row,col 代表抛物线顶点的坐标, x1,y1 是抛物线起点相对顶点的坐标*/
/*t 为抛物线绕顶点旋转的角度*/
void spara(row,col,x1,y1,t,color)
int row,col,x1,y1,t,color;
{
    int n,dx,dy,x,y,x2,y2;
    double ct,st;
    double f,fx,fy,b,a,rx;
    st=(double)t*3.1415926/180.0; /*把角度转化为弧度*/
    ct=cos(st); st=sin(st);
    n=abs(x1)+abs(y1); n=n+n;
    dx=1; dy=1; f=0.0; /*初始化工作*/
    if (x1>0) dx=-1;
    if (y1>0) dy=-1;
    a=y1; b=x1; b=b*b;
    rx=-a-a; fx=2.0*x1*dx+1.0;
    fy=-a*fx; fy=b;
    if (dy<0) fy=-fy;
    x=x1; y=y1;
    x2=(double)x*ct-(double)y*st+2000.5;
    y2=(double)x*st+(double)y*ct+2000.5;
    x2=x2-2000; y2=y2-2000;
    putpixel(row-y2,col+x2,color);
    while (n>0) /*具体的运算法则见上面的公式*/
    {
        n=n-1;
        if (f>=0.0)
        { x=x+dx;
```

```

x2=(double)x*ct-(double)y*st+2000.5;
y2=(double)x*st+(double)y*ct+2000.5;
x2=x2-2000; y2=y2-2000;
putpixel(row-y2,col+x2,color);
if (fx>0.0) f=f-fx;
else f=f+fx;
fx=fx+rx;
if (fx==0.0||(fx<0.0&&fx-rx>0.0)|| (fx>0.0&&fx-rx<0.0))
{ dy=-dy; fy=-fy; f=-f; }
}
else
{ y=y+dy;
x2=(double)x*ct-(double)y*st+2000.5;
y2=(double)x*st+(double)y*ct+2000.5;
x2=x2-2000; y2=y2-2000;
putpixel(row-y2,col+x2,color);
if (fy>0.0) f=f+fy;
else f=f-fy;
}
}
return;
}

void main()
{
    int i,color;
    int gdriver = DETECT , gmode;
    color = 1;
    registerbgidriver(EGAVGA_driver);
    initgraph(&gdriver,&gmode,"..\bgi");           /*初始化图形界面*/
    for (i=1;i<=4;i++)                         /*先画出 4 个互成 90° 的抛物线*/
    {
        spara(200,200,100,100,i*90,color);
        color+=3;
        getch();
    }
    color = 1;
    for (i=1;i<=11;i++)                       /*再画 12 个互成 30° 的抛物线*/
    {
        spara(200,200,100,100,i*30,color);
        color++;
    }
    getch();
    closegraph();
    return;
}

```



## 归纳注释

本例中首次使用了 Turbo C 的图形库函数，编程环境具体的配置和图形函数的详细使用方法参见【第四部分 图形篇】中的说明。

# 实例 97 绘制正态分布曲线

## 实例说明

实现绘制正态分布曲线。程序运行结果如图 97-1 所示，其中 Alpha 为 5，θ 为 80。

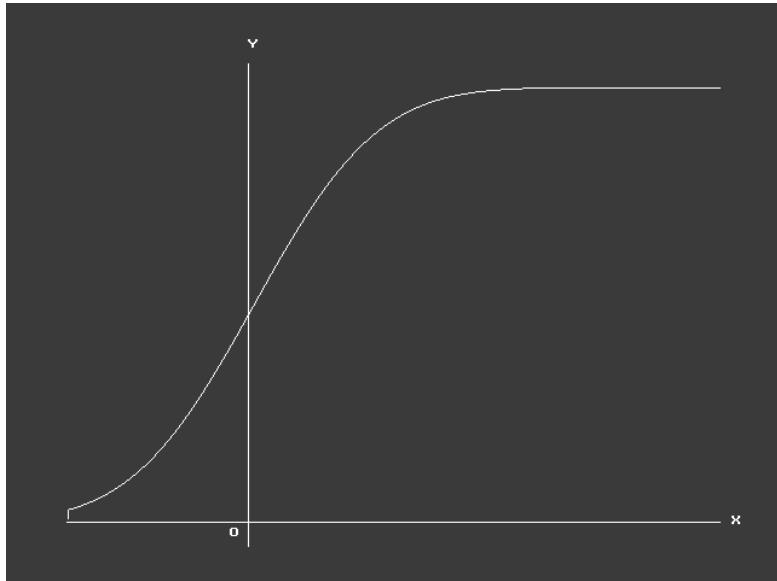


图 97-1 实例 97 程序运行结果

## 实例解析

正态分布是实际应用中极为广泛的分布函数之一，也称为高斯分布。正态函数的定义如下：

$$P(a, \sigma, x) = \frac{1}{\sqrt{2\pi}\sigma} \int_{-\infty}^x e^{-\frac{(t-a)^2}{2\sigma^2}} dt$$

其中  $a$  为随机变量的数学期望（平均值）； $\sigma (\sigma > 0)$  是随机变量方差的根。但是在实际使用过程中，因为这是一个广义积分式，所以很难用程序计算，而且计算量较大。通常使用误差函数来计算：

$$P(a, \sigma, x) = \frac{1}{2} + \frac{1}{2} \operatorname{erf}\left(\frac{x-a}{\sqrt{2}\sigma}\right)$$

其中误差函数定义如下：

$$\operatorname{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt$$

## 程序代码

### 【程序 97】 绘制正态分布曲线

```
#include "stdio.h"
#include "math.h"
#include "graphics.h"
```

```

double lgam1(x) /*Gamma 函数的计算*/
double x;
{
    int i;
    double y,t,s,u;
    static double a[11]={ 0.0000677106,-0.0003442342,
        0.0015397681,-0.0024467480,0.0109736958,
        -0.0002109075,0.0742379071,0.0815782188,
        0.4118402518,0.4227843370,1.0};
    if (x<=0.0)
    { printf("err**x<=0!\n"); return(-1.0); }
    y=x;
    if (y<=1.0)
    {
        t=1.0/(y*(y+1.0)); y=y+2.0;
    }
    else if (y<=2.0)
    {
        t=1.0/y; y=y+1.0;
    }
    else if (y<=3.0) t=1.0;
    else
    {
        t=1.0;
        while (y>3.0)
            { y=y-1.0; t=t*y; }
    }
    s=a[0]; u=y-2.0;
    for (i=1; i<=10; i++)
        s=s*u+a[i];
    s=s*t;
    return(s);
}
double lgam2(a,x) /*不完全 Gamma 函数*/
double a,x;
{
    int n;
    double p,q,d,s,s1,p0,q0,p1,q1,qq;
    if ((a<=0.0)|| (x<0.0))
    { if (a<=0.0) printf("err**a<=0!\n");
    if (x<0.0) printf("err**x<0!\n");
    return(-1.0);
    }
    if (x+1.0==1.0) return(0.0);
    if (x>1.0e+35) return(1.0);
    q=log(x); q=a*q; qq=exp(q);
    if (x<1.0+a)
    {
        p=a; d=1.0/a; s=d;
        for (n=1; n<=100; n++)
        {
            p=1.0+p; d=d*x/p; s=s+d;
            if (fabs(d)<fabs(s)*1.0e-07)
            {
                s=s*exp(-x)*qq/lgam1(a);
                return(s);
            }
        }
    }
}
```

```

        }
    }
}
else
{
    s=1.0/x; p0=0.0; p1=1.0; q0=1.0; q1=x;
    for (n=1; n<=100; n++)
    {
        p0=p1+(n-a)*p0; q0=q1+(n-a)*q0;
        p=x*p0+n*p1; q=x*q0+n*q1;
        if (fabs(q)+1.0!=1.0)
        {
            s1=p/q; p1=p; q1=q;
            if (fabs((s1-s)/s1)<1.0e-07)
            {
                s=s1*exp(-x)*qq/lgaml(a);
                return(1.0-s);
            }
            s=s1;
        }
        p1=p; q1=q;
    }
}
printf("a too large !\n");
s=1.0-s*exp(-x)*qq/lgaml(a);
return(s);
}

double lerrf(x) /*误差函数*/
double x;
{
    double y;
    if (x>=0.0)
        y=lgam2(0.5,x*x);
    else
        y=-lgam2(0.5,x*x);
    return(y);
}
double lgass(a,d,x) /*正态分布函数*/
double a,d,x;
{
    double y;
    if (d<=0.0) d=1.0e-10;
    y=0.5+0.5*lerrf((x-a)/(sqrt(2.0)*d));
    return(y);
}

main()
{
    int i;
    double j;
    double a, d;

    int gdriver = DETECT, gmode;

```

```

clrscr();
printf("This program will draw the Normal Distribution Graph.\n");
printf("Please input the mathematical expectation (Alpha): ");
scanf("%lf", &a );
printf("Please input the variance (Sita >0): ");
scanf("%lf", &d );
/*registerbgidriver( EGAVGA_driver );*/
initgraph( &gdriver, &gmode, "e:\tc\bgi" );

setbkcolor( BLUE );
moveto( 50, 430 );
lineto( 590, 430 );
outtextxy( 600, 425, "X" );
moveto( 200, 50 );
lineto( 200, 450 );
outtextxy( 200, 30, "Y" );
outtextxy( 185, 435, "O" );

setcolor( RED );
moveto( 51, 430 - 100 * lgass( a, d, -150.0 ) );
for( i = 51; i <= 590; i++ )
{
    j = 430 - 360 * lgass( a, d, (double)(i-200) );
    lineto( i, j );
}
getch();
closegraph();
}

```



## 归纳注释

本实例根据输入的数学期望和方差进行正态分布曲线的绘制。在绘制曲线时，采用图形模式。

# 实例 98 求解非线性方程



## 实例说明

用二分法求解非线性方程  $f(x)=0$  在指定区间  $[a,b]$  内的实根。程序运行结果如图 98-1 所示。

```

TC
This is a program to solve Nonlinear function
by Binary Divisive Procedure.

The Nonlinear function is:
f(x)=((((x-5.0)*x+3.0)*x+1.0)*x-7.0)*x+7.0)*x-20.0

>> Solve successfully!
>> The results are:
>> The function has 2 roots, they are:
>> x(0)=-1.402463e+00
>> x(1)= 4.333755e+00

Press any key to quit...

```

图 98-1 实例 98 程序运行结果



## 实例解析

从端点  $x_0=a$  开始, 以  $h$  为步长, 逐步往后进行搜索。对于每一个子区间  $[x_i, x_{i+1}]$  (其中  $x_{i+1}=x_i+h$ )

如果  $f(x_i)=0$ , 那么  $x_i$  为一个实根, 并且从  $x_i+h/2$  开始再往后搜索。如果  $f(x_{i+1})=0$ , 那么  $x_{i+1}$  为一个实根, 并且从  $x_{i+1}+h/2$  开始再往后搜索。如果  $f(x_i)f(x_{i+1})>0$ , 那么说明当前子区间内无实根, 从  $x_{i+1}$  开始再往后搜索。如果  $f(x_i)f(x_{i+1})<0$ , 则说明当前子区间内有实根, 这时要反复将子区间减半, 直到发现一个实根, 或者子区间长度划分到了小于  $\text{eps}$  为止。在后一种情况下, 取子区间的中点作为方程的一个实根, 然后再从  $x_{i+1}$  开始往后搜索, 其中  $\text{eps}$  为预先给定的精度要求。

以上过程一直进行到区间右端点  $b$  为止。

函数 BinSearchRoot 实现了上述功能。需要指出的是, 程序中还需要一个函数来计算对于每个给定的点, 多项式对应的值。这里使用函数 Equation 进行求值。在求值的时候, 为了提高效率, 没有使用直接带入求值, 而是使用霍纳法求值。

举例说明霍纳法: 要求  $3x^3-5x^2+x+1$ , 就先把表达式转化为  $((3x-5)x+1)x+1$ 。在使用第一种方法时需要作  $3+2+1=6$  次乘法, 3 次加减法; 而第二种方法只需要做 4 次乘法, 3 次加减法, 大大减少了计算量。



## 程序代码

### 【程序 98】 求解非线性方程

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

int BinSearchRoot(a,b,h,eps,x,m) /*用二分法计算非线性方程的实根*/
int m;
/*参数意义
a 所求的根的下界
b 所求的根的上界, 即所求的根落在区间 [a,b]之内
h 递进的步长
eps 精度
x 根的值
m 预计的根的个数*/
double a,b,h,eps,x[];
{
    extern double Equation(); /*求解的非线性方程*/
    int n,js;
    double z,y,z1,y1,z0,y0;
    n=0; z=a; y=Equation(z);
    while ((z<=b+h/2.0)&&(n!=m)) /*对给定步长的子区间进行搜索*/
    {
        if (fabs(y)<eps) /*当前的判定点是方程的根*/
        {
            n=n+1;
            x[n-1]=z;
            z=z+h/2.0;
            y=Equation(z);
        }
        else /*当前点不是方程的根*/
    }
```

```

{
    z1=z+h;
    y1=Equation(z1);
    if (fabs(y1)<eps) /*下一个点是方程的根*/
    {
        n=n+1;
        x[n-1]=z1;
        z=z1+h/2.0;
        y=Equation(z);
    }
    else if (y*y1>0.0) /*该区间内无根*/
    { y=y1; z=z1; }
    else /*该区间内有根*/
    {
        js=0; /*标志, 0 表示未找到根, 1 表示已经确定了根*/
        while (js==0)
        {
            if (fabs(z1-z)<eps) /*区间的长度小于给定的精度, 可以当作已经找到了根*/
            {
                n=n+1;
                x[n-1]=(z1+z)/2.0; /*把区间的中位置值作为根*/
                z=z1+h/2.0; /*把寻找的位置放到下一个区间内*/
                y=Equation(z);
                js=1; /*在当前区间内已经找到了根*/
            }
            else /*区间比给定的精度大, 则进行二分*/
            {
                z0=(z1+z)/2.0; /*区间二分*/
                y0=Equation(z0);
                if (fabs(y0)<eps) /*z0 位置为根*/
                {
                    x[n]=z0;
                    n=n+1;
                    js=1;
                    z=z0+h/2.0;
                    y=Equation(z);
                }
                else if ((y*y0)<0.0) /*[z,z0]内有根*/
                { z1=z0; y1=y0; }
                else { z=z0; y=y0; }
            }
        }
    }
}
return(n); /*返回根的个数*/
}

main()
{
    int i,n;
    static int m=6;
    static double x[6];
    clrscr();
    puts("This is a program to solve Nonlinear function\n by Binary Divisive

```

```

Procedure.");
puts("\n The Nonlinear function is:");
puts("\n f(x)=((((x-5.0)*x+3.0)*x+1.0)*x-7.0)*x+7.0)*x-20.0\n");
n=BinSearchRoot(-2.0,5.0,0.2,0.000001,x,m);
puts("\n >> Solve successfully!\n >> The results are:");
printf(" >> The function has %d roots, they are:\n",n);/*输出根的个数*/
for (i=0; i<=n-1; i++)
    printf(" >> x(%d)=%13.7e\n",i,x[i]);
printf("\n Press any key to quit...\n");
getch();
}

double Equation(x)
double x;
{
    double z;
    z=((((x-5.0)*x+3.0)*x+1.0)*x-7.0)*x+7.0)*x-20.0;
    return(z);
}

```



## 归纳注释

在使用二分发时，要注意步长  $h$  的选择，如果步长取得过大，可能会导致某些实根的丢失；如果选得过小，会增加计算工作量。

# 实例 99 实矩阵乘法运算



## 实例说明

求  $m \times n$  阶实矩阵  $A$  与  $n \times k$  阶实矩阵  $B$  的乘积矩阵  $C=AB$ 。程序运行结果如图 99-1 所示。

```

C:\Command Prompt - exit
This is a real-matrix-multiplication program.
It calculate the two matrixes C<m*k>=A<m*n>B<n*k>.
>> Please input the number of rows in A, m= 5
>> Please input the number of cols in A, n= 4
>> Please input the number of cols in B, k= 3
>> Please input the 20 elements in A one by one:
1 2 3 4 5 -5 0 3 4 4 3 3 9 8 4 3 2 4 3 2
>> Please input the 12 elements in B one by one:
1 2 3 4 9 4 3 2 4 3 2 0
>> The result of C<5*3>=A<5*4>B<4*3> is:
30.00000 34.00000 23.00000
-6.00000 -29.00000 -5.00000
38.00000 56.00000 40.00000
62.00000 104.00000 75.00000
33.00000 50.00000 34.00000
Press any key to quit...

```

图 99-1 实例 99 程序运行结果



## 实例解析

矩阵相乘遵循如下法则：

- (1)  $A$  的列数与  $B$  的行数必须相同；
- (2)  $A_{m \times n}$  与  $B_{n \times k}$  的乘积为  $m \times k$  的矩阵；
- (3) 结果矩阵  $C$  中每个元素  $c_{ij}$  的值是  $A$  中  $i$  行的元素与  $B$  中  $j$  列的元素的乘积和。用公式表示就是  $C_{ij} = \sum_{t=0}^{n-1} a_{it} b_{tj}, i = 0, 1, \dots, m-1, j = 0, 1, \dots, k-1$ 。

程序中最主要的函数是 void MatrixMul(a,b,m,n,k,c)，其形式参数说明如下：

- (1)  $a$  为双精度实型二维数组，体积为  $m \times n$ ，存放矩阵  $A$  的元素；
- (2)  $b$  为双精度实型二维数组，体积为  $n \times k$ ，存放矩阵  $B$  的元素；
- (3)  $m$  为整型变量，矩阵  $A$  的行数，也是乘积矩阵  $C=AB$  的行数；
- (4)  $n$  为整型变量，矩阵  $A$  的列数，也是  $B$  矩阵的行数；
- (5)  $k$  为整型变量，矩阵  $B$  的列数，也是乘积矩阵  $C=AB$  的列数；
- (6)  $c$  为双精度实型二维数组，体积为  $m \times k$ ，存放乘积矩阵  $C=AB$  的元素。



## 程序代码

### 【程序 99】 实矩阵乘法运算

```
#include "stdio.h"
#define MAX 255
void MatrixMul(a,b,m,n,k,c) /*实矩阵相乘*/
int m,n,k; /*m矩阵A的行数, n矩阵B的行数, k矩阵B的列数*/
double a[],b[],c[]; /*a为A矩阵, b为B矩阵, c为结果, 即c = AB */
{
    int i,j,l,u;
    /*逐行逐列计算乘积*/
    for (i=0; i<=m-1; i++)
        for (j=0; j<=k-1; j++)
        {
            u=i*k+j; c[u]=0.0;
            for (l=0; l<=n-1; l++)
                c[u]=c[u]+a[i*n+l]*b[l*k+j];
        }
    return;
}

main()
{
    int i,j,m,n,k;
    double A[MAX];
    double B[MAX];
    double C[MAX];
    for(i=0;i<MAX;i++)
        C[i]=1.0;
    clrscr();
    puts("This is a real-matrix-multiplication program.\n");
    puts("It calculate the two matrixes C(m*k)=A(m*n)B(n*k).\n");
}
```

```

printf(" >> Please input the number of rows in A, m= ");
scanf("%d",&m);
printf(" >> Please input the number of cols in A, n= ");
scanf("%d",&n);
printf(" >> Please input the number of cols in B, k= ");
scanf("%d",&k);
printf(" >> Please input the %d elements in A one by one:\n",m*n);
for(i=0;i<m*n;i++)
    scanf("%lf",&A[i]);
printf(" >> Please input the %d elements in B one by one:\n",n*k);
for(i=0;i<n*k;i++)
    scanf("%lf",&B[i]);

MatrixMul(A,B,m,n,k,C); /*计算 C 的结果*/
/*格式化输出结果*/
printf("\n >> The result of C(%d*%d)=A(%d*%d)B(%d*%d) is:\n",m,k,m,n,n,k);
for (i=0; i<m; i++)
{
    for (j=0; j<k; j++)
        printf("%10.5f    ",C[i*k+j]);
    printf("\n");
}
printf("\n Press any key to quit...\n");
getch();
return 0;
}

```



## 归纳注释

程序运行时，要求用户输入矩阵  $A$ 、 $B$  的行列数，并依次输入相应的元素，计算后输出两矩阵相乘的结果。

# 实例 100 求解线性方程



## 实例说明

用高斯（Guass）消去法求解  $N$  阶线性代数方程组  $Ax=B$ 。程序运行结果如图 100-1 所示。

```

C:\TC
This is a program to solve N order linear equation set Ax=B.
It use Guass Elimination Method to solve the equation set:
a<0,0>x0+a<0,1>x1+a<0,2>x2+...+a<0,n-1>xn-1=b0
a<1,0>x0+a<1,1>x1+a<1,2>x2+...+a<1,n-1>xn-1=b1
...
a<n-1,0>x0+a<n-1,1>x1+a<n-1,2>x2+...+a<n-1,-1>xn-1=bn-1
>> Please input the order n <>1>: 4
>> Please input the 16 elements of matrix A<4*4> one by one:
0.2368 0.2471 0.2568 1.2671 0.1968 0.2071 1.2168 0.2271
0.1581 1.1675 0.1768 0.1871 1.1161 0.1254 0.1397 0.1490
>> Please input the 4 elements of matrix B<4*1> one by one:
1.8471 1.7471 1.6471 1.5471
>> The solution of Ax=B is x<4*1>:
x<0>=1.040577 x<1>=0.987051 x<2>=0.935040 x<3>=0.881282
Press any key to quit...

```

图 100-1 实例 100 程序运行结果



## 实例解析

高斯消去法解线性代数方程的基本原理如下。

对于线性方程组：

$$\begin{cases} a_{0,0}x_0 + a_{0,1}x_1 + a_{0,2}x_2 + \dots + a_{0,n-1}x_n = b_0 \\ a_{1,0}x_0 + a_{1,1}x_1 + a_{1,2}x_2 + \dots + a_{1,n-1}x_n = b_1 \\ \dots \\ a_{n-1,0}x_0 + a_{n-1,1}x_1 + a_{n-1,2}x_2 + \dots + a_{n-1,n-1}x_n = b_{n-1} \end{cases}$$

其中系数矩阵为  $A$ ，未知量向量为  $X$ ，值向量为  $B$ 。计算的方法分为两步进行。

第 1 步，消去过程，对于  $k$  从 0 到  $n-2$  做以下 3 步。

从系数矩阵  $A$  的第  $k$  行、第  $k$  列开始的右下角子阵中选取绝对值最大的元素，并通过行交换与列交换把它交换到主元素（即对角线元素）的位置上。

归一化：  $\begin{cases} a_{kj} / a_{kk} \rightarrow a_{kj} & j = k+1, \dots, n-1 \\ b_k / a_{kk} \rightarrow b_k \end{cases}$

消去：  $\begin{cases} a_{ij} - a_{ik}a_{kj} \rightarrow a_{ij} & j = k+1, \dots, n-1 \\ b_j - a_{ik}b_k \rightarrow b_j & j = k+1, \dots, n-1 \end{cases}$

第 2 步，回代过程。

$$\begin{aligned} b_{n-1} / a_{n-1,n-1} &\rightarrow x_{n-1} \\ b_j - \sum_{j=i+1}^{n-1} a_{ij}x_j &\rightarrow x_i & i = n-2, \dots, 1, 0 \end{aligned}$$

最后对解向量中的元素顺序进行调整。



## 程序代码

### 【程序 100】 求解线性方程

```
#include "stdlib.h"
#include "math.h"
#include "stdio.h"

#define MAX 255

int Guass(a,b,n)
int n;
double a[],b[];
{
    int *js,l,k,i,j,is,p,q;
    double d,t;
    js=malloc(n*sizeof(int));
    l=1;
    for (k=0;k<=n-2;k++)
    {

```

```

d=0.0;
/*下面是换主元部分，即从系数矩阵 A 的第 k 行，第 k 列之下的部分选出绝对值最大的元，交换到对角线上。*/
for (i=k;i<=n-1;i++)
    for (j=k;j<=n-1;j++)
    {
        t=fabs(a[i*n+j]); /*fabs()定义在 math.h 中，含义是求一个浮点数的绝对值*/
        if (t>d) { d=t; js[k]=j; is=i; }
    }
if (d+1.0==1.0) l=0; /*主元为 0*/
/*主元不为 0 的时候*/
else
{
    if (js[k]!=k)
        for (i=0;i<=n-1;i++)
        {
            p=i*n+k; q=i*n+js[k];
            t=a[p]; a[p]=a[q]; a[q]=t;
        }
    if (is!=k)
    {
        for (j=k;j<=n-1;j++)
        {
            p=k*n+j; q=is*n+j;
            t=a[p]; a[p]=a[q]; a[q]=t;
        }
        t=b[k]; b[k]=b[is]; b[is]=t;
    }
}
if (l==0)
{
    free(js); printf("fail\n");
    return(0);
}
d=a[k*n+k];
/*下面为归一化部分*/
for (j=k+1;j<=n-1;j++)
{
    p=k*n+j; a[p]=a[p]/d;
}
b[k]=b[k]/d;
/*下面为矩阵 A,B 消元部分*/
for (i=k+1;i<=n-1;i++)
{
    for (j=k+1;j<=n-1;j++)
    {
        p=i*n+j;
        a[p]=a[p]-a[i*n+k]*a[k*n+j];
    }
    b[i]=b[i]-a[i*n+k]*b[k];
}
}

d=a[(n-1)*n+n-1];

```

```

/*矩阵无解或有无限多解*/
if (fabs(d)+1.0==1.0)
{
    free(js); printf("该矩阵为奇异矩阵\n");
    return(0);
}
b[n-1]=b[n-1]/d;
/*下面为迭代消元*/
for (i=n-2;i>=0;i--)
{
    t=0.0;
    for (j=i+1;j<=n-1;j++)
        t=t+a[i*n+j]*b[j];
    b[i]=b[i]-t;
}
js[n-1]=n-1;
for (k=n-1;k>=0;k--)
    if (js[k]!=k)
        { t=b[k]; b[k]=b[js[k]]; b[js[k]]=t; }
free(js);
return(1);
}

main()
{
    int i,n;
    double A[MAX];
    double B[MAX];
    clrscr();
    puts("This is a program to solve N order linear equation set Ax=B.");
    puts("\n It use Guass Elimination Method to solve the equation set:");
    puts("\n     a(0,0)x0+a(0,1)x1+a(0,2)x2+...+a(0,n-1)xn-1=b0");
    puts("     a(1,0)x0+a(1,1)x1+a(1,2)x2+...+a(1,n-1)xn-1=b1");
    puts("     ....");
    puts("     a(n-1,0)x0+a(n-1,1)x1+a(n-1,2)x2+...+a(n-1,-1)xn-1=bn-1\n");
    printf(" >> Please input the order n (>1): ");
    scanf("%d",&n);
    printf(" >> Please input the %d elements of matrix A(%d*d) one by
one:\n",n*n,n,n);
    for(i=0;i<n*n;i++)
        scanf("%lf",&A[i]);
    printf(" >> Please input the %d elements of matrix B(%d*1) one by one:\n",n,n);
    for(i=0;i<n;i++)
        scanf("%lf",&B[i]);
    if (Guass(A,B,n)!=0) /*调用Guass 消去, 1 为计算成功*/
        printf(" >> The solution of Ax=B is x(%d*1):\n",n);

    for (i=0;i<n;i++) /*打印结果*/
        printf("x(%d)=%f ",i,B[i]);
    puts("\n Press any key to quit...");
    getch();
}

```



## 归纳注释

在主函数中输入系数矩阵 A 和 B 的值，然后调用 Guass() 函数来进行计算。其中第一个参数是系数矩阵 A，第二个参数是向量 C，第 3 个参数是未知数的个数。最后 Guass() 函数把解向量赋值给 B。当矩阵有惟一解的时候，函数返回值为 1，否则返回 0。

# 实例 101 $n$ 阶方阵求逆



## 实例说明

对  $n$  阶方阵 A 求逆，程序运行效果如图 101-1 所示。

```
gn TC
*****
* This program is to inverse a square matrix A<nxn>. *
*****
>> Please input the order n of the matrix <n>0: 4
>> Please input the elements of the matrix one by one:
>> 1 7 3 5 6 23 4 5 7 8 9 1 3 4 5 7
Matrix A:
1.0000000    7.0000000    3.0000000    5.0000000
6.0000000    23.0000000    4.0000000    5.0000000
7.0000000    8.0000000    9.0000000    1.0000000
3.0000000    4.0000000    5.0000000    7.0000000

A's Inverse Matrix A-:
-0.6586207    0.1551724    -0.0534483    0.3672414
0.1172414    0.0229885    0.0068966    -0.1011494
0.4172414    -0.1436782    0.1568966    -0.2178161
-0.0827586    0.0229885    -0.0931034    0.1988506

Product of A and A-:
1.0000000    -0.0000000    -0.0000000    -0.0000000
-0.0000000    1.0000000    -0.0000000    0.0000000
-0.0000000    0.0000000    1.0000000    0.0000000
-0.0000000    0.0000000    -0.0000000    1.0000000

Press any key to quit...
```

图 101-1 实例 101 程序运行效果



## 实例解析

首先，对于  $k$  从 0 到  $n-1$  做如下几步。

从第  $k$  行、第  $k$  列开始的右下角子阵中，选取绝对值最大的元素，并记住此元素所在的行号和列号，再通过行交换和列交换将它交换到主元素的位置上，这一步称为全选主元。

$$\begin{cases} 1/a_{kk} \rightarrow a_{kk} \\ a_{kj}a_{kk} \rightarrow a_{kj} & j = 0, 1, \dots, n-1, \quad j \neq k \\ a_{ij} - a_{ik}a_{kj} \rightarrow a_{ij} & i, j = 0, 1, \dots, n-1, \quad i, j \neq k \\ -a_{ik}a_{kk} \rightarrow a_{ik} & i = 0, 1, \dots, n-1, \quad i \neq k \end{cases}$$

最后，根据在全选主元过程中所记录的行、列交换的信息进行恢复，恢复的原则如下。

在全选主元过程中，先交换的行、列后恢复；原来的行（列）交换用列（行）交换来恢

复。

程序中最重要的函数是 `brinv()`，该函数返回一个整型标志值。如果返回的标志值为 0，则表示矩阵 A 是奇异矩阵，并输出信息“这是奇异矩阵，无法求逆矩阵”。如果返回的标志值不是 0，则表示正常返回。其参数含义如下：

a 为双精度实型二维数组，体积为  $n \times n$ ，存放原矩阵 A；返回时存放其逆矩阵  $A^{-1}$ 。

N 为整型变量，是矩阵的阶数。



## 程序代码

### 【程序 101】 方阵求逆

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#define MAX 255

void MatrixMul(a,b,m,n,k,c) /*实矩阵相乘*/
int m,n,k; /*m 矩阵 A 的行数, n 矩阵 B 的行数, k 矩阵 B 的列数*/
double a[],b[],c[]; /*a 为 A 矩阵, b 为 B 矩阵, c 为结果, 即 c = AB */
{
    int i,j,l,u;
    /*逐行逐列计算乘积*/
    for (i=0; i<=m-1; i++)
        for (j=0; j<=k-1; j++)
        {
            u=i*k+j; c[u]=0.0;
            for (l=0; l<=n-1; l++)
                c[u]=c[u]+a[i*n+l]*b[l*k+j];
        }
    return;
}
int brinv(a,n) /*求矩阵的逆矩阵*/
int n; /*矩阵的阶数*/
double a[]; /*矩阵 A*/
{
    int *is,*js,i,j,k,l,u,v;
    double d,p;
    is=malloc(n*sizeof(int));
    js=malloc(n*sizeof(int));
    for (k=0; k<=n-1; k++)
    {
        d=0.0;
        for (i=k; i<=n-1; i++)
            /*全选主元, 即选取绝对值最大的元素*/
            for (j=k; j<=n-1; j++)
            {
                l=i*n+j; p=fabs(a[l]);
                if (p>d) { d=p; is[k]=i; js[k]=j; }
            }
        /*全部为 0, 此时为奇异矩阵*/
        if (d+1.0==1.0)
    }
```

```

        free(is); free(js); printf(" >> This is a singular matrix, can't be
inversed!\n");
        return(0);
    }
/*行交换*/
if (is[k]!=k)
    for (j=0; j<=n-1; j++)
    {
        u=k*n+j; v=is[k]*n+j;
        p=a[u]; a[u]=a[v]; a[v]=p;
    }
/*列交换*/
if (js[k]!=k)
    for (i=0; i<=n-1; i++)
    {
        u=i*n+k; v=i*n+js[k];
        p=a[u]; a[u]=a[v]; a[v]=p;
    }
l=k*n+k;
a[l]=1.0/a[l]; /*求主元的倒数*/
/* a[kj]a[kk] -> a[kj] */
for (j=0; j<=n-1; j++)
    if (j!=k)
    {
        u=k*n+j; a[u]=a[u]*a[l];
    }
/* a[ij] - a[ik]a[kj] -> a[ij] */
for (i=0; i<=n-1; i++)
    if (i!=k)
        for (j=0; j<=n-1; j++)
            if (j!=k)
            {
                u=i*n+j;
                a[u]=a[u]-a[i*n+k]*a[k*n+j];
            }
/* -a[ik]a[kk] -> a[ik] */
for (i=0; i<=n-1; i++)
    if (i!=k)
    {
        u=i*n+k; a[u]=-a[u]*a[l];
    }
}
for (k=n-1; k>=0; k--)
{
    /*恢复列*/
    if (js[k]!=k)
        for (j=0; j<=n-1; j++)
        {
            u=k*n+j; v=js[k]*n+j;
            p=a[u]; a[u]=a[v]; a[v]=p;
        }
    /*恢复行*/
    if (is[k]!=k)
        for (i=0; i<=n-1; i++)

```

```

    {
        u=i*n+k; v=i*n+is[k];
        p=a[u]; a[u]=a[v]; a[v]=p;
    }
}
free(is); free(js);
return(1);
}
print_matrix(a,n)/*打印的方阵 a 的元素*/
int n; /*矩阵的阶数*/
double a[]; /*矩阵 a*/
{
    int i,j;
    for (i=0; i<n; i++)
    {
        for (j=0; j<n; j++)
            printf("%13.7f\t",a[i*n+j]);
        printf("\n");
    }
}
main()
{
    int i,j,n=0;
    double A[MAX],B[MAX],C[MAX];
    static double a[4][4]={ {0.2368,0.2471,0.2568,1.2671},
                           {1.1161,0.1254,0.1397,0.1490},
                           {0.1582,1.1675,0.1768,0.1871},
                           {0.1968,0.2071,1.2168,0.2271} };
    static double b[4][4],c[4][4];
    clrscr();
    puts("*****");
    puts("* This program is to inverse a square matrix A(nxn). *");
    puts("*****");
    while(n<=0)
    {
        printf(" >> Please input the order n of the matrix (n>0): ");
        scanf("%d",&n);
    }

    printf(" >> Please input the elements of the matrix one by one:\n >> ");
    for(i=0;i<n*n;i++)
    {
        scanf("%lf",&A[i]);
        B[i]=A[i];
    }
    for(i=0;i<4;i++)
        for(j=0;j<4;j++)
            b[i][j]=a[i][j];

    i=brinv(A,n);

    if (i!=0)
    {

```

```

printf("    Matrix A:\n");
print_matrix(B,n);
printf("\n");
printf("    A's Inverse Matrix A-:\n");
print_matrix(A,n);
printf("\n");
printf("    Product of A and A- :\n");
MatrixMul(B,A,n,n,n,C);
print_matrix(C,n);
}
printf("\n Press any key to quit...");
getch();
}

```



## 归纳注释

在本实例中，要求输入矩阵的阶次，并依次输入矩阵的每个元素，求出该矩阵的逆矩阵，并在最后计算  $AA^{-1}$ ，以检验结果的正确性。

# 实例 102 复矩阵乘法



## 实例说明

求  $m \times n$  阶虚矩阵  $A$  与  $n \times k$  阶虚矩阵  $B$  的乘积矩阵  $C=AB$ 。程序运行效果如图 102-1 所示。

```

CN IC
*****
* This is a complex-matrix-multiplication program. *
* It calculate the two matrixes C(m*k)=A(m*n)B(n*k). *
*****
>> Please input the number of rows in A, m= 3
>> Please input the number of cols in A, n= 2
>> Please input the number of cols in B, k= 4
>> Please input the 6 elements in Ar one by one:
>> 1 2 43 5 76 5
>> Please input the 6 elements in Ai one by one:
>> 3 9 8 4 5 3
>> Please input the 8 elements in Br one by one:
>> 3 45 8 4 3 5 65 8
>> Please input the 8 elements in Bi one by one:
>> 3 5 9 5 8 4 3 5
Real part of C(3*4)=A(3*2)*B(2*4):
 -72.0000000 4.0000000 84.0000000 -40.0000000
 88.0000000 1904.0000000 585.0000000 152.0000000
 204.0000000 3408.0000000 879.0000000 304.0000000
Complex part of C(3*4)=A(3*2)*B(2*4):
 55.0000000 193.0000000 624.0000000 99.0000000
 205.0000000 615.0000000 726.0000000 304.0000000
 292.0000000 640.0000000 934.0000000 449.0000000
Press any key to quit...

```

图 102-1 实例 102 程序运行效果



## 实例解析

复矩阵相乘遵循如下法则。

- (1)  $A$  的列数与  $B$  的行数必须相同。
- (2)  $A_{m \times n}$  与  $B_{n \times k}$  的乘积为  $m \times k$  的矩阵。
- (3) 结果矩阵  $C$  中每个元素  $C_{ij}$  的值是  $A$  中  $i$  行的元素与  $B$  中  $j$  列的元素的乘积和。用公式表示就是：

$$c_{ij} = \sum_{t=0}^{n-1} a_{it} b_{tj} \quad i = 0, 1, \dots, m-1, \quad j = 0, 1, \dots, k-1$$

复数相乘可采用如下的算法：

设

$$e + fi = (a + bi) \times (c + di)$$

令

$$p = ac, q = bd, s = (a + b) \times (c + d)$$

则

$$e = p - q, f = s - p - q$$

程序中最主要的函数是 CmatrixMul(), 其中：

ar 为双精度实型二维数组，体积为  $m \times n$ ，存放矩阵  $A$  的实部元素；  
 ai 为双精度实型二维数组，体积为  $m \times n$ ，存放矩阵  $A$  的虚部元素；  
 br 为双精度实型二维数组，体积为  $n \times k$ ，存放矩阵  $B$  的实部元素；  
 bi 为双精度实型二维数组，体积为  $n \times k$ ，存放矩阵  $B$  的虚部元素；  
 m 为整型变量，矩阵  $A$  的行数，也是乘积矩阵  $C=AB$  的行数；  
 n 为整型变量，矩阵  $A$  的列数，也是  $B$  矩阵的行数；  
 k 为整型变量，矩阵  $B$  的列数，也是乘积矩阵  $C=AB$  的列数；  
 cr 为双精度实型二维数组，体积为  $m \times k$ ，存放乘积矩阵  $C=AB$  的实部元素；  
 ci 为双精度实型二维数组，体积为  $m \times k$ ，存放乘积矩阵  $C=AB$  的虚部元素。



## 程序代码

### 【程序 102】 复矩阵乘法

//本实例源码参见光盘



## 归纳注释

程序运行时，由用户输入  $A$ 、 $B$  矩阵的行、列数以及  $m$ 、 $n$ 、 $k$ ，并依次输入两个矩阵的实元素和虚元素，实现复矩阵相乘，并输出结果。

## 实例 103 求定积分



## 实例说明

用变步长辛普森法求定积分。程序运行效果如图 103-1 所示。

```
C:\> TC
*****
* This program is to calculate the Value of
* a definite integral by Simpson Method.
*****
>> The result of definite integral is :
>> SIGMA<0,1>ln<1+x>/<1+x^2>dx = 2.72198e-01
-----
Press any key to quit...
```

图 103-1 实例 103 程序运行效果



## 实例解析

用变步长辛普森 (Simpson) 法则求定积分  $S = \int_a^b f(x)dx$  值的实现方法如下。

用梯形公式计算  $T_n = h[f(a)+f(b)]/2$ , 其中  $n=1$ ,  $h=b-a$ , 且令  $S_n = T_n$

用变步长梯形法则计算  $T_{2n} = \frac{1}{2}T_n + \frac{h}{2} \sum_{k=0}^{n-1} f(x_k + \frac{h}{2})$

用辛普森求积公式计算  $S_{2n} = (4T_{2n} - T_n)/3$

若  $|S_{2n} - S_n| \geq \varepsilon$ , 则令  $2n \rightarrow n$ ,  $h/2 \rightarrow h$ , 转到步骤 (2) 继续进行计算; 否则结束,  $S_{2n}$  即为所求积分的近似值。其中  $\varepsilon$  为事先给定的求积分精度。

函数 `fsimpf()` 表示调用被积函数的值, 参数为需要计算的  $x$  值。`fsimp()` 是使用辛普森法则进行计算的函数, 参数意义如下。

$a$  为双精度实型变量, 表示积分下限;

$b$  为双精度实型变量, 表示积分上限, 要求  $b > a$ ;

$eps$  为双精度实型变量, 表示积分的精度要求。

在本例中, 使用辛普森法则计算定积分:  $S = \int_0^1 \frac{\ln(1+x)}{1+x^2} dx$

精度取  $\varepsilon = 0.000001$



## 程序代码

### 【程序 103】 求定积分

// 本实例源码参见光盘



## 归纳注释

主程序先输出提示信息, 接着调用 `fsimp` 函数计算定积分, 然后输出计算结果, 并退出程序。

# 实例 104 求满足特异条件的数列



## 实例说明

输入  $m$  和  $n$  ( $20 \geq m \geq n > 0$ ), 求出满足以下方程的正整数数列  $i_1, i_2, \dots, i_n$ , 使得  $i_1+i_2+\dots$

$+in=m$ , 且  $i_1 \geq i_2 \geq \dots \geq i_n$ 。例如: 当  $n=4$ ,  $m=8$  时, 将得到如下 5 个数列:

5 1 1 1      4 2 1 1      3 3 1 1      3 2 2 1      2 2 2 2

程序运行效果如图 104-1 所示。

```
*****  
* This is a program to get number sequence with *  
* special characteristic. Input n and m, get the sequence*  
* S1, S2, ..., Sn, where S1+S2+...+Sn=m and S1>=S2>=...>=Sn*  
* e.g. n=4, m=8, S: 5 1 1 1, 4 2 1 1, 3 3 1 1, 3 2 2 1, 2 2 2 2*  
*****  
>> Input element number of the sequence <n<=10>: 5  
>> Input the sum of the elements <m<=20>: 12  
>> Possible sequences are as follows:  
No. 1: 5 1 1 1  
No. 2: 4 2 1 1  
No. 3: 3 3 1 1  
No. 4: 3 2 2 1  
No. 5: 2 2 2 1  
No. 6: 2 2 1 1  
No. 7: 2 1 1 1  
No. 8: 1 1 1 1  
No. 9: 5 1 1 1  
No. 10: 4 2 2 1  
No. 11: 3 3 2 1  
No. 12: 3 2 2 2  
No. 13: 2 2 2 2  
  
Press any key to quit...
```

图 104-1 实例 104 程序运行效果

## 实例解析

可将原题抽象为将  $m$  分解为  $n$  个整数, 且  $n$  个整数的和为  $m$ ,  $i_1 \geq i_2 \geq \dots \geq i_n$ 。分解整数的方法很多, 由于题目中有 “ $i_1 \geq i_2 \geq \dots \geq i_n$ ” 提示我们可先确定最右边  $i_n$  元素的值为 1, 然后按照条件使前一个元素的值一定大于等于当前元素的值, 不断地向前推就可以解决问题。下面的程序允许用户选定  $m$  和  $n$ , 并输出满足条件的所有数列。

## 程序代码

### 【程序 104】 求满足特异条件的数列

//本实例源码参见光盘

## 归纳注释

本实例根据所要求的数列长度  $n$  以及和  $m$ , 对  $m$  进行分解, 试探出所有的数列, 然后输出。

## 实例 105 超长正整数的加法

## 实例说明

实现超长正整数的加法运算。程序运行效果如图 105-1 所示。

```
*****  
* This program is to calculate *  
* the addition of king sized positive integer. *  
*****  
>> Input S1= 9999999999999923084702933334999999999999  
>> Input S2= 23974923874092387401932743892047032948393  
>> The addition result is as follows.  
  
S1= 9999999999999923084702933334999999999999  
S2= 23974923874092387401932743892047032948393  
S1+S2=123974923874092310486635677227047032948392  
  
Press any key to quit...
```

图 105-1 实例 105 程序运行效果

## 实例解析

首先要设计一种数据结构来表示一个超长的正整数，然后才能够设计算法。

首先采用一个带有表头结点的环形链来表示一个非负的超大整数，如果从低位开始为每个数字编号，则第 1~4 位、第 5~8 位……的每 4 位组成的数字，依次放在链表的第 1 个、第 2 个……第 n 结点中，不足 4 位的最高位存放在链表的最后一个结点中，表头结点的值规定为 -1。例如：大整数“587890987654321”可用如下的带表头结点 head 的链表表示。

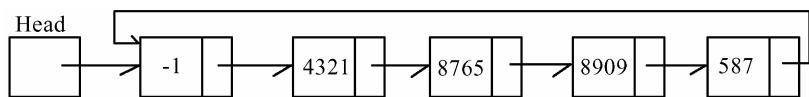


图 105-2 超长正整数的链表表示

按照此数据结构，可以从两个表头结点开始，顺序依次对应相加，求出所需要的进位后，代入下一个结点的运算。

## 程序代码

### 【程序 105】 超长正整数的加法

//本实例源码参见光盘

## 归纳注释

本实例采用 `inputint` 函数来输入超长正整数，用 `addint` 函数实现两个超长正整数的加法，用 `insert_after` 来添加一个链表的结点，用 `printint` 来实现超长正整数的输出。

# 04

## 第四部分 图形篇

### 精彩导读

- 金刚石图案
- 抛物样条曲线
- Mandelbrot 分形图案
- VGA256 色模式编程
- 运动的小车
- 图形时钟

# 实例 106 绘制直线

## 实例说明

在图形模式下绘制直线。程序运行效果如图 106-1 所示。



图 106-1 实例 106 程序运行效果

## 实例解析

### TC 2.0 与图形接口

TC 2.0 具有相当强的图形处理能力，支持 CGA、MCGA、EGA、VGA、IBM8514 和 Hercules 等图形显示器。一般的 IBM PC 型显示器可以在两种基本视频模式下工作，一是图形方式，另一种是文本方式。文本方式即常见的命令行方式，屏幕上可以显示的最小单位是字符。常用的 VGA 显示适配器，可显示 80 列 50 行文本。图形方式下，屏幕上每一个可以控制的单元叫做像素 (pixel)，它是组成图形的基本元素，一般称为点。通常把屏幕上所包含的像素的个数叫做分辨率。分辨率越高，显示的图形越细致，质量越好。VGA 显示器的分辨率为  $640 \times 480$ ，即 VGA 在水平方向上有 640 个像素，垂直方向上有 480 个像素。

在图形方式下，屏幕上每个像素的显示位置用点坐标系来描述。在这种坐标系中，屏幕左上角为坐标原点  $(0, 0)$ ，水平方向为  $x$  轴，自左向右递增，垂直方向为  $y$  轴，自上向下递增。如图 106-2 所示。分辨率不同，水平方向和垂直方向上的点数也不一样，即  $\max_x$ 、 $\max_y$  数值不同。

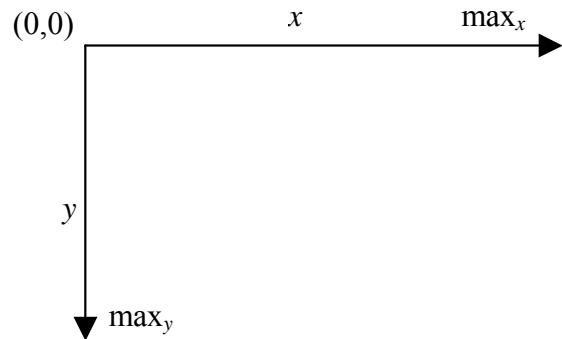


图 106-2 像素坐标示意图

在 TC 中，坐标数据可以用两种形式给出，一是绝对坐标，另一是相对坐标。绝对坐标的参考点是坐标系的原点 (0, 0)， $x$  和  $y$  只能取规定范围内的正整数，其坐标值在整个屏幕范围内确定。相对坐标是相对于“当前点”的坐标，所以其坐标的参考点是当前点。在相对坐标中， $x$  和  $y$  的取值是相对于当前点在  $x$  方向和  $y$  方向上的增量，这个增量可以是正的，也可以是负的。

## TC 图形库函数

Turbo C 2.0 具有 70 多个图形库函数，因此其图形功能极为丰富。所有这些库函数均在头文件“graphics.h”中定义，所以，凡是在程序中要调用这些图形函数，都必须在程序文件的开头写上文件包含命令“#include <graphics.h>”。

TC 2.0 的图形库函数主要有 6 大类：图形系统管理、屏幕管理、绘图函数、图形属性控制、填充和图形方式下的文本操作等。

### (1) 图形系统管理

在一般缺省情况下，屏幕为 80 列 50 行的文本方式，此时，所有的图形函数均不能操作，因此在使用图形函数绘图之前，必须将屏幕显示适配器设置为一种图形模式，即所谓的“图形方式初始化”。在绘图完毕后，要回到文本方式，必须关闭图形方式。TC 2.0 提供了 14 个函数对图形系统进行控制和管理工作。

① 图形方式初始化通过函数 initgraph 来完成。其调用格式为：

```
initgraph(*gdriver, *gmode, *path);
```

函数 initgraph 是通过从磁盘上装入一个图形驱动程序来初始化图形系统，并将系统设置为图形方式。其中 3 个参数的含义如下。

$gdriver$  是一个整型值，用来指定要装入的图形驱动程序，该值在头文件 graphics.h 中定义，常用的是 DETECT、EGA、VGA 和 IBM8514。使用 DETECT，由系统自动检测图形适配器的最高分辨率模式，并装入相应的图形驱动程序。

$gmode$  是一个整型值，用来设置图形显示模式，不同的图形驱动程序有不同的图形显示模式，即使是同一个图形驱动程序下，也有几种图形显示模式。图形显示模式决定了显示的分辨率、可同时显示的颜色的多少、调色板的设置方式以及存储图形的页数。常用的几种显示模式如表 106-1 所示。

$path$  是一个字符串，用来指明图形驱动程序所在的路径。如果驱动程序就在用户当前目录下，则该参数可以为空字符串，否则应给出具体的路径。一般情况下，Turbo C 安装在 C 盘的 TC 目录中，则该路径为 C:\TC，如果写在参数中则为“C:\TC”。

表 106-1 几种常用的显示模式

图形驱动程序 (gdriver)	图形显示模式 (gmode)	值	分辨率	颜色数	页
EGA	EGALO	0	640×200	16	1
	EGAHI	1	640×350	16	2
VGA	VGALO	0	640×200	16	2
	VGAMED	1	640×350	16	2
	VGAHI	2	640×480	16	1
IBM8514	IBM8514LO	0	640×480	256	
	IBM8514HI	1	1024×768	256	

例如，在程序中使用 VGA 图形驱动程序，图形显示模式为 VGAHI，即 VGA 高分辨率图形模式，分辨率为 640×480，则 initgraph 函数的调用方式如下：

```
int gdriver=VGA, gmode=VGAHI;  
initgraph(&gdriver, &gmode, "C:\TC");
```

也可以用整型常数代替符号常数，例如：

```
int gdriver=9,gmode=2;  
initgraph(&gdriver,&gmode,"C:\\TC");
```

这两种方式是等效的。

另外，可使用 DETECT 模式，由系统自动检测，并把图形显示模式设置为检测到的驱动程序的最高分辨率。例如：

```
int gdriver=DETECT,gmode;  
initgraph(&gdriver,&gmode,"C:\\TC");
```

② 关闭图形方式。在运行图形程序绘图结束后，要回到文本方式，以进行其他工作，这时应关闭图形方式。关闭图形方式要调用函数 closegraph。其调用格式为：

```
closegraph();
```

其作用是释放所有图形系统分配的存储区，恢复到调用 initgraph 之前的状态。

## (2) 屏幕管理

TC 2.0 提供了 11 个函数用于对屏幕和视图区进行控制管理。常用的有以下 3 种。

### ① 设置视图区

```
setviewport(x1,y1,x2,y2,c);
```

该函数在屏幕上定义一个以(x1,y1)为左上角坐标，(x2,y2)为右下角坐标的视图区。c 为裁剪状态参数，当 c=1 时，超出视图区的图形部分被自动裁剪掉，当 c=0 时，则对超出视图区的图形不做裁剪处理。

视图区建立后，所有的图形输出坐标都是相对于当前视图区的，即视图区的左上角点为坐标原点(0,0)，而与图形在屏幕上的位置无关。

### ② 清除视图区

清除视图区函数为 clearviewport，它的作用是清除当前的视图区，将当前点位置设置于屏幕的左上角 (0, 0) 点。执行后，原先的视图区不再存在。调用格式为：

```
clearviewport();
```

### ③ 清屏

清屏使用函数 cleardevice，它的作用是清除全屏幕，并将当前点位置设置为原点(0,0)。但是其他的图形系统设置保持不变，如线型、填充模式、文本格式和模式等，如果设置了视图区，则视图区的设置保持不变，包括当前点位置设置在视图区的左上角。调用格式为：

```
cleardevice();
```

## (3) 绘图函数

绘图函数是编写绘图程序的基础，也是任何一种图形软件的核心内容。Turbo C 的 BGI (Borland Graphics Interface) 提供了大量的基本绘图函数，以方便图形设计。这些绘图函数可分为直线类、圆弧类、多边形类等。

本实例主要说明直线类绘图函数的用法。

## 直线类绘图函数

直线类绘图函数用于绘制直线，可以用两种坐标，绝对坐标和相对坐标。

### (1) line 函数

line 函数用于在指定两点之间画一条直线段：

```
line(int x1, int y1, int x2, int y2);
```

参数 x1、y1、x2、y2 使用绝对坐标，(x1,y1) 和 (x2,y2) 分别为直线的两个端点坐标。用 line 函数画直线时，当前点的位置不变。

### (2) lineto 函数

`lineto` 函数用于从当前点位置到指定位置  $(x,y)$  画一条直线，并改变当前点的位置。所以在画一条直线的同时，当前点的位置也移到了指定点，即直线的终点。调用格式为：

```
lineto(int x, int y);
```

#### (3) `moveto` 函数

函数 `moveto(int x, int y)` 用于将当前点移动到  $(x,y)$ 。

#### (4) `linerel` 函数

`linerel` 函数使用相对坐标画直线。其功能是从当前点位置开始画线到指定点位置，指定点位置的坐标不是以绝对坐标形式给出的，而是以其相对于当前点（即直线的起点）位置的坐标增量给出的。调用格式：

```
linerel(int dx, int dy);
```

假设当前点位置坐标  $(x,y)$ ，则 “`linerel(dx,dy);`” 等效于 “`lineto(x+dx, y+dy);`”。

#### (5) `moverel` 函数

该函数的功能与 `moveto` 函数相似，但它使用的是相对坐标。`moverel(int dx, int dy)` 用于将当前点位置在  $x$  和  $y$  方向上分别移动增量  $dx$  和  $dy$ 。



## 程序代码

### 【程序 106】 绘制直线

```
#include <graphics.h>
void main()
{
    int gdriver=DETECT,gmode;
    initgraph(&gdriver,&gmode,"c:\\\\tc");
    cleardevice();
    printf("\n
        Draw lines with function 'line'..");
    line(160,120,480,120);
    line(480,120,480,360);
    line(480,360,160,360);
    line(160,360,160,120);
    getch();
    cleardevice();
    getch();
    printf("\n    Draw lines with function 'lineto'..");
    moveto(160,120);
    lineto(480,120);
    lineto(480,360);
    lineto(160,360);
    lineto(160,120);
    getch();
    cleardevice();
    getch();
    printf("\n    Draw lines with function 'linerel'..");
    moveto(160,120);
    linerel(320,0);
    linerel(0,240);
    linerel(-320,0);
    linerel(0,-240);
    getch();
    closegraph();
```

```
}
```



## 归纳注释

要使用 Turbo C 2.0\3.0 中的画图模式，首先应该在 Option 菜单中的 Linker→Libraries 选项里选中 Graphic Library。然后每次使用时应该在程序中进行初始化，即在代码中加入两条语句“int gdriver=DETECT,gmode; initgraph(&gdriver,&gmode, “xxxxx”);”，其中 xxxxx 是用户的 Turbo C 的图形库目录所在路径。退出程序前，应该调用 closegraph() 函数退出图形模式，回到文本模式。

本实例通过绘制矩形来说明直线类绘图函数的用法。

# 实例 107 绘制圆



## 实例说明

本实例实现圆形的绘制。程序运行效果如图 107-1 所示。

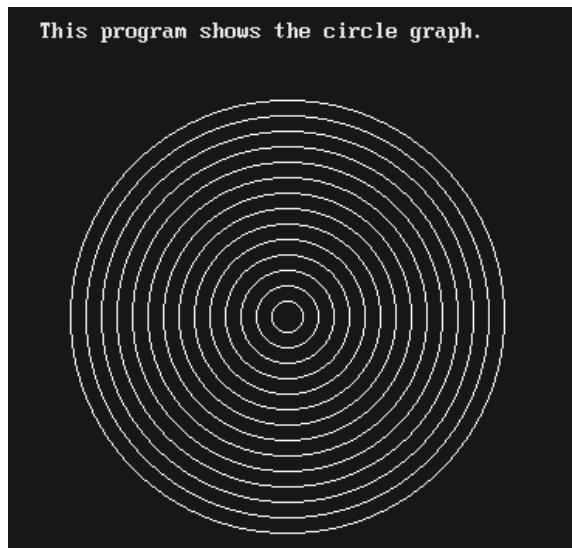


图 107-1 实例 107 程序运行效果



## 实例解析

圆的绘图函数是 circle。该函数用于以指定圆心和半径画圆。其调用格式为：

```
circle(int x, int y, int r);
```

其中 (x,y) 为指定圆心的坐标，r 为圆的半径。例如：

```
circle(320,240,100);
```

执行结果是以点 (320,240) 为圆心，以 100 为半径画一个圆。



## 程序代码

### 【程序 107】 绘制圆

```
/* draw circles */
#include <graphics.h>
```

```

void main()
{
    int b;
    int gdriver=DETECT,gmode;
    initgraph(&gdriver,&gmode,"c:\\tc");
    cleardevice();
    printf("\n\n\n      This program shows the circle graph.\n");
    for(b=10;b<=140;b+=10)
        circle(320,240,b);
    getch();
    closegraph();
}

```



## 归纳注释

程序实现了画一组同心圆。首先初始化图形模式，再清除屏幕，然后打印提示信息，通过 for 循环语句画一组圆心相同，半径依次变大的同心圆，最后，按任意键退出图形模式，程序结束。

# 实例 108 绘制圆弧



## 实例说明

本实例实现圆弧的绘制。程序运行效果如图 108-1 所示。

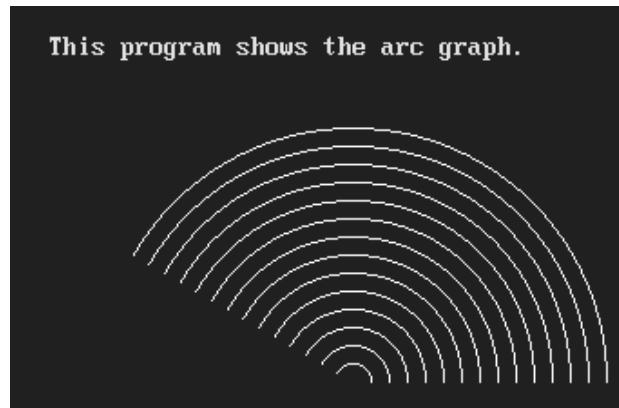


图 108-1 实例 108 程序运行效果



## 实例解析

画圆弧的函数是 arc，其调用格式是：

```
arc(int x, int y, int angs, int ange, int r);
```

其中，(x, y) 为圆弧所在圆心的坐标，angs, ange 分别为圆弧的起始角和终止角，单位为“度”，r 为圆弧的半径。

例如，调用 “arc(320,240,90,180,100);” 的结果是以点(320,240)为圆心，100 为半径，从 90° ~180° 画了四分之一圆的圆弧。

当圆弧的起始角 `angs=0`, 终止角 `ange=360` 时, 则可以画出一个整圆。



## 程序代码

### 【程序 108】 绘制圆弧

```
/* draw arc */
#include <graphics.h>
void main()
{
    int b;
    int gdriver=DETECT,gmode;
    initgraph(&gdriver,&gmode, "c:\\tc");
    cleardevice();
    printf("\n\n\n  This program shows the arc graph.\n");
    for(b=10;b<=140;b+=10)
        arc(320,240,0,150,b);
    getch();
    closegraph();
}
```



## 归纳注释

程序首先初始化图形模式, 再打印出提示信息, 然后通过 `for` 循环画一组半径逐渐增大的从  $0^\circ \sim 150^\circ$  的同心圆弧, 最后按任意键退出图形模式, 结束程序。

# 实例 109 绘制椭圆



## 实例说明

本实例实现椭圆的绘制。程序运行效果如图 109-1 所示。

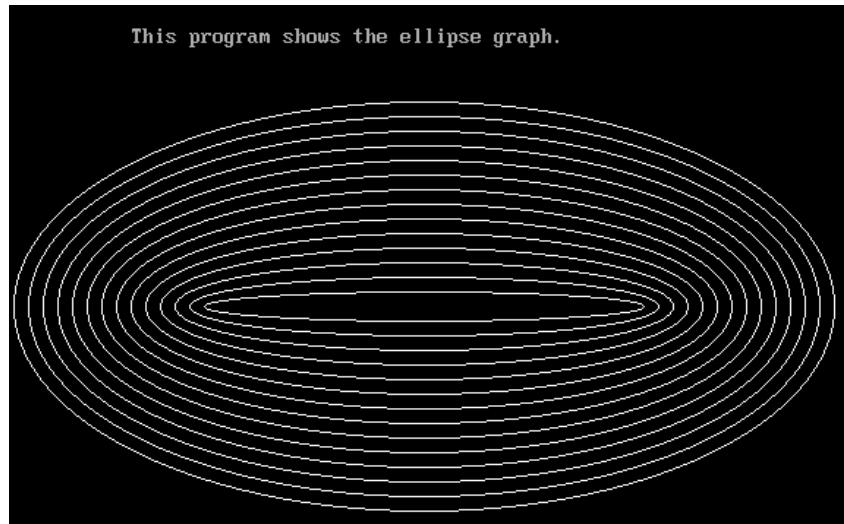


图 109-1 实例 109 程序运行效果



## 实例解析

画椭圆的函数是 ellipse。该函数用于画椭圆弧或椭圆。其调用格式为：

```
ellipse(int x, int y, int angs, int ange, int xr, int yr);
```

其中，(x,y) 为椭圆的中心坐标，angs, ange 为椭圆弧的起始角和终止角，单位为“度”，xr, yr 分别为椭圆的水平半轴和垂直半轴。

如果 angs=0, ange=360，则可以画出一个完整的椭圆。

xr>yr，则画出长轴为水平方向的椭圆或椭圆弧；

xr<yr，则画出长轴为垂直方向的椭圆或椭圆弧；

xr=yr，则可以画圆或圆弧。



## 程序代码

### 【程序 109】 绘制椭圆

```
/* draw ELLIPSE */
#include <graphics.h>
void main()
{
    int a=150,b;
    int gdriver=DETECT,gmode;
    initgraph(&gdriver,&gmode,"c:\\\\tc");
    cleardevice();
    for(b=10;b<=140;b+=10,a+=10)
        ellipse(320,240,0,360,a,b);
    getch();
    closegraph();
}
```



## 归纳注释

本实例程序先初始化图形模式，再打印出提示信息，然后通过 for 循环画一组半轴逐渐增大的从 0° ~360° 的同中心椭圆，最后按任意键退出图形模式，结束程序。

多边形绘图函数主要有 rectangle(int x1, int y1, int x2, int y2)，用于绘制以 (x1,y1) 为左上角点，以 (x2,y2) 为右下角点的矩形； drawpoly(int nps, int \*pxy)，用于绘制具有 nps 个顶点的多边折线，数组 pxy 用于存放这些顶点的坐标，比如，有一个名为 xy 的整型数组中存放了 4 个顶点的坐标为 [x1,y1, x2, y2, x3, y3, x4, y4]，则调用格式为 drawpoly(4,xy)，如果最后一个点的坐标与第一个点的坐标相同，则可以画一个封闭的多边形。

## 实例 110 设置背景色和前景色



## 实例说明

本实例实现对背景色和前景色的设置。程序运行效果如图 110-1 所示。

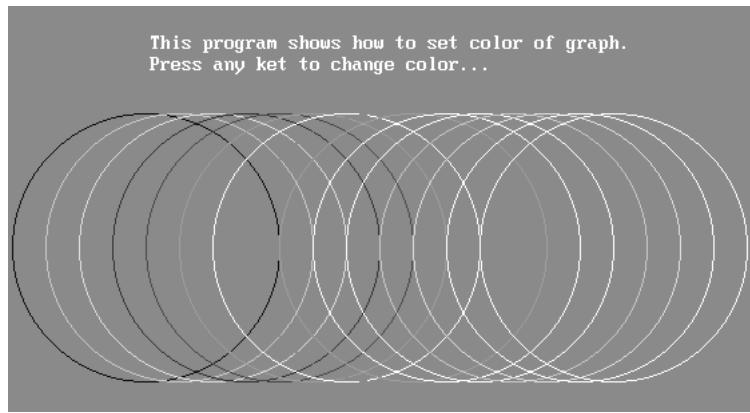


图 110-1 实例 110 程序运行效果



## 实例解析

图形的属性控制包括颜色和线型。颜色包括背景色和前景色。背景色指的是屏幕的颜色，即绘图时的底色。前景色指的是绘图时图形线条所用的颜色。任何绘图函数都是在当前的颜色（包括背景色和前景色）和线型状态下进行绘图的。在前面的例子中没有提到颜色和线型，是因为用了系统的缺省值。系统的缺省值是：背景色为黑色，前景色为白色，线型为实型。

如果要用到系统缺省值以外的颜色和线型，则可以利用图形颜色和线型控制函数来设置。

### (1) 背景色设置

设置背景色函数为 `setbkcolor`，其功能是设置绘图时的背景颜色。调用格式为：

```
setbkcolor(int color);
```

参数 `color` 代表所取的颜色，可以为整型常数，也可以用符号常数表示。TC 2.0 定义的颜色如表 110-1 所示。

表 110-1

颜色表

符 号 名	数 值	颜 色	符 号 名	数 值	颜 色
BLACK	0	黑 色	DARKGRAY	8	深灰色
BLUE	1	蓝 色	LIGHTBLUE	9	浅蓝色
GREEN	2	绿 色	LIGHTGREEN	10	浅绿色
CYAN	3	青 色	LIGHTCYAN	11	浅青色
RED	4	红 色	LIGHTRED	12	浅红色
MAGENTA	5	紫红色	LIGHTMAGENTA	13	淡紫色
BROWN	6	棕 色	YELLOW	14	黄 色
LIGHTGRAY	7	浅灰色	WHITE	15	白 色

例如，要把背景色设置成浅蓝色，可以调用：“`setbkcolor(LIGHTBLUE);`”或“`setbkcolor(9);`”。

### (2) 设置前景色

函数 `setcolor` 用于设置前景色，即绘图用的颜色。调用格式为“`setcolor(int color);`”其参数 `color` 的含义与 `setbkcolor` 的参数相同。



## 程序代码

### 【程序 110】 设置背景色和前景色

```
/* set color */
#include <graphics.h>
```

```

void main()
{
    int cb,cf;
    int gdriver=DETECT,gmode;
    initgraph(&gdriver,&gmode,"c:\\\\tc");
    cleardevice();
    printf("\n      This program shows how to set color of graph.");
    printf("\n      Press any key to change color...");
    for(cb=0;cb<=15;cb++)
    {
        setbkcolor(cb);
        for(cf=0;cf<=15;cf++)
        {
            setcolor(cf);
            circle(100+cf*25,240,100);
        }
        getch();
    }
    getch();
    closegraph();
}

```



### 归纳注释

程序首先初始化图形方式，清除屏幕，打印输出提示信息，然后在第一个 for 循环中依次设置 0~15 号颜色，在每种背景色下，依次设置 0~15 号前景色，并画一个该颜色的圆，按任意键，换一种背景色，如此循环，最后按任意键，关闭图形方式，结束程序。

## 实例 111 设置线条类型



本实例实现对线条类型的设置。程序运行结果如图 111-1 所示。

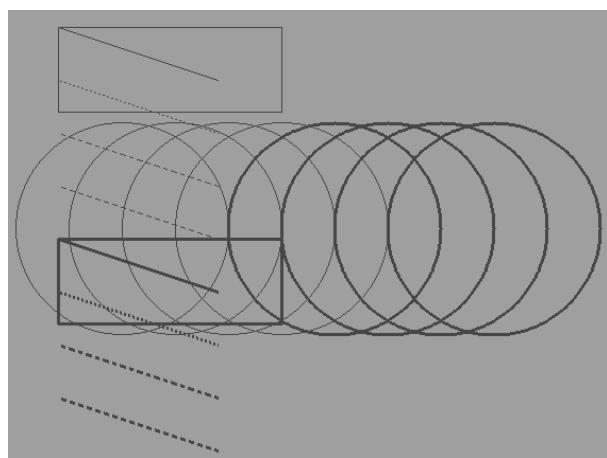


图 111-1 实例 111 程序运行结果



## 实例解析

设置线条类型采用函数 `setlinestyle` 来实现。该函数用于设置当前绘图所用的线型和宽度，这些设置仅限于对直线类图形有效。其调用格式为：

```
setlinestyle(int sty, int pat, int b);
```

该函数所用的 3 个参数含义如下。

(1) `sty` 用来定义所画直线的类型。包括 `SOLID_LINE`、`DOTTED_LINE`、`CENTER_LINE`、`DASHED_LINE` 和 `USERBIT_LINE`，其对应数值依次为 0~4，含义为实线（缺省）、点线、中心线、虚线和用户自定义类型。

(2) `pat` 用于用户自定义线型。如果是使用前 4 种系统预定义的线型，则该参数可取 0 值。

(3) `b` 指定所画直线的粗细，以像素为单位。可取 `NORM_WIDTH` 和 `THICK_WIDTH` 两种值（对于数值为 1 和 3），含义是 1 个像素宽（缺省）和 3 个像素宽。

当函数 `setlinestyle` 的第一个参数为 `USERBIT_LINE`（或 4）时，可以由用户自己定义直线类型。此时，第 3 个参数的意义不变。在第二个参数中定义直线的类型，该参数是一个 16 位二进制码，每一位（bit）表示一个像素。某一位（bit）置 1 时表示直线上相应位置有显示，置 0 时为空。例如，1111 1111 1111 1111，16 位全置 1，因此是画一条实线。而 1010 1010 1010 1010，隔位置 1，因此是画一条点线。

在实际编写程序时，一般把 16 位二进制数转换为 4 位十六进制数，每 4 位二进制数转换为 1 位十六进制数。故上面的两个例子转换为十六进制数为 `FFFF` 和 `AAAA`。函数的调用方法为“`setlinestyle(4,0xAAAA,1);`”，用这种方法，可以根据需要定义各种线型。



## 程序代码

### 【程序 111】 设置线条类型

```
/* set line style */
#include <graphics.h>
void main()
{
    int i,j,c,x=50,y=50,k=1;
    int gdriver=DETECT,gmode;

    initgraph(&gdriver,&gmode,"c:\\tc");
    cleardevice();
    setbkcolor(9);
    setcolor(4);
    for(j=1;j<=2;j++)
    {
        for(i=0;i<4;i++)
        {
            setlinestyle(i,0,k);
            line(50,50+i*50+(j-1)*200,200,200+i*50+(j-1)*200);
            rectangle(x,y,x+210,y+80);
            circle(100+i*50+(j-1)*200,240,100);
        }
        k=3;
        x=50;
    }
}
```

```
    y=250;
}
getch();
closegraph();
}
```



## 归纳注释

程序用于演示系统预定义的 4 种线型。程序首先初始化图形方式，清屏，设置背景色和前景色，在 for 循环中设置两种不同的线型宽度，在第 2 个 for 循环中设置 4 种不同的线型，并分别绘制直线、矩形和圆。可以看出，直线会产生变化，而矩形和圆只在不同像素的线型粗细时会有变化。

# 实例 112 设置填充类型和填充颜色



## 实例说明

本实例实现对填充类型以及填充颜色的设置。程序运行结果如图 112-1 所示。



图 112-1 实例 112 程序运行结果



## 实例解析

(1) `setfillstyle` 函数用来设置当前填充模式和填充颜色，以便用于填充一个指定的封闭区域，调用格式为：

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

其中，参数 `pattern` 用于指定填充模式。系统有 13 种预定义的填充模式，见表 112-1。参数 `color` 用于指定填充颜色。

表 112-1

系统预定义的填充模式

宏	值	含    义	宏	值	含    义
EMPTY_FILL	0	用背景颜色填充	HATCH_FILL	7	网格线填充
SOLID_FILL	1	实填充	XHATCH_FILL	8	斜网格线填充
LINE_FILL	2	用线“—”填充	INTERLEAVE_FILL	9	间隔点填充
LTSLASH_FILL	3	用斜线填充（阴影线）	WIDE_DOT_FILL	10	稀疏点填充
SLASH_FILL	4	用粗斜线填充（粗阴影线）	CLOSE_DOT_FILL	11	密集点填充
BKSLASH_FILL	5	用粗反斜线填充（粗阴影线）	USER_FILL	12	用户定义的模式
LTBKSLASH_FILL	6	用反斜线填充（阴影线）			

(2) `floodfill` 函数用于对一个指定区域进行填充操作，其填充模式和颜色由 `setfillstyle` 函数指定。其调用格式为：

```
floodfill(int x, int y, int bcolor);
```

参数 (x,y) 指位于填充区域内任一点的坐标，该点作为填充的起始点。bcolor 为填充区域的边界颜色。例如：

```
setcolor(RED);
circle(320,240,150);
setfillstyle(SOLID_FILL, GREEN);
floodfill(320,240,RED);
```

该段程序的作用是，用红颜色画一个圆，然后用绿色色块填充该圆。

(3) 其他填充函数。以下所列的几个填充函数，均需事先由函数 `setfillstyle` 指定当前的填充模式和颜色。

① `fillellipse`(int x, int y, int rx, int ry) 函数，用于画一个填充的实椭圆，用当前颜色画出边线。其参数 (x, y) 为椭圆中心坐标，rx, ry 分别为椭圆的水平和垂直半轴长。

② `sector`(int x, int y, int angs, int ange, int rx, int ry) 函数，用于画一个填充的椭圆扇区，用当前颜色画出边线。其参数意义与 `ellipse` 函数相同。

③ `fillpoly`(int nps, int \*pxy) 函数，用于画并填充一个多边形（必须使首末两点重合以确保多边形封闭），边线用当前颜色画出。其参数意义与 `drawpoly` 函数相同。

## 程序代码

### 【程序 112】 设置填充类型和填充颜色

```
//本实例源码参见光盘
```

## 归纳注释

程序首先初始化图形方式，清屏，设置背景色，在 `for` 循环中设置不同的前景色，画矩形，并设置不同的填充颜色，填充该矩形，形成一系列的填充矩形。最后按任意键，关闭图形方式，结束程序。

## 实例 113 图形文本的输出

## 实例说明

本实例实现图形模式下的文本输出。程序运行结果如图 113-1 所示。

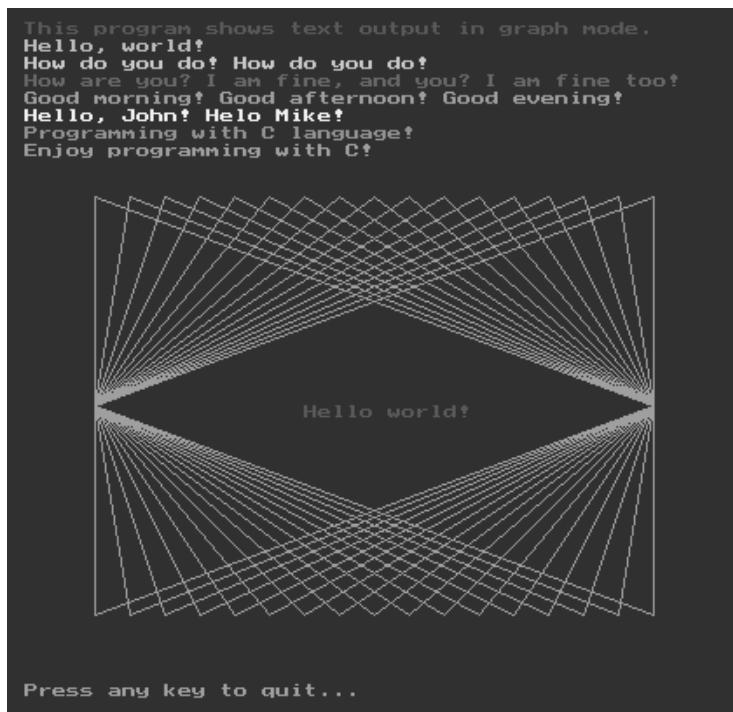


图 113-1 实例 113 程序运行结果



## 实例解析

### (1) outtext 函数

outtext 函数用于在当前位置输出一个文本字符串。其调用格式：

```
outtext(char *text);
```

参数 text 是一个字符串。例如：

```
outtext("Hello world");
```

将在当前位置输出一个字符串“Hello world”。

### (2) outtextxy 函数

outtextxy 函数用来在 (x, y) 位置输出一个字符串。其调用格式为：

```
outtextxy(int x, int y, char *text);
```

参数 (x, y) 为指定位置的坐标, text 为待输出的字符串。



## 程序代码

### 【程序 113】 图形文本的输出

```
//本实例源码参见光盘
```



## 归纳注释

本实例演示了图形方式下简单的文本输出。程序先初始化图形方式，清屏，设置前景色，移动当前点位置，输出提示信息，再设置前景色，移动当前点位置到下一行，再输出一串字符串，如此反复。接着在 for 循环中画一系列的直线，形成窗口图形，并在窗口中输出“Hello world!”。最后提示按任意键，关闭图形方式，结束程序。

# 实例 114 金刚石图案

## 实例说明

将半径为  $R$  的圆周等分成  $n$  份, 然后用直线将各等分点两两相连, 这样形成的图案称之为“金刚石”图案。本实例将实现金刚石图案的绘制。程序运行结果如图 114-1 所示。

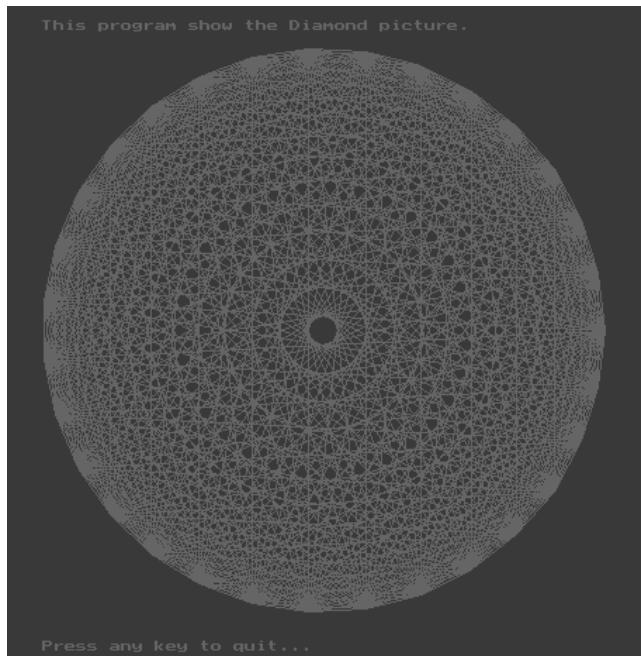


图 114-1 实例 114 程序运行结果

## 实例解析

程序要求输入等分的份数  $n$  和圆周的半径  $r$ , 并根据这两个参数计算圆周上等分点的坐标, 存入  $x$ ,  $y$  数组, 最后对这些等分点用 `line` 函数进行两两连线, 形成所要的“金刚石”图案。

## 程序代码

### 【程序 114】 金刚石图案

//本实例源码参见光盘

## 归纳注释

在本实例程序中, 简单的应用 `line` 函数进行连线, 在等分份数  $n$  为奇数时, 可以用 `moveto` 函数把当前点移到一个等分点上, 再用 `lineto` 一笔绘完整个图案。

# 实例 115 飘带图案

## 实例说明

直线段的起点和终点坐标如果均按余弦和正弦函数的规律变化，则形成具有不同长度和不同位置的线段组成的图案，该图案看起来好像是随风飘扬起来的一条飘带。本实例将实现飘带图案的绘制。程序运行结果如图 115-1 所示。

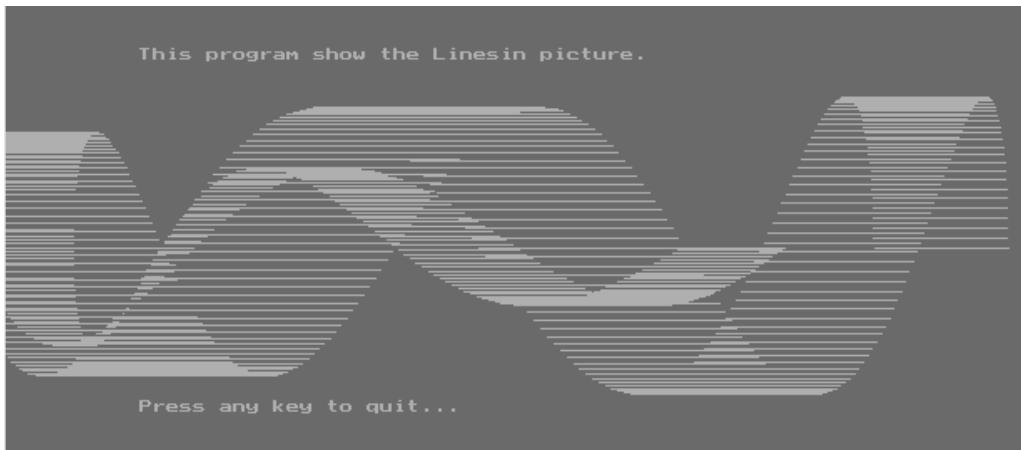


图 115-1 实例 115 程序运行结果

## 实例解析

绘制这样的图案，可采用如下算法：通过 for 循环计算直线段的起点  $x$  轴坐标、 $y$  轴坐标，终止点的  $x$  轴坐标、 $y$  轴坐标（与起点的  $y$  轴坐标相同），这些坐标满足余弦和正弦函数的规律。

$$\begin{aligned}x_1 &= \frac{1}{2}(\max(x)-80) \times \cos(1.6a) + \frac{1}{2}\max(x) \\y_1 &= \max(y) - (90 \times \sin(8a) \times \cos(\frac{a}{2.5}) + \max(\frac{y}{2})) \\x_2 &= \frac{1}{2}(\max(x)-80) \times \cos(1.8a) + \frac{1}{2}\max(x)\end{aligned}$$

其中， $\max(x)$ 是指水平方向的最大分辨率（通常为 640）， $\max(y)$ 是指垂直方向的最大分辨率（通常为 480）， $a \in (a, \pi)$ ，并以  $\pi/380$  的步长递增。

## 程序代码

### 【程序 115】 飘带图案

//本实例源码参见光盘

## 归纳注释

通过改变  $x1$ ,  $y1$ ,  $x2$  计算公式中  $\cos$  和  $\sin$  函数前面的幅值的大小，可以改变飘带的宽度和样式，读者可以试着编写类似的程序。

# 实例 116 圆环图案

## 实例说明

把一个半径为  $R_{large}$  的圆周等分成  $n$  份，然后以每个等分点为圆心，以  $R_{small}$  为半径画  $n$  个圆，将形成环的图案。本实例将实现圆环图案的绘制。程序运行效果如图 116-1 所示。

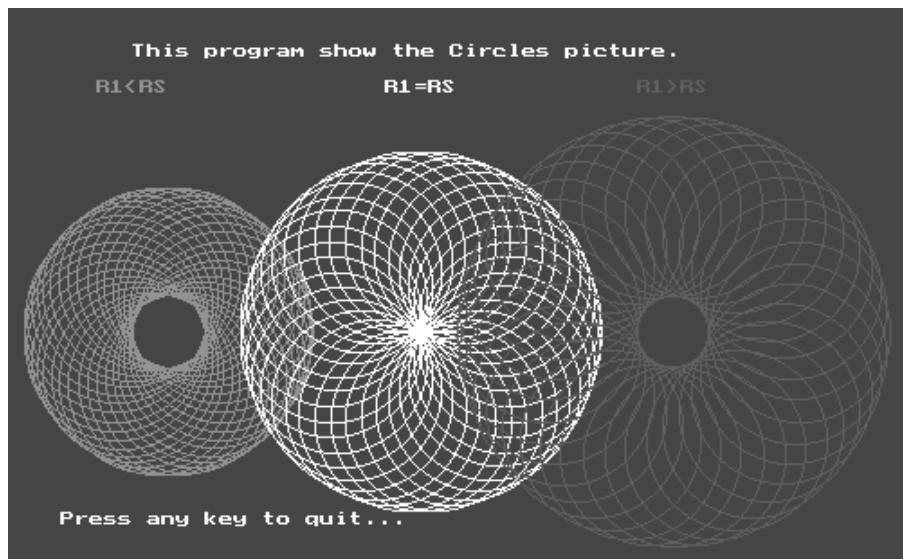


图 116-1 实例 116 程序运行效果

## 实例解析

画圆环图案的算法是在 for 循环中，依次计算  $n$  等分点的坐标，并以该坐标作为圆心，以  $R_{small}$  为半径画圆。这样的圆环有 3 种情况， $R_l > R_s$ 、 $R_l = R_s$  和  $R_l < R_s$ 。

## 程序代码

### 【程序 116】 圆环图案

//本实例源码参见光盘

## 归纳注释

本实例采用 circles 函数来完成圆环图案的绘制。程序首先初始化图形方式，清屏，设置前景色，输出提示信息，改变前景色，输出  $R_l > R_s$  情况的提示，调用 circles 函数画  $R_l > R_s$  情况下的圆弧图案，再依次改变前景色，画  $R_l = R_s$  和  $R_l < R_s$  情况下的圆弧。最后提示按任意键，关闭图形方式，结束程序。

# 实例 117 肾形图案

## 实例说明

本实例实现肾形图案的绘制。程序运行效果如图 117-1 所示。

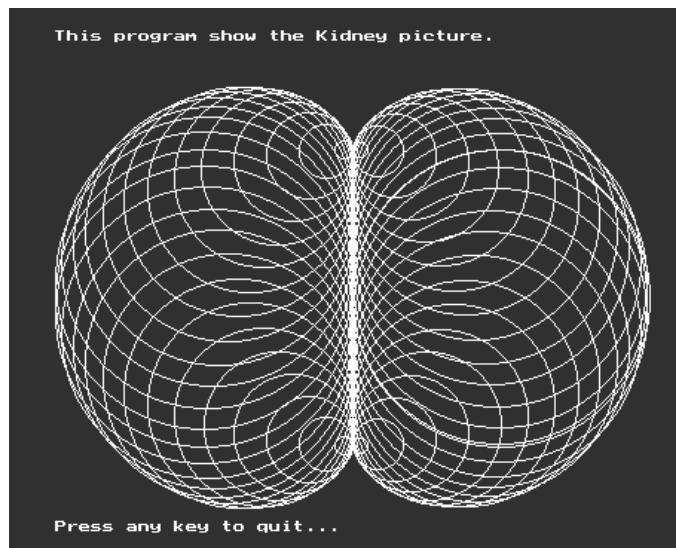


图 117-1 实例 117 程序运行效果

## 实例解析

在实例 116 的圆环图案中，当小圆半径不是定值，而是按规律变化时，将形成肾形图案，即：将一个以  $R$  为半径的圆周等分成  $n$  份，然后以等分点到该圆垂直直径的距离为半径画圆。

根据几何知识可知，圆周上任意点  $A(x_a, y_a)$  到圆心为  $(x_o, y_o)$  的圆  $O$  的垂直直径的距离为  $|x_a - x_o|$ ，即  $A$  点的  $x$  轴坐标与圆心  $x$  轴坐标的差的绝对值，如图 117-2 所示。

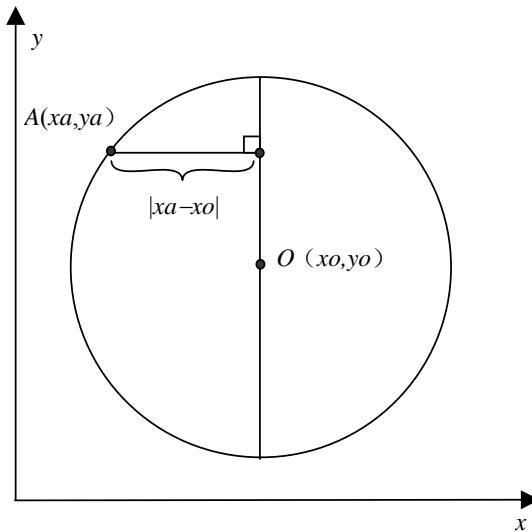


图 117-2 圆周上点到圆的垂直直径的距离示意图



## 程序代码

### 【程序 117】 肾形图案

//本实例源码参见光盘



## 归纳注释

程序首先要求输入圆的半径 R，接着初始化图形方式，清屏，设置背景色和前景色，输出提示信息，在 for 循环中依次计算圆周上的 n 个等分点坐标，并计算该点的小圆半径，画小圆，从而形成肾形图案。最后提示按任意键，关闭图形方式，结束程序。

# 实例 118 心脏形图案



## 实例说明

本实例实现心脏形图案的绘制。程序运行效果如图 118-1 所示。

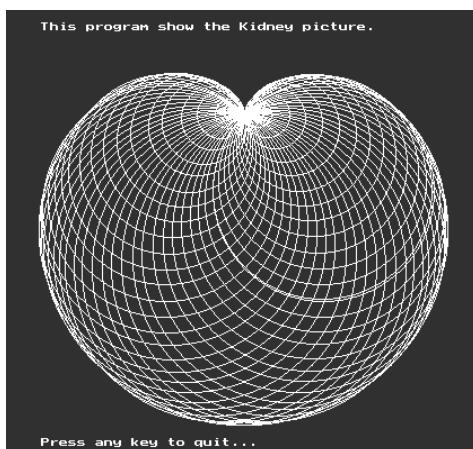


图 118-1 实例 118 程序运行效果



## 实例解析

在实例 117 中，如果等分点到圆周上定点的距离为小圆半径进行绘图，所得到的即为心脏形图案。设等分点坐标为  $(xn, yn)$ ，其到定点  $(xo, yo)$  的距离为：

$$r_i = \sqrt{(xn - xo)^2 + (yn - yo)^2}$$

在本例中，定点取大圆  $O(xo, yo)$ （半径为  $R$ ）的垂直直径的上顶点，即  $(xo, yo-R)$ 。



## 程序代码

### 【程序 118】 心脏形图案

//本实例源码参见光盘



## 归纳注释

本实例的关键在于 for 循环，依次计算各等分点坐标和小圆半径，并画小圆。

# 实例 119 渔网图案



## 实例说明

渔网是用线结成很多网眼组成的。本实例实现渔网图案的绘制。程序运行效果如图 119-1 所示。

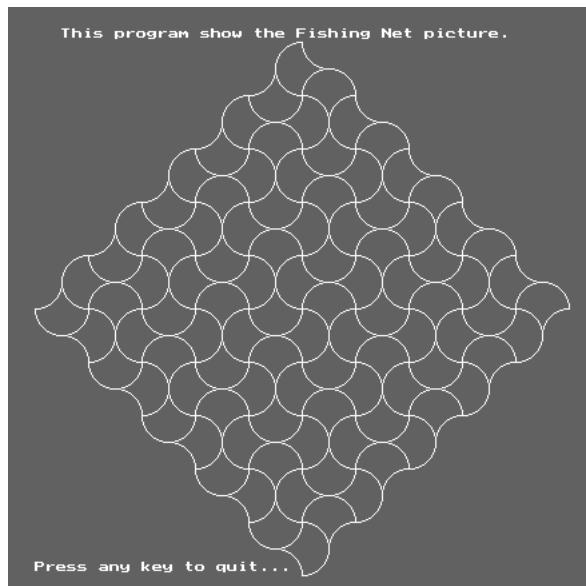


图 119-1 实例 119 程序运行效果



## 实例解析

程序中，所画渔网的网眼是由圆弧组成的，每个网眼由 4 段圆弧组成，每段圆弧是四分之一圆。程序的关键在于确定各段圆弧的圆心位置。



## 程序代码

### 【程序 119】 渔网图案

//本实例源码参见光盘



## 归纳注释

本实例采用圆弧组成渔网，也可以采用椭圆弧来组成，其长短轴方向不同。读者可以在理解本程序规律的基础上，把圆弧改成椭圆弧，绘制渔网。

# 实例 120 沙丘图案

## 实例说明

本实例实现沙丘图案的绘制。程序运行效果如图 120-1 所示。

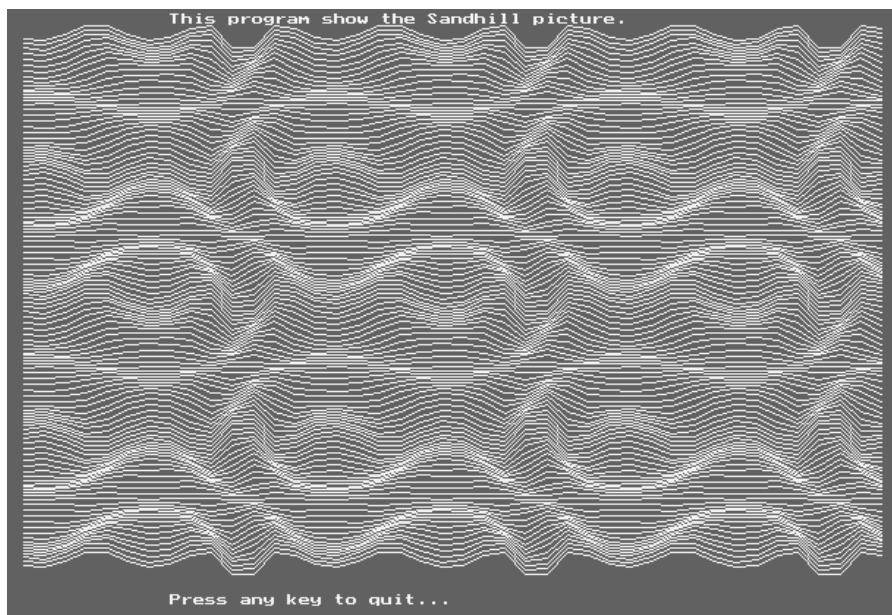


图 120-1 实例 120 程序运行效果

## 实例解析

画沙丘图案时，可采用 `lineto` 函数一笔画成。根据沙丘的变化规律，通过计算 `cos` 和 `sin` 函数确定线段点的坐标，然后调用 `lineto` 函数来绘制。

## 程序代码

### 【程序 120】 沙丘图案

//本实例源码参见光盘

# 实例 121 设置图形方式下的文本类型

## 实例说明

本实例实现对图形方式下的文本类型的设置。程序运行效果如图 121-1 所示。

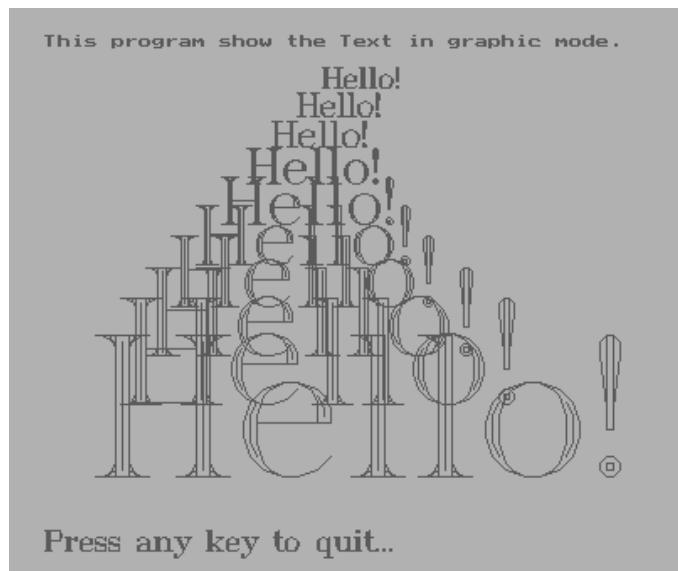


图 121-1 实例 121 程序运行效果

## 实例解析

在图形方式下，BGI 提供了两种输出文本的方式，一种是位映像字符（或称点阵字符）；另一种是笔划字体（或称矢量字符）。位映像字符为缺省方式，即在一般情况下输出文本时，都是以位映像字符显示的。

笔划字体不是以位模式表示的，每个字符被定义为一系列的线段或笔划的组合。笔划字体可以灵活地改变其大小，而且不会降低其分辨率。系统提供了 4 种不同的笔划字体，即小号字体、3 倍字体、无衬线字体和黑体。每种笔划字体都存放在独立的字体文件中，文件扩展名为.chr，一般情况下安装在与 BGI 相同的目录下。为了使用笔划字体，必须装入相应的字体文件。

### 设置文本类型

函数 `settextstyle` 用于在使用笔划字体之前装入字体文件，其调用格式为：

```
settextstyle(int font, int direction, int csize);
```

`font` 用来指定所使用的字体，其取值可为 `DEFAULT_FONT`（或 0），`TRIPLEX_FONT`（或 1），`SMALL_FONT`（或 2），`SANSSERIF_FONT`（或 3），`GOTHIC_FONT`（或 4），其含义分别为 8×8 位图字体（缺省），三重矢量字体，小号矢量字体，无衬线矢量字体，黑体矢量字体。

`direction` 用于指定文本的输出方向，取值为 `HORIZ_DIR`（或 0），表示从左向右输出（缺省），`VERT_DIR`（或 1），表示从上到下输出。

`csize`：用来表示字符的大小。该参数实际上是一个放大系数，它表示对 8×8 点阵的放大倍数，其取值范围是 1~10，它既影响点阵字符，也影响笔划字体。

调用 `settextstyle` 函数后，设置了输出字符的字体、输出方向及大小，这些设置将 `outtext` 和 `outtextxy` 所产生的文本输出。

## 程序代码

### 【程序 121】 设置图形方式下的文本类型

```
//本实例源码参见光盘
```



## 归纳注释

本实例程序的运行效果是：对用户的问候语“Hello!”由小到大逐渐推向屏幕中央。程序中的函数 `delay` 用于设置延迟执行时间，因为如果执行太快将看不到字符串变化的过程。延迟时间可通过开始时输入的 `t` 值来控制。

# 实例 122 绘制正多边形



## 实例说明

本实例实现正多边形的绘制。程序运行效果如图 122-1 所示。

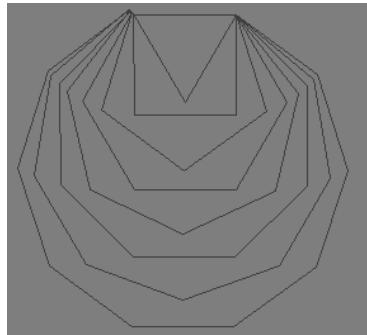


图 122-1 实例 122 程序运行效果



## 实例解析

要绘制一个图形，首先要明确绘制这个图形需要哪些数据。一般来说，绘制一个惟一确定的图形，需要有两种数据：一种是确定图形形状的数据，称为定形参数；另一种是确定图形画在什么地方的位置数据，称为定位参数。

对于一个任意正多边形来说，它需要的参数如图 122-2 所示。

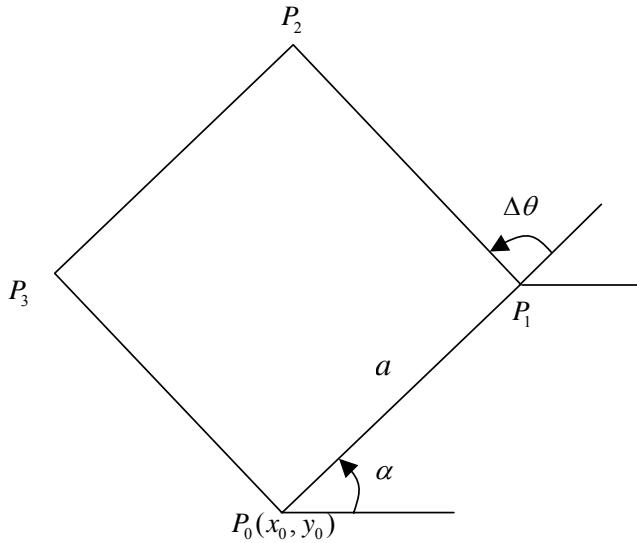


图 122-2 正多边形的定形参数

定形参数为①边数  $n$ ,  $n$  不能小于 3。②边长  $a$ 。

定位参数为①起始顶点  $P_0$  的坐标  $(x_0, y_0)$ 。②起始边的倾斜角  $\alpha$ 。

给定了这两组参数，就可以惟一地确定一个任意位置和形状的正多边形。但在绘图程序中，必须把这两组参数转化为具体的绘图数据，对于正多边形来说，具体的绘图数据就是多边形各顶点的坐标。

根据图形中的几何关系，可以列出计算正多边形各顶点的坐标计算公式如下：

$$P_0: (x_0, y_0)$$

$$P_1: \begin{cases} x_1 = x_0 + a \cos \alpha \\ y_1 = y_0 + a \sin \alpha \end{cases}$$

$$P_2: \begin{cases} x_2 = x_1 + a \cos(\alpha + \Delta\theta) \\ y_2 = y_1 + a \sin(\alpha + \Delta\theta) \end{cases}$$

.....

$$P_i: \begin{cases} x_i = x_{i-1} + a \cos(\alpha + (i-1)\Delta\theta) \\ y_i = y_{i-1} + a \sin(\alpha + (i-1)\Delta\theta) \end{cases}$$

其中  $\Delta\theta = 2\pi / n$ 。



## 程序代码

### 【程序 122】 绘制正多边形

//本实例源码参见光盘



## 归纳注释

程序中，`polygon(x0,y0,a,n,af)`函数用于绘制任意正多边形，其参数  $(x_0, y_0)$  为定位角点坐标， $a$  为正多边形边长， $n$  为正多边形边数， $af$  为起始边与  $x$  轴正向的夹角（度）。

# 实例 123 正六边形螺旋图案



## 实例说明

本实例采用外接圆来构造正六边形的螺旋图案。程序运行效果如图 123-1 所示。

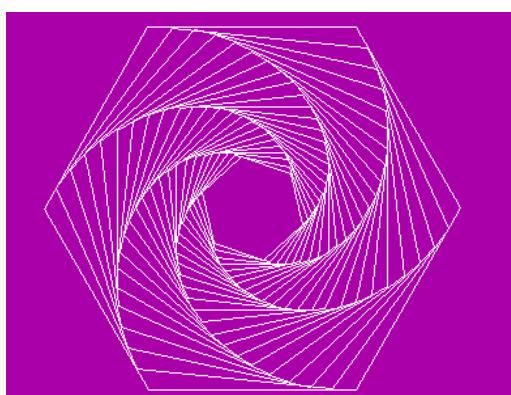


图 123-1 实例 123 程序运行效果



## 实例解析

除了实例 122 中所采用的方法，另外一种构造正多边形的方法是利用正多边形的外接圆来构造，这在有些应用场合使用起来更方便，如图 123-2 所示。

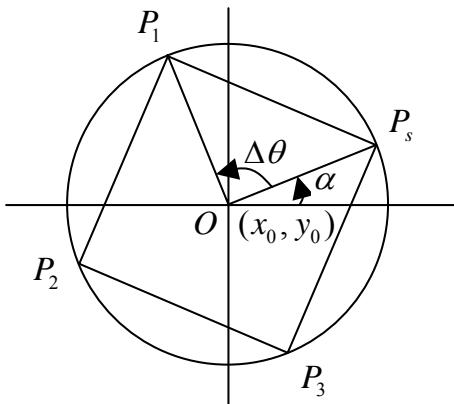


图 123-2 正多边形和外接圆

在这种构造方式中，定形参数为①边数  $n$ ， $n$  不能小于 3。②外接圆半径  $R$ 。

定位参数为①外接圆圆心坐标  $(x_0, y_0)$ 。②起始点半径的倾角  $\alpha$ 。

根据这些参数，同样可以列出计算多边形各顶点坐标的公式如下：

$$P_s : \begin{cases} x_s = x_0 + R \cos \alpha \\ y_s = y_0 + R \sin \alpha \end{cases}$$

$$P_1 : \begin{cases} x_1 = x_0 + R \cos(\alpha + \Delta\theta) \\ y_1 = y_0 + R \sin(\alpha + \Delta\theta) \end{cases}$$

.....

$$P_i : \begin{cases} x_i = x_0 + R \cos(\alpha + i\Delta\theta) \\ y_i = y_0 + R \sin(\alpha + i\Delta\theta) \end{cases}$$

其中  $\Delta\theta = 2\pi / n$ 。



## 程序代码

### 【程序 123】 正六边形螺旋图案

// 本实例源码参见光盘



## 归纳注释

本例程序中，`polygonc(int x0, int y0, int r, int n, float af)` 函数用于绘制任意正多边形，其参数  $(x_0, y_0)$  为正多边形外接圆心坐标， $r$  为外接圆半径， $n$  为正多边形边数， $af$  为起始点半径与  $x$  轴正向的夹角（度）。

在绘制螺旋正六边形时，关键是相邻两个正六边形的外接圆半径的关系，根据图形的几何关

系可得  $r = \sqrt{3} / 2 \times r / \cos((30 - \text{theta}) \times 0.0174533)$ 。

主程序中，外接圆半径  $r$ 、组成螺旋的正六边形个数  $n$ 、两个相邻正六边形之间的转角  $\text{theta}$ ，也可以通过用户输入来自定义。

## 实例 124 正方形螺旋拼块图案

### 实例说明

正方形螺旋块是由基本图形正方形经过变换而构成的。图 124-1 所示的图案是由 16 个相似的方块拼接而成的，每一个方块是由一个正方形形成的螺旋图案（类似于实例 123 中的正六边形螺旋图），而每个相邻方块的螺旋方向相反。本实例实现对正方形螺旋拼块图案的绘制。程序运行效果如图 124-1 所示。

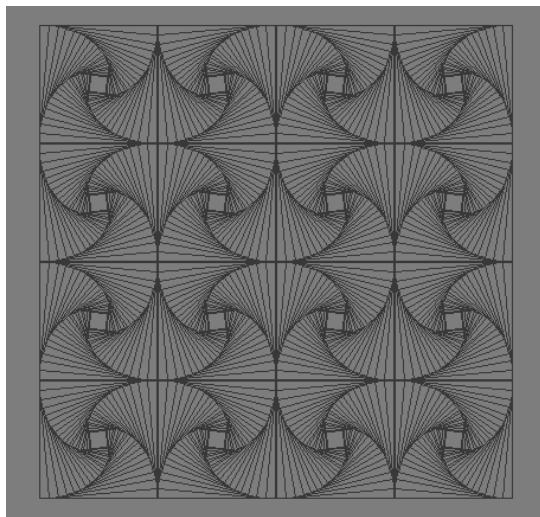


图 124-1 实例 124 程序运行效果

### 实例解析

绘制螺旋正方形的关键是相邻两个正方形之间的关系，如图 124-2 所示。

设两个正方形的外接圆半径分别为  $R_1$  和  $R_2$ ，边长为  $a_1$  和  $a_2$ ，旋转角度为  $\theta$ ，则两个正方形之间的缩小系数为  $f = a_2/a_1$ 。因为经过旋转和缩小后的正方形 2 的 4 个顶点刚好位于正方形 1 的 4 条边上，所以，这个缩小系数  $f$  和转角  $\theta$ 紧密相关。从图 124-2 中可以看出，在直角三角形 PQS 中， $QS = a_2 \sin \theta$ ， $PS = a_2 \cos \theta$ ， $QS + PS = a_1 = a_2(\sin \theta + \cos \theta)$ ，所以可得：

$$f = a_2 / a_1 = 1 / (\sin \theta + \cos \theta)$$

考虑到转角  $\theta$ 可以取正值（逆时针旋转）或负值（顺时针旋转），但缩小系数  $f$  总是正值，所以在求  $f$  时， $\theta$ 必须取绝对值，即：

$$f = 1 / (\sin |\theta| + \cos |\theta|)$$

则两个正方形的外接圆半径之间的关系也是： $R_2=f\times R_1$ 。  
且外接圆半径与正方形边长关系  $R=a/\sqrt{2}$ ，转角关系为  $\alpha_2=\alpha_1+\theta$ 。

从以上分析，可以编程如下：

① 绘制正多边形函数，采用与实例 123 相同的 polygonc 函数。

② 绘制方块函数 block(int x, int y, int a, int n, int theta)，用于绘制正向和反向的正方形螺旋图案。该函数根据旋转角度 theta 和缩小比例系数 f，以不同的实参调用 polygonc 函数。其中 (x, y) 为方块左下角定位点坐标，a 为方块的边长，n 为组成方块的正方形数量，theta 为两个相邻正方形之间的转角（单位为度）。

③ 把方块拼成要求的图案：这是由 main 函数完成，主要是计算每个方块的定位点坐标，然后调用不同旋转方向的方块绘制函数 block。

## 程序代码

### 【程序 124】 正方形螺旋拼块图案

//本实例源码参见光盘

## 归纳注释

程序首先要求输入图案的长度 length，正方形螺旋方块中的正方形个数 n，以及相邻两个正方形之间的转角 theta，然后初始化图形方式，接着在两个 for 循环中计算方块的定位点坐标，改变螺旋方向，调用 block 函数进行螺旋方块的绘制。最后按任意键关闭图形方式，结束程序。

## 实例 125 图形法绘制圆

## 实例说明

本实例采用图形学画圆算法进行圆的绘制。程序运行效果如图 125-1 所示。

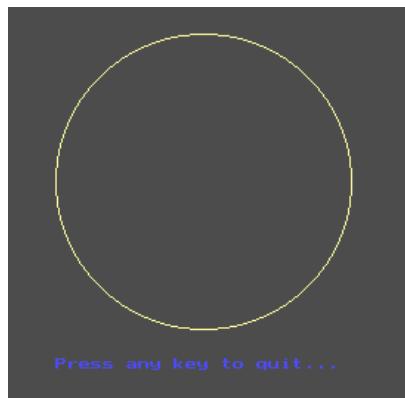


图 125-1 实例 125 程序运行效果



## 实例解析

本实例使用图形学中的算法画圆，比起传统的画圆方法（逐点计算  $x, y$  坐标值）极大地缩减了程序的运算量。因为逐点计算时，每个点都要进行两次乘法和一次三角运算；而在这个算法中，只是计算从 $(0,r)$ 点开始每次  $x$  坐标加 1 时，一个相关参数  $p$  的变化， $p$  的值决定了  $y$  坐标在下一个点要减少的值为 0 或为 1。算法具体描述如下。

根据输入的半径  $r$  和圆心  $x, y$  选定画圆的初始点 $(0,r)$ 。

参数  $p$  设定初值  $p=5/4-r$ 。

当  $p < 0$  时，则下一个点为 $(x+1,y)$ ， $p=p+2x+3$ 。

当  $p > 0$  时，则下一个点为 $(x+1,y-1)$ ， $p=p+2x-2y+5$ 。

如果  $x > y$ ，则程序结束，画成了八分之一的圆弧。最后再通过对称的办法画完整个圆。



## 程序代码

### 【程序 125】 图形学画圆

```
#include <stdio.h>
#include <graphics.h>
#include <dos.h>
/*这是根据给出的圆心坐标和点坐标分别在 8 个象限画点的子程序*/
void circlePoint( int xCenter, int yCenter, int x, int y )
{
    putpixel( xCenter + x, yCenter + y, YELLOW );
    putpixel( xCenter - x, yCenter + y, YELLOW );
    putpixel( xCenter + x, yCenter - y, YELLOW );
    putpixel( xCenter - x, yCenter - y, YELLOW );
    putpixel( xCenter + y, yCenter + x, YELLOW );
    putpixel( xCenter - y, yCenter + x, YELLOW );
    putpixel( xCenter + y, yCenter - x, YELLOW );
    putpixel( xCenter - y, yCenter - x, YELLOW );
}

void myCircle(int xCenter,int yCenter,int radius)
{
    int x, y, p;
/*初始化各个参数*/
    x = 0;
    y = radius;
    p = 1 - radius;
    circlePoint(xCenter, yCenter, x, y);
/*循环中计算圆上的各点坐标*/
    while( x < y ) {
        x++;
        if( p < 0 )
            p += 2*x+1;
        else
        {
            y--;
            p+=2*(x-y)+1;
        }
    }
}
```

```

        circlePoint( xCenter, yCenter, x, y);
    }
}

void main()
{
    int gdriver=DETECT, gmode; /*这是用 c 画图时必须要使用的图像入口*/
    int i;
    int xCenter, yCenter, radius;
    printf("Please input center coordinate :(x,y) ");
    scanf("%d,%d", &xCenter, &yCenter );
    printf("Please input radius : ");
    scanf("%d", &radius );
/*这条语句初始化整个屏幕并把入口传给 gdriver,注意引号中是 tc 中 bgi 目录的完整路径*/
    registerbgidriver(EGAVGA_driver);
    initgraph(&gdriver, &gmode, "c:\\tc");
    setcolor( BLUE );
    myCircle(xCenter, yCenter, radius);
    getch();
    closegraph();
    return;
}

```



### 归纳注释

本实例采用图形学的方法画圆，而不是调用 Turbo C 的 circle 函数来画。可以说，应用 putpixel(int x, int y, int color) 函数可以画出所有的图形（该函数是在点 (x,y) 画一个颜色为 color 的点）。但是，这个函数是一个像素一个像素的画，效率太低。因此，Turbo C 提供了众多的库函数来画各种图形。在这些库函数不能满足需要时，可采用 putpixel 函数来画所需要的图形。

## 实例 126 递归法绘制三角形图案



### 实例说明

本实例用递归方法绘制三角形图案。程序运行效果如图 126-1 所示。

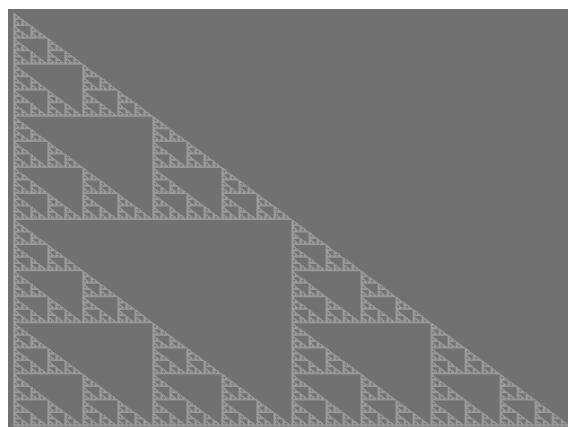


图 126-1 实例 126 程序运行效果



## 实例解析

C 语言允许函数之间的递归调用，利用这种特点，也可以设计出构思巧妙的图案。整个递归三角形图案是由大小不同的许多三角形组成的。除了最外面那个大三角形外，其余的三角形都由连接包含它的三角形的三边中点构成的。但是，这种连接三角形的三边中点是有选择的，因为如果没有选择地全部连线，那未必会形成如图 126-1 所示的那样许多大小不等的三角形。这种选择的方法如图 126-2。在三角形 ABC 中，连结三边中点形成三角形 PQR，这样就最终产生 4 个小三角形： $\triangle PQR$ ， $\triangle APR$ ， $\triangle PBQ$ ，和 $\triangle PQC$ 。在这 4 个小三角形中，只将其中 3 个位于角上的小三角形（即 $\triangle APR$ ， $\triangle PBQ$ ， $\triangle PQC$ ）连接其三边中点形成新的三角形。

程序中，主要使用一个绘制三角形的函数 tria (int xa, int ya, int xb, int yb, int xc, int yc, int n)。该函数用来连接一个三角形的三边中点绘制一个三角形，同时又递归调用自身，在形成的 3 个角上的三角形中连接三边中点构成三角形。



## 程序代码

### 【程序 126】 递归法绘制三角形

//本实例源码参见光盘



## 归纳注释

主程序首先要求输入递归调用的深度，接着初始化图形方式，清屏，设置背景色和前景色，画最外面的大三角形，调用函数 tria 递归绘制三角形，最后按任意键，关闭图形方式，结束程序。函数 void tria (int xa, int ya, int xb, int yb, int xc, int yc, int n) 中参数(xa,ya), (xb,yb), (xc,yc)为三角形 3 个顶点坐标，n 为递归深度。

## 实例 127 图形法绘制椭圆



## 实例说明

本实例实现用图形法绘制椭圆。程序运行效果如图 127-1 所示。

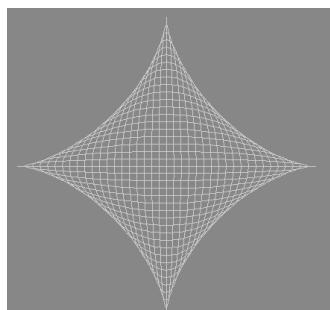


图 127-1 实例 127 程序运行效果



## 实例解析

### 绘制曲线的基本方法

平面解析几何中，把一条曲线中的点  $(x, y)$  满足的关系  $y=f(x)$  函数式称为曲线的方程，同样，该曲线即为这个方程的曲线。例如圆的方程为  $x^2+y^2=R^2$ ，椭圆的方程为  $x^2/a^2+y^2/b^2=1$ 。在绘制这些曲线时，可以借助各种标志工具，比如画圆可以用圆规，画椭圆可以用椭圆规，但是对于非圆曲线，绘制时的更一般的方法是借助于曲线板。先在平面上确定一些满足条件的、位于曲线上的坐标点，然后借用曲线板把这些点分段光滑地连接成曲线。绘出的曲线的精确程度，取决于所选择的数据点的精度和数量，坐标点的精度越高、点的数量取得越多，连成的曲线越接近于理想曲线。

其实，这个方法也是用计算机来绘制各种曲线的基本原理，即把曲线离散化，把它们分割成很多短直线段，用这些短直线段组成的折线来逼近曲线。至于这些短直线段取多长，则取决于图形输出设备的精度和绘制的曲线所要求的精度，但所要求的精度不能逾越图形设备实际具备的精度。

### 椭圆曲线的绘制

椭圆的标准方程：

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1$$

其中， $a$  和  $b$  分别为椭圆的长、短半轴。在实际绘图中，采用参数方程来表示椭圆更为方便，即：

$$\begin{cases} x = f(t) = a \times \cos t \\ y = g(t) = b \times \sin t \end{cases} \quad (0 \leq t \leq 2\pi)$$

式中， $t$  为参数变量，它的取值范围从  $0 \sim 2\pi$ ，即一个圆周。可以看出，这个参数的实际意义是椭圆上的点所对应的中心角。这样，根据参数方程，每确定一个  $t$  值，就可以计算得到对应于  $t$  值的位于该椭圆上的一个确定点  $(x, y)$ 。当  $t=t_i$  时，可以得到椭圆上一点  $(x_i, y_i)$ ：

$$\begin{cases} x_i = a \times \cos t_i \\ y_i = b \times \sin t_i \end{cases}$$

让参变量  $t$  增加一个增量  $\Delta t$ ，使  $t_{i+1}=t_i+\Delta t$ ，代入参数方程，得另一点  $(x_{i+1}, y_{i+1})$ ：

$$\begin{cases} x_{i+1} = a \times \cos t_{i+1} \\ y_{i+1} = b \times \sin t_{i+1} \end{cases}$$

连接两点  $(x_i, y_i)$  和  $(x_{i+1}, y_{i+1})$ ，便可以近似地认为绘制了椭圆上的一段弧，如图 127-2 所示。由于计算机的运算速度很快，并且运算精度极高，所以可以很容易地把  $\Delta t$  值取得很小，以便能获得更多的坐标点，这样就可以使绘制出的椭圆更为精确，看上去是一条极为光滑的曲线。

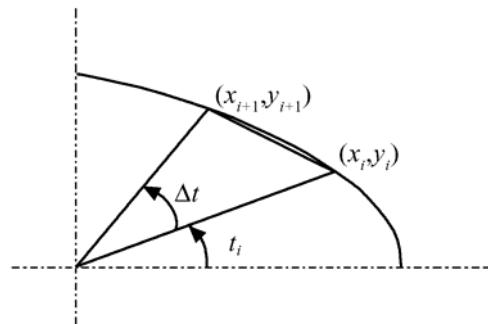


图 127-2 椭圆弧



## 程序代码

### 【程序 127】 图形法绘制椭圆

//本实例源码参见光盘



## 归纳注释

程序用自定义的椭圆绘制函数 `ellipse1(int x0, int y0, int a, int b, int dt)` 绘制了椭圆群的星形图案。该函数的参数 ( $x_0, y_0$ ) 为椭圆的中心坐标,  $a, b$  分别为椭圆的长、短半轴,  $dt$  为参变量的增量, 即  $\Delta t$ 。

# 实例 128 抛物样条曲线



## 实例说明

本实例实现抛物样条曲线的绘制。程序运行效果如图 128-1 所示。

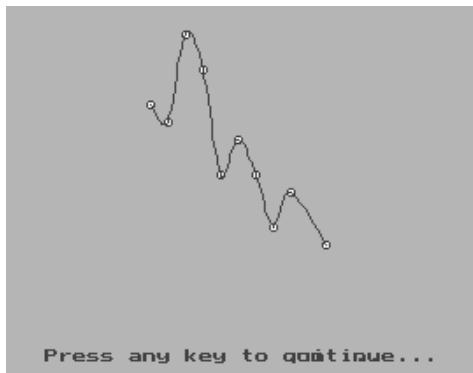


图 128-1 实例 128 程序运行效果



## 实例解析

### 抛物样条曲线

工程上通常需要把一系列的离散的测量点用一条光滑的曲线连接起来, 绘制成曲线图形。在拟合生成曲线的众多方法中, 一般总要选择一种简单一些的曲线, 作为拟合生成其他曲线的基本曲线, 然后对这种基本曲线做一些适当的数学处理, 来生成完整的拟合曲线。抛物样条曲线, 顾名思义, 就是选择抛物线这样一种较为简单的二次曲线作为基本曲线来拟合给定离散型值点生成的曲线。

设有不在同一直线上的 3 点:  $P_1(x_1, y_1)$ ,  $P_2(x_2, y_2)$ ,  $P_3(x_3, y_3)$ , 则通过该给定的 3 点定义的一条抛物线的方程为:

$$\begin{aligned}
P(t) &= A_1 + A_2 t + A_3 t^2 \\
&= (2t^2 - 3t + 1)P_1 + (4t - 4t^2)P_2 + (2t^2 - t)P_3 \quad (0 \leq t \leq 1)
\end{aligned}$$

写成矩阵的参数方程形式为：

$$[x(t) \ y(t)] = \begin{bmatrix} t^2 & t & 1 \end{bmatrix} \begin{bmatrix} 2 & -4 & 2 \\ -3 & 4 & -1 \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 & y_1 \\ x_2 & y_2 \\ x_3 & y_3 \end{bmatrix}$$

这样，根据参变量  $t$  的取值，就可以一一计算出位于曲线上的数据点，然后顺次连线绘出图形。

### 抛物线加权合成

设有一离散型值点列  $P_i (i=1, 2, \dots, n)$ ，则可以按上面的方程每经过相邻 3 点作一段抛物线，一共可以作  $n-2$  条抛物线段。设其中第  $i$  条抛物线段为经过  $P_i, P_{i+1}, P_{i+2}$  3 点，其表达式为：

$$S_i(t_i) = (2t_i^2 - 3t_i + 1)P_i + (4t_i - 4t_i^2)P_{i+1} + (2t_i^2 - t_i)P_{i+2} \quad (0 \leq t \leq 1)$$

同理，第  $i+1$  条抛物线为经过  $P_{i+1}, P_{i+2}, P_{i+3}$  3 点，其表达式为（见图 128-2）：

$$S_{i+1}(t_{i+1}) = (2t_{i+1}^2 - 3t_{i+1} + 1)P_{i+1} + (4t_{i+1} - 4t_{i+1}^2)P_{i+2} + (2t_{i+1}^2 - t_{i+1})P_{i+3} \quad (0 \leq t \leq 1)$$

一般来说，每两段曲线之间的搭接区间，两条抛物线是不可能重合的。为了用一条光滑的曲线拟合整个型值点列，必须有一个办法让这些抛物线段按照一定的法则结合成一条曲线，这种结合的办法就是加权合成。

在加权合成的过程中，要选择两个合适的权函数  $f(T)$  和  $g(T)$ ，则加权合成后的曲线为  $P_{i+1}(t)$ ：

$$P_{i+1}(t) = f(T) \cdot S_i(t_i) + g(T) \cdot S_{i+1}(t_{i+1})$$

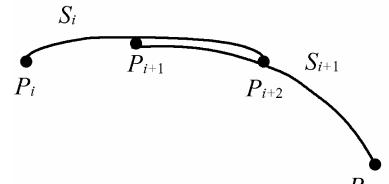


图 128-2  $S_i$  和  $S_{i+1}$

取权函数为  $f(T)=1-T$ ,  $g(T)=T$ ,  $0 \leq T \leq 1$ , 另取  $T=2t$ ,  $t_i=0.5+t$ ,  $t_{i+1}=t$ , 则有：

$$\begin{aligned}
P_{i+1}(t) &= (-4t^3 + 4t^2 - t)P_i + (12t^3 - 10t^2 + 1)P_{i+1} \\
&\quad + (-12t^3 + 8t^2 + t)P_{i+2} + (4t^3 - 2t^2)P_{i+3} \quad (i=1, 2, \dots, n-3) \quad (0 \leq t \leq 0.5)
\end{aligned}$$

上式表达了每相邻的 4 个点可以决定中间的一段抛物样条曲线。

### 抛物样条曲线的端点条件

$n$  个型值点  $P_i$  可以加权合成生成  $n-3$  段抛物样条曲线，但  $n$  个型值点之间有  $n-1$  个区段，其首尾两段曲线  $P_1P_2$  和  $P_{n-1}P_n$  段，由于缺乏连续相邻的 4 点这样的条件而无法产生。为此，必须在两端各加一个辅助点  $P_0$  和  $P_{n+1}$ 。问题是这两点是如何加上去的，它依据什么原则，这就是所谓的“端点条件”。这里，仅介绍两种常用的方法。

(1) 自由端条件。这种方法的原理比较简单，它让所补之点  $P_0$  和  $P_{n+1}$  与原两端点  $P_1$  和  $P_n$  分别重合，即  $P_0=P_1$ ,  $P_{n+1}=P_n$ 。这样的补点方法称为自由端条件，这种方法一般适用于对曲线的两端没有什么特殊的要求的情况。

(2) 形成封闭曲线。为了在  $n$  个型值点之间形成封闭曲线，就要生成  $n$  段曲线段，而不是原

理的  $n-1$  段。所以，在补点工作中要加上 3 个点，首先让首尾两点重合，然后各向前后延长一点，即  $P_{n+1}=P_1$ ,  $P_0=P_n$ ,  $P_{n+2}=P_2$ 。



## 程序代码

### 【程序 128】 抛物样条曲线

// 本实例源码参见光盘



## 归纳注释

程序通过读取文件中点的坐标来绘制抛物样条曲线。抛物样条曲线的绘制函数 void parspl(int p[][2], int n, int k, int e) 的参数 p 是型值点的坐标数组，n 是型值点数，k 是插值数，即把参变量 t 区间细分的份数，e 是端点条件，e=1 时，为自由端点，e=2 时为画封闭曲线。函数 marking 的作用是对型值点进行标记，在调试程序时便于核对曲线程序是否正确，在实际应用中是不需要的。

# 实例 129 Mandelbrot 分形图案



## 实例说明

本实例实现 Mandelbrot 分形图案的绘制。程序运行效果如图 129-1 所示。

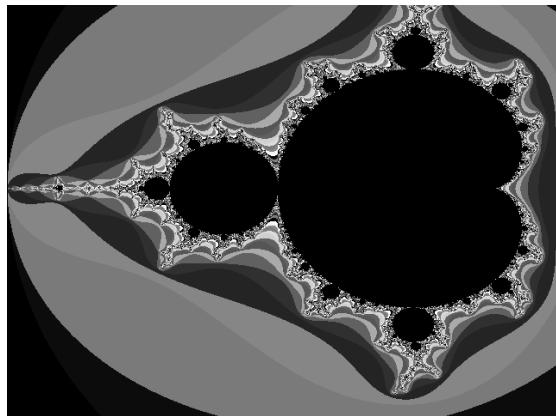


图 129-1 实例 129 程序运行效果



## 实例解析

分形图形是数学中一种很奇妙的现象，它们一般都可以通过简单的数学公式描述产生无比复杂和美妙的图形。而且分形图形的一个最为奇妙之处就是它可以被无限细分，把它的某一个细节放大后会得到和原来图形一样的效果。Mandelbrot 分形就是最著名的分形图形之一，它是通过使用复数的迭代运算来得到图形的形状和颜色，虽然公式很简单，但是得到的结果却让人惊叹不已。

Mandelbrot 分形的原理如下。

用迭代公式：

$$\begin{cases} z[0] = zInit \\ z[k] = z[k-1] \times z[k-1] + z[0] \end{cases}$$

其中  $z[i]$  是复数，计算时要使用复数的运算法则。

Mandelbrot 图形集的初始化要求：

$$\begin{aligned} -2.25 &\leq \operatorname{Re}(zInit) \leq 0.75 \\ -1.25 &\leq \operatorname{Im}(zInit) \leq 1.25 \end{aligned}$$

其中  $\operatorname{Re}(z)$  表示  $z$  的实部， $\operatorname{Im}(z)$  表示  $z$  的虚部。

为了方便起见，最终的图形是画在复平面上的，以  $x$  轴为实部， $y$  轴为虚部，并用该点的迭代次数作为该点的颜色。当图像上每一个点的迭代次数越多时，整个图形也就越清晰。



## 程序代码

### 【程序 129】 Mandelbrot 分形图案

```
#include <graphics.h>

typedef struct { float x, y; } complex; /* 定义复数的结构，x 表示实部，y 表示虚部 */

complex complexSquare( complex c )
/* 计算复数的平方
(x+yi)^2 = (x^2-y^2) + 2xyi */
{
    complex csq;
    csq.x = c.x * c.x - c.y * c.y;
    csq.y = 2 * c.x * c.y;
    return csq;
}

int iterate( complex zInit, int maxIter )
/* 迭代计算颜色，maxIter 是最多迭代的次数 */
{
    complex z = zInit;
    int cnt = 0;

    /* 当 z*x > 4 的时候退出 */
    while((z.x * z.x + z.y * z.y <= 4.0) && (cnt < maxIter))
    {
        /* 迭代公式：z[k] = z[k-1]^2 + zInit, cnt 是迭代次数 */
        z = complexSquare( z );
        z.x += zInit.x;
        z.y += zInit.y;
        cnt++;
    }
    return cnt;
}

void mandelbrot( int nx, int ny, int maxIter, float realMin, float realMax, float
imagMin, float imagMax )
/* 画 Mandelbrot 图形的主程序，参数意义如下 */
```

```

nx: x 轴的最大值
ny: y 轴的最大值
maxIter: 迭代的最大次数
realMin: 初值 zInit 的实部最小值
realMax: 初值 zInit 的实部最大值
imagMin: 初值 zInit 的虚部最小值
imagMax: 初值 zInit 的虚部最大值
*/
{
    float realInc = (realMax - realMin) / nx; /*x 轴迭代的步长*/
    float imagInc = (imagMax - imagMin) / ny; /*y 轴迭代的步长*/
    complex z; /*初值 zInit*/
    int x, y; /*点(x,y)的横纵坐标*/
    int cnt; /*迭代的次数*/

    for( x = 0, z.x = realMin; x<nx; x++, z.x += realInc )
    {
        for( y = 0, z.y = imagMin; y < ny; y++, z.y+= imagInc )
        {
            cnt = iterate( z, maxIter ); /*计算迭代次数*/
            if( cnt == maxIter ) /*当迭代最大时, 为黑色*/
                putpixel( x, y, BLACK );
            else /*否则将迭代次数作为颜色*/
                putpixel( x, y, cnt );
        }
    }
}

void main()
{
    int gdriver = 9, gmode=2;
    /*registerbgidriver( EGAVGA_driver );*/
    initgraph( &gdriver, &gmode, "e:\tc\bgi" );
    mandelbrot( 640, 480, 255, -2.0, 0.55, -1.0, 1.25 );
    getch();
    closegraph();
}

```



## 归纳注释

由于 Turbo C 提供的图形驱动的限制, 因此最多只能使用到 256 色, 也就是 256 次迭代。同时, 迭代次数越多, 描绘的速度越慢。

## 实例 130 绘制布朗运动曲线



### 实例说明

本实例实现布朗运动曲线的绘制。程序运行效果如图 130-1 所示。

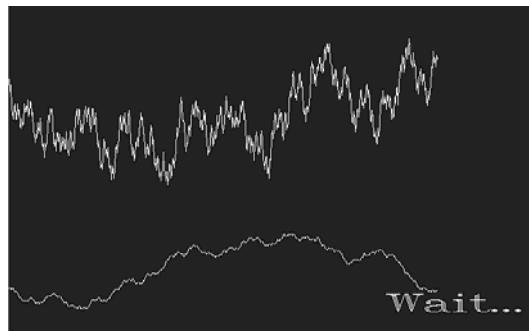


图 130-1 实例 130 程序运行效果

## 实例解析

悬浮在液体或气体中的微粒所做的永不停息的无规则运动，叫做布朗运动。作布朗运动的微粒（直径约为  $10^{-5}\sim 10^{-3}$  cm）称为布朗微粒。布朗运动是英国植物学家布朗于 1827 年观察悬浮在溶液中花粉运动时发现的。这些小的颗粒，为液体的分子所包围，由于液体分子的热运动，小颗粒受到来自各个方向液体分子的碰撞，布朗粒子受到不平衡的冲撞，而做沿冲量较大方向的运动。又因为这种不平衡的冲撞，使布朗微粒得到的冲量不断改变方向。所以布朗微粒做无规则的运动。

本实例程序主要采用分形技术来绘制一维的布朗运动曲线。

## 程序代码

### 【程序 130】 布朗运动曲线

// 本实例源码参见光盘

## 实例 131 艺术清屏

## 实例说明

在编制程序时，经常要用到清屏处理，例如 DOS 下的 `cls()`，Turbo C 下的 `clrscr()` 函数等都具有清屏功能。但这些函数或命令都是一般意义的清屏，并没有显示其清屏规律。本例利用 C 语言编制了几个子函数，分别用于实现上清屏、下清屏、中清屏等要求，这样既可以达到清屏的目的，又能增加屏幕的艺术美观。程序运行效果如图 131-1 所示。

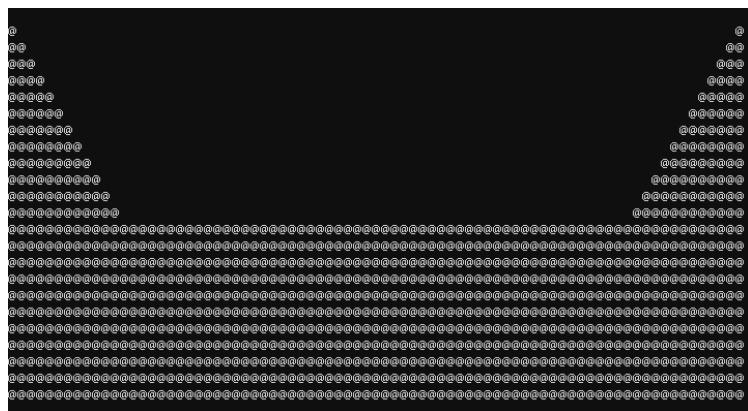


图 131-1 实例 131 程序运行效果



## 实例解析

程序运行时，首先调用 `puta()`子函数将屏幕画满字符@，然后调用水平、垂直和中心清屏3个子程序，使用不同的循环方法消除屏幕上的字符。



## 程序代码

### 【程序 131】 艺术清屏

//本实例源码参见光盘



## 归纳注释

本实例程序的几个主要子函数 `zcls`、`recls`、`bcls` 和 `dcls`，分别用于实现中心清屏、矩形清屏、闭幕清屏和自下清屏等功能，主函数调用只是演示程序，具体应用时可根据需要修改调用。

# 实例 132 矩形区域的颜色填充



## 实例说明

本实例实现矩形区域的颜色填充。程序运行效果如图 132-1 所示。

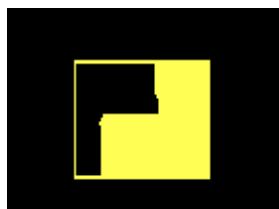


图 132-1 实例 132 程序运行效果



## 实例解析

Windows 自带的“画图”程序有一个“小灌筒”形状的画图工具，用来对一个封闭区域进行填充。本例也实现与其类似的功能，对程序中设定的一个矩形区域进行颜色填充。本例使用的算法是种子算法，算法的大致思想是，使用递归的方法，对每一个点读取它的颜色值，如果和填充色不同则在该点画点，然后继续依次判断它的上下左右点是否已经被画过，直到退回最初的递归调用。种子算法是图形学中一个很常用的解决问题的方法，有兴趣的读者可以参阅相关书籍。



## 程序代码

### 【程序 132】 矩形区域的颜色填充

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

```

#include <graphics.h>

void main()
{int gd=VGA,gm=VGALO;
 /*registerbgidriver(EGAVGA_driver);*/
 initgraph(&gd,&gm,"e:\\tc\\bgi"); /*设置图形模式*/
 setcolor(YELLOW);
 rectangle(105,105,175,135); /*画正方形*/
 full(120,120,YELLOW); /*调填充函数*/
 getch(); /*等待*/
 closegraph(); /*关闭图形模式*/
}

#define DELAY_TIME 5/*填充点后延长的时间，用来观看填充的过程，单位为毫秒*/

int full(int x,int y,int color1)/*递归的填充函数*/
{int color2,x1,y1;
 x1=x; y1=y;
 if(kbhit())return;
 color2=getpixel(x1,y1); /*读(x,y)点颜色值*/
 if(color2!=color1) /*判断是否与填充色相等*/
 {putpixel(x1,y1,color1); /*画点(x1,y1) */
 delay(DELAY_TIME);
 getch();
 x1++;
 full(x1,y1,color1); /*递归调用*/
 }
 x1=x; y1=y;
 color2=getpixel(x1-1,y1); /*读(x1-1,y1)点颜色值*/
 if(color2!=color1) /*判断是否与填充色相等*/
 {putpixel(x1,y1,color1); /*画点(x1,y1) */
 delay(DELAY_TIME);
 x1--;
 full(x1,y1,color1); /*递归调用*/
 }
 x1=x; y1=y;
 color2=getpixel(x1,y1+1); /*读(x1,y1+1)点颜色值*/
 if(color2!=color1) /*判断是否与填充色相等*/
 {putpixel(x1,y1,color1); /*画点(x1,y1) */
 delay(DELAY_TIME);
 y1++;
 full(x1,y1,color1); /*递归调用*/
 }
 x1=x; y1=y;
 color2=getpixel(x1,y1-1); /*读(x1,y1-1)点颜色值*/
 if(color2!=color1) /*判断是否与填充色相等*/
 {putpixel(x1,y1,color1); /*画点(x1,y1) */
 delay(DELAY_TIME);
 y1--;
 full(x1,y1,color1); /*递归调用*/
 }
 return;
}

```



## 归纳注释

程序中的常量 `DELAY_TIME` 可以控制填充速度, 单位是毫秒。也可以将 `delay(DELAY_TIME)` 语句换成 `getch()` 语句, 实现手动填充。

要注意填充面积不要过大, 否则递归层次太多, 消耗大量内存, 将导致 DOS 崩溃。图形库的路径在编译运行程序时需要在 `initgraph()` 函数中指出, 具体路径可根据用户安装 TC 的位置来决定。

# 实例 133 VGA256 色模式编程



## 实例说明

本实例程序通过 DOS 中断调用实现直接写视频缓冲区的功能, 并进行 VGA256 色模式编程。程序运行效果如图 133-1 所示。

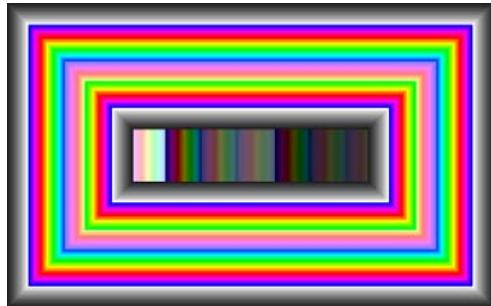


图 133-1 实例 133 程序运行效果



## 实例解析

运行该程序后, 首先由主程序通过 `InitScr()` 函数进行 `13H` 中断调用, 再由 `LineV()` 和 `Rect()` 函数子程序在屏幕上绘出如图 133-1 所示的效果图, 然后等待用户输入任意键, 系统通过调用 `RstScr()` 退出 `13H` 中断调用, 最后退出 `main()` 函数, 程序结束。



## 程序代码

### 【程序 133】 VGA256 色

// 本实例源码参见光盘



## 归纳注释

`putpoint()` 函数是通过该程序中的常指针变量 “`char far *p`” 直接取得视频缓冲区的首址, 然后直接在缓冲区内填充颜色。当然, 也可以利用 VGA BIOS 中断在屏幕上画点, 但是这种方法的速度要比直接写视频缓冲区的速度慢得多。

值得注意的是, 由于采用直接写视频缓冲区的做法, 也使得该程序的可移植性较差。这是因为不同的系统, 其缓冲区位址也各不相同。

# 实例 134 绘制蓝天图案

## 实例说明

实现随机蓝天图案的绘制。程序运行效果如图 134-1 所示。

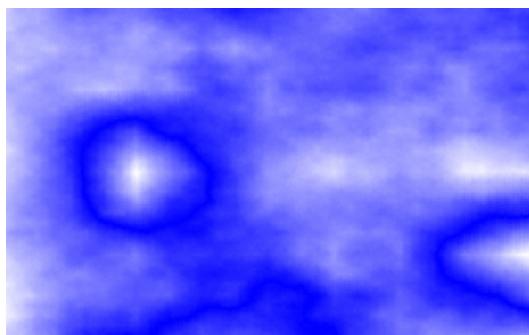


图 134-1 实例 134 程序运行效果

## 实例解析

本例使用了 BIOS 中断 INT 10H 中的 10H 号子功能调用来填充屏幕，得以生成美丽的图案。

### INT 10H 的 10H 号

INT 10H 的 10H 号功能调用是设置调色板寄存器的，多用于 VGA/EGA 系统中。调用 INT 10H 中断服务器时，AL 寄存器中的子功能号决定操作的结果。

00H 设定一个调色板寄存器，BH 中存放要设置的值，BL 中存放目的寄存器的代号。

01H 设置“过扫描”（Overscan）寄存器，BH 中存放要设置的值。

02H 设置所有的调色板寄存器和过扫描寄存器。“ES:DX”指向一个 17 字节的表，其中前 16 字节为调色板值，最后一个字节存放过扫描寄存器的值。

03H 切换“聚集/闪烁”位，BL 为 0 时，激活聚焦功能，为 1 时激活闪烁功能。

07H 读取指定调色板寄存器的值。

08H 读取过扫描寄存器的值。

09H 读取所有的调色板寄存器和过扫描寄存器的值。

10H 设定指定的颜色寄存器。

12H 设定一组颜色寄存器。

13H 选择颜色页。

15H 读取指定的颜色寄存器的值。

17H 读取一组颜色寄存器的值。

1AH 读取颜色页的状态。

## 程序代码

### 【程序 134】 美丽的随机图案

//本实例源码参见光盘



## 归纳注释

在程序执行过程中，首先用 `set_mode()` 函数指定屏幕分辨率为  $320 \times 280$  像素，然后使用 `set_pattern()` 函数设置调色板，将颜色寄存器设置成需要的值。此处就调用了 INT 10H 的 12H 号子功能，即设定一组颜色寄存器的值。接下来调用 `plot()` 函数和 `Sub_device()` 函数递归地填充屏幕区域。最后在程序结束的时候将显示模式调整回文本模式。

# 实例 135 屏幕检测程序



## 实例说明

本实例将实现使用多种显示填充模式进行屏幕检测的功能。运行该程序后，将使用各种不同的填充模式来对屏幕的中心区域进行检测，并且伴随有背景音乐。程序运行效果如图 135-1 所示。

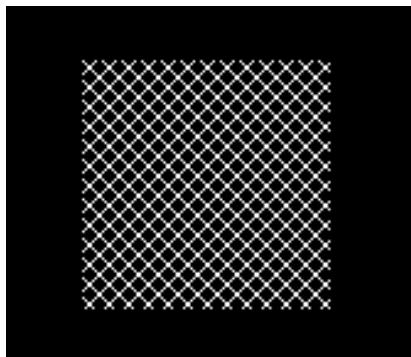


图 135-1 实例 135 程序运行效果



## 实例解析

在本例中使用了设置中断向量的方法来实现 PC Speaker 的发声。

中断向量是一个地址值，指向某一个特定的系统服务程序的开始地址（也称入口地址）。

在 DOS 环境下，系统内存的最开始的部分保存着系统的中断向量表，每个中断向量占用 4 字节的空间。例如最常用 DOS 功能服务 INT 21H 的入口地址就存放在中断向量表的 0x84 位置上。本例中使用的中断向量是 0x1C，因为这个中断向量是空闲的，所以对它的修改不会带来错误。

`getvect()` 和 `setvect()` 函数是 Turbo C 提供的对中断向量进行操作的函数。在程序中使用了 “`handler=getvect(0x1c)`” 这样的语句，就是把中断向量表中原有的内容存放在 `handler` 中。然后使用 `setvect()` 函数把 `music()` 函数的入口地址写入到中断向量 0x1C 中。在程序结束的时候，还应该把 `handler` 重新写回到中断向量 0x1C 中，以便恢复系统的默认设置。

`outportb()` 与 `inportb()` 函数是对端口进行读写的功能函数，本例中使用了 0x41，0x42 与 0x61 端口。其中 0x40~0x42 端口是系统的计时器与计数器端口，而 0x61 端口则是系统扬声器的端口。在程序的开始部分声明了一个枚举类型的 `NOTE`，表示每个音符的频率高低，`song` 则是一个数组，记录了一个曲子的音符。用户可以自行设定背景音乐，方法是修改 `song` 数组的内容。



## 程序代码

### 【程序 135】 屏幕检测

//本实例源码参见光盘



## 归纳注释

在主函数中使用 `setfillstyle()` 函数设置各种填充模式，在屏幕的正中画出一个边长 100 个像素的正方形，并进行填充，测试屏幕。

# 实例 136 运动的小车动画



## 实例说明

本实例实现运动的小车动画的绘制。程序运行效果如图 136-1 所示。

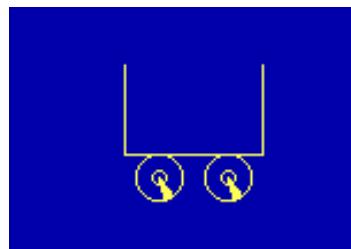


图 136-1 实例 136 程序运行效果



## 实例解析

在 Turbo C 中使用 `putimage()` 函数进行动画绘制的功能，基本思路如下。

首先分配一块内存用来存放需要显示在屏幕上的小车的图像。然后用程序更新这块内存区域，也就是改变小车运动的状态，同时把它显示到屏幕上。显示的时候使用 `putimage()` 函数。

Turbo C 中的 `graphics` 图形库提供了丰富的画图函数，通过简单的形状可以组合出丰富多彩的图案来。在本例中用到的函数如下。

`rectangle()` 函数的作用是在屏幕上指定的位置画一个矩形，声明如下：

```
void far rectangle(int left,int top,int right,int bottom);
```

其中 `left,top` 是矩形左上角的 `x, y` 坐标；`right, bottom` 是矩形右下角的 `x, y` 坐标。

`circle()` 函数的作用是在屏幕上画一个圆，声明如下：

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

其中 `x, y` 是圆心的坐标；`radius` 是圆的半径。

`pieslice()` 函数的作用是在屏幕上画一个填充的扇形，声明如下：

```
void far pieslice(int x,int y,int start,int end,int radius);
```

其中 `x, y` 是圆心坐标；`radius` 是扇形的半径；`start` 是扇形起始的角度，`end` 是扇形终止的角度。



## 程序代码

### 【程序 136】 运动的小车

//本实例源码参见光盘



## 归纳注释

本实例中的小车就是使用 rectangle()、circle() 和 pieslice() 几个函数进行组合得到的。通过矩形、圆、圆弧、椭圆等图形的组合可以得到更多复杂的图形，这也是计算机图形设计的基础。

# 实例 137 动态显示位图



## 实例说明

实现动态显示 16 色位图的功能。程序运行效果如图 137-1 所示。



图 137-1 实例 137 程序运行效果



## 实例解析

程序中首先要求用户选择要显示的 16 色位图文件，注意只能是 16 色的才能产生正确的结果。读入用户文件名后，主程序调用 bmp\_to\_dat() 函数先在磁盘上搜索到用户输入的文件，然后把位图读入内存并转存为.dat 文件。这是因为 Turbo C 提供的 putimage() 函数不能直接将位图显示在屏幕上，所以只能先把位图文件转成 putimage() 函数能够识别的数据格式。

位图文件的首部包含了位图的信息，例如文件的大小、位图的长度、宽度和颜色的数量等。首部后面放置的是整个图像每个点的颜色信息，bmp\_to\_dat() 读出其中的颜色信息，把整个位图转存成.dat 文件。

接下来主程序循环 100 次，在不同的位置上显示出这个图片，给用户一种动态的感觉。



## 程序代码

### 【程序 137】 动态显示位图

//本实例源码参见光盘



## 归纳注释

`putimage()`函数的功能是将一幅位图显示到屏幕上，其原型如下：

```
void far putimage (int left,int top,void far *bitmap,int op);
```

其中，参数 `left`, `top` 是该图片左上角的 `x`, `y` 坐标值。`Bitmap` 是一个指向存放图片的内存指针，其中前两个字分别存储图片的长和宽，余下的部分存放位图颜色信息。`op` 是显示的模式，即图片中的像素与屏幕上已有的像素之间是如何影响的。

# 实例 138 利用图形页实现动画



本实例利用图形页来实现一个较复杂的动画。程序运行效果如图 138-1 所示。

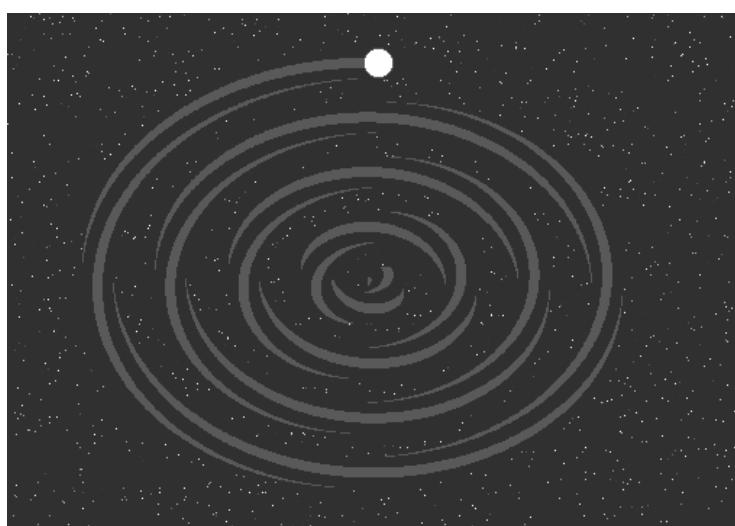


图 138-1 实例 138 程序运行效果



## 实例解析

计算机图形的动画显示实际上是一系列静止图像在不同位置的重放。大部分动态显示程序模拟运动的基本方法是相同的，即在屏幕某一显示位置上先擦除一个静止图像，然后在临近的位置上绘出下一幅图，程序重复地执行擦除和绘制的过程，就产生所需要的动画效果。这种动画方式对于简单的图形效果是很好的，但对于较为复杂的图形来说，效果就不是很好了，因为复杂图形的重画时间较长。为了解决这一问题，本例展示了如何使用多页方式显示动画。

在 Turbo C 的图形子系统中，提供了两个重要的页面设置函数，即设置图形输出活动页函数 `setactivepage()` 和设置可见图形页函数 `setvisualpage()`，其函数声明为：

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

多个图形页交替显示的过程如下：在所用的两个页面中，当一个可见页面用于显示时，另一

个关闭页同时用于绘图。当新的画面绘成后，就把两页进行互换，原来作为显示用的页面现在关闭用来绘制新的图形，而原来的绘图页面被激活作为可见页。

一般可把画面显示顺序做如下安排：第1页用于显示动画过程的1, 3, 5……画面，第2页用于显示2, 4, 6……画面，如此交替下去，利用页面转换技术进行动态显示。因为图形的擦除和重画过程都在后台进行，屏幕上出现的仅仅是整幅画面的瞬间切换，所以动态效果十分平滑。

## 程序代码

### 【程序 138】 利用图形页实现动画

//本实例源码参见光盘

## 归纳注释

图形页实际上是一个虚假页面，是内存中开辟的一块内存缓冲区。活动图像既可以是当前显示页，也可以是非显示页。当用函数 `setactivepage()` 选定某一页作为活动页后，其后所有的图形输出都针对该页。有了多个图形页，程序就可以先将图形输出到一个非显示屏幕页上，然后调用 `setvisualpage()` 改变可见页来快速显示、关闭图形页面中的画面。

## 实例 139 图形时钟

## 实例说明

本实例实现走动的图形时钟的绘制。程序运行效果如图 139-1 所示。

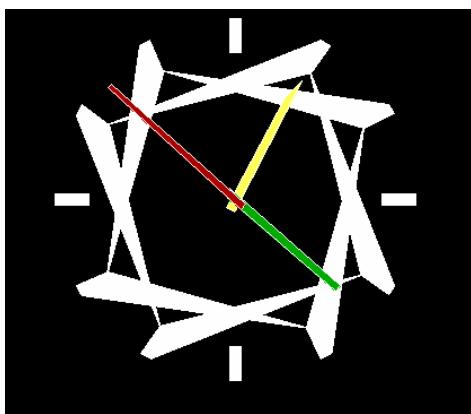


图 139-1 实例 139 程序运行效果

## 实例解析

运行该程序后，首先由 `DrawClock()` 子函数画出表盘，再由 `gettime(&newtime)` 检测系统时间，每秒钟调用 `DrawClock()` 更新一次表盘，用户按任意键退出 `main()` 函数，程序结束。`Setfillstyle()` 函数设置绘图的填充模式和颜色；`fillpoly()` 函数画一个填充的多边形。



## 程序代码

### 【程序 139】 图形时钟

```
#include<math.h>
#include<dos.h>
#include<graphics.h>
#define CENTERX 320 /*表盘中心位置*/
#define CENTERY 175
#define CLICK 100 /*喀嗒声频率*/
#define CLICKDELAY 30 /*喀嗒声延时*/
#define HEBEEP 10000 /*高声频率*/
#define LOWBEEP 500 /*低声频率*/
#define BEEPDELAY 200 /*报时声延时*/

/*表盘刻度形状*/
int Mrk_1[8]={-5,-160,5,-160,5,-130,-5,-130, };
int Mrk_2[8]={-5,-160,5,-160,2,-130,-2-130, };

/*时针形状*/
int HourHand[8]={-3,-100,3,-120,4, 10,-4,10};

/*分钟形状*/
int MiHand[8]={-3,-120,3,-120,4, 10,-4,10};

/*秒针形状*/
int SecHand[8]={-2,-150,2,-150,3, 10,-3,10};

/*发出喀嗒声*/
void Click()
{
    sound(CLICK);
    delay(CLICKDELAY);
    nosound();
}

/*高声报时*/
void HighBeep()
{
    sound(HEBEEP);
    delay(BEEPDELAY);
    nosound();
}

/*低声报时*/
void LowBeep()
{
    sound(LOWBEEP);
}

/*按任意角度画多边形*/
void DrawPoly(int *data,int angle,int color)
{
```

```

int usedata[8];
float sinang,cosang;
int i;
sinang=sin((float)angle/180*3.14);
cosang=cos((float)angle/180*3.14);
for(i=0;i<8;i+=2)
{
    usedata[i] =CENTERX+ cosang*data[i]-sinang*data[i+1]+.5;
    usedata[i+1]=CENTERY+sinang*data[i]+cosang*data[i+1]+.5;
}
setfillstyle(SOLID_FILL,color);
fillpoly(4,usedata);
}

/*画表盘*/
void DrawClock(struct time *cutime)
{
    int ang;
    float hourrate,minrate,secrate;

    setbkcolor(BLACK);
    cleardevice();
    setcolor(WHITE);

    /* 画刻度*/
    for(ang=0;ang<360;ang+=90)
    {
        DrawPoly(Mrk_1,ang,WHITE);
        DrawPoly(Mrk_2,ang+30,WHITE);
        DrawPoly(Mrk_2,ang+60,WHITE);
    }
    secrate=(float)cutime->ti_sec/60;
    minrate=((float)cutime->ti_min+secrate)/60;
    hourrate=(((float)cutime->ti_hour/12)+minrate)/12;
    ang=hourrate*360;
    DrawPoly(HourHand,ang,YELLOW); /*画时针*/
    ang=minrate*360;
    DrawPoly(MiHand,ang, GREEN); /*画分针*/
    ang=secrate*360;
    DrawPoly(SecHand,ang, RED); /*画秒针*/
}
main()
{
    int gdriver=EGA,
        gmode=EGAHII;
    int curpage;
    struct time curtime ,newtime ;
    initgraph(&gdriver,&gmode,"c:\tc");
    setbkcolor(BLUE);
    cleardevice();
    gettime(&curtime);
    curpage=0;
    DrawClock(&curtime);
    while(1)
}

```

```

{
    if(kbhit())
        break; /*按任意键退出*/
    gettime(&newtime); /*检测系统时间*/
    if(newtime.ti_sec!=curtime.ti_sec)/*每 1 秒更新一次时间*/
    {
        if(curpage==0)
            curpage=1;
        else
            curpage=0;
        curtime=newtime;
        /*设置绘图页*/
        setactivepage(curpage);
        /*在图页上画表盘*/
        DrawClock(&curtime);
        /*设置绘图页为当前可见页*/
        setvisualpage(curpage);
        /*0 分 0 秒高声报时*/
        if(newtime.ti_min==0&&newtime.ti_sec==0)
            HighBeep();
        /* 59 分 55 至 60 秒时低声报时*/
        else if(newtime.ti_min==59&&
                newtime.ti_sec<=59)
            LowBeep();/*其他时间只发出喀嗒声*/
        else
            Click();
    }
}
closegraph();
}

```



## 归纳注释

在实现发声功能时调用了系统函数 sound()。为了减少程序规模，定义了函数 DrawPoly()来画表针，DrawPoly 调用 TC 的绘图函数 setfillstyle(), fillpoly()来实现。

# 实例 140 音乐动画



## 实例说明

本实例实现音乐动画的绘制。程序运行效果如图 140-1 所示。



图 140-1 实例 140 程序运行效果



## 实例解析

(1) 动画部分的实现。动画部分主要使用了 Turbo C 实现的多页面功能。Turbo C 在 VGA 模式下可以设置多个页面。利用页面间的切换，可以体现出动画的效果。主要使用的函数有 setactivepage()，setvisualpage()，cleardevice()。Setactivepage()的功能是设置当前活动的页面，当设置以后，可以对活动页面进行各种属性的设置以及绘图，默认的活动页面是 0 号页面。本例中在两个页面上分别绘制出两对位置、大小不同的扇形和三维立方体。

(2) 音乐部分的实现。本例使用设置中断向量的方法来实现 PC Speaker 的发声。



## 程序代码

### 【程序 140】 音乐动画

// 本实例源码参见光盘



## 归纳注释

setvisualpage()函数的功能是将某一个页面设置成最前，也就是可视的页面。使用 setvisualpage() 函数在 0 号与 1 号显示页面之间切换，就出现了动画的效果。Cleardevice()函数的功能是在图形模式下清除屏幕，相对应地在文本模式下可以用 system("cls") 来清除屏幕。

# 05

## 第五部分 系统篇

### 精彩导读

- 修改环境变量
- 读取 CMOS 信息
- 获取 BIOS 设备列表
- 锁住硬盘
- 备份/恢复硬盘分区表
- 设计口令程序

# 实例 141 读取 DOS 系统中的国家信息



## 实例说明

本实例通过系统调用获取 DOS 系统中的国家信息并将其显示。程序运行效果如图 141-1 所示。

```
This program is to get the country information:  
>> Date format: yy/mm/dd  
>> Currency symbol \$  
>> Decimal separator ,  
>> Date separator - Time separator :  
>> Currency symbol precedes with no leading spaces  
>> Currency significant digits 2  
>> 24 hour time  
>> Data separator ,  
Press any key to quit....
```

图 141-1 实例 141 程序运行效果



## 实例解析

### country 函数与 country 结构

country 函数用于获取国家信息，其函数原型（在 dos.h 中定义）如下：

```
country *country (int code, struct country *info);
```

调用成功，则返回一个指向 country 型结构的指针。参数中，code 值是指定了要选择的国家代码。如果 code 参数的值为-1，country 函数将把当前国家代码设置成用户指定的代码。如果 code 参数的值不是-1，country 函数将当前国家代码设置存入缓冲区。

country 结构是在 dos.h 中定义的，如下所示：

```
struct country  
{  
    int co_date;          /*日期格式*/  
    char co_cuur[5];      /*流通货币符号*/  
    char co_thsep[2];     /*千位分隔符*/  
    char co_desep[2];     /*小数分隔符*/  
    char co_dtsep[2];     /*数据分隔符*/  
    char co_tmsep[2];     /*时间分隔符*/  
    char co_currstyle;   /*流通货币方式*/  
    char co_digits;       /*有效数字*/  
    char co_time;         /*时间*/  
    char co_case;         /*大小写*/  
    char co_dasep;        /*数据分隔符*/  
    char co_fill[10];      /*补白*/  
}
```



## 程序代码

### 【程序 141】 获取国家信息

//本实例源码参见光盘

# 实例 142 修改环境变量



## 实例说明

本实例通过系统调用实现对环境变量的读取和修改。程序运行效果如图 142-1 所示。

注意：运行本程序时，将对系统环境变量进行修改。

```
C:\TC>path  
PATH=C:\DOS;C:\TC;C:\BOOT  
  
C:\TC>142  
This program is to get the Path and change it.  
>> COMSPEC=C:\COMMAND.COM  
>> PROMPT=$p$g  
>> PATH=C:\DOS;C:\TC;C:\BOOT;c:\temp  
>> TEMP=C:\DOS  
Press any key to quit....
```

图 142-1 实例 142 程序运行效果



## 实例解析

### 环境变量的获取和修改

环境变量是包含诸如驱动器、路径或文件名之类的字符串。环境变量控制着多种程序的行为。例如，TEMP 环境变量指定程序放置临时文件的位置。最常用的系统变量还有 PATH 环境变量，PATH 指定了系统搜索的路径。例如用户输入 fdisk 命令，系统将依次在 PATH 变量指定的各个路径中寻找 fdisk，直到找到第一个名字叫做 fdisk 的可执行文件为止(可执行文件是指扩展名为 EXE, CMD, COM, VBS 等的文件)。

getenv (const char\* name) 函数的功能是获取名为 name 的环境变量，返回值是一个字符串指针。putenv (const char\* name) 函数的功能是设定指定名为 name 的环境变量。由于系统不允许在程序中直接更改环境变量，所以对环境变量的更改只能依赖于 putenv() 函数。putenv() 函数也可以删除一个环境变量，例如语句 putenv ("myEnv=") 即可删除 myEnv 环境变量。

environ 是 Turbo C 内置 (Build-in) 的全局变量，可以在任何的程序中访问这个变量。environ 是一个字符型数组，其含义是系统中所有的环境变量。可以用循环的方式，将其全部输出。



## 程序代码

### 【程序 142】 修改环境变量

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

```

#include <stdlib.h>
#include <alloc.h>
#include <string.h>
#include <dos.h>

int main(void)
{
    char *path, *ptr;
    int i = 0;
    clrscr();
    puts(" This program is to get the Path and change it.");
    /* 获得当前环境变量中的 path 信息 */
    ptr = getenv("PATH");

    /* 更新 path */
    path=(char *)malloc(strlen(ptr)+15); /*分配内存空间，比 path 的长度多 15 个字节*/
    strcpy(path,"PATH="); /*复制 path*/
    strcat(path,ptr);
    strcat(path,";c:\\temp"); /*在当前的 path 后追加一个路径*/

    /* 更新路径信息并显示所有的环境变量 */
    putenv(path); /*设置环境变量*/
    while (environ[i]) /*循环输出所有的环境变量*/
        printf(" >> %s\n",environ[i++]);
    printf(" Press any key to quit... ");
    getch();
    return 0;
}

```



### 归纳注释

程序先清屏，输出提示信息，接着调用 `getenv` 函数获取环境变量，在当前环境变量中追加 `temp` 路径，并更新路径，最后输出所有的环境变量，按任意键结束程序。

## 实例 143 显示系统文件表



### 实例说明

本实例通过系统调用实现系统文件表的显示。程序将把系统文件表的详细内容（主要是各个文件的文件句柄）显示出来。通过本例可以学习系统文件表的结构和 `intdos()`, `intdosx()` 等函数的应用。程序运行效果如图 143-1 所示。

```

5 entries in table
AUX 0 bytes 0 attribute 6 references
CON 0 bytes 0 attribute 18 references
PRN 0 bytes 0 attribute 6 references
25 entries in table
Press any key to quit...

```

图 143-1 实例 143 程序运行效果



## 实例解析

### 系统文件表

文件句柄是进入进程文件表的索引值，这些值又指向系统文件表。系统文件表存放着每个文件的信息，文件可以是 DOS、设备驱动器、驻留内存的程序，或用户程序打开的文件。如表 143-1 所示显示了 DOS 系统文件表的内容。

表 143-1

DOS 系统文件表

00H	指向下一个表的远端指针	16H	日期标记
04H	本表的入口数	18H	文件长度
06H	本入口的句柄	1CH	当前指针偏移量
08H	文件打开模式	20H	相关簇
0AH	文件属性	22H	目录入口扇区
0BH	局部/远端设备	26H	目录入口偏移量
0DH	驱动程序头文件或 DPB	27H	可执行文件名
12H	起始簇	34H	保留
14H	时间标记		

实际上 DOS 把系统文件表分成两段。第一段包含 5 个入口，第二段为用户在 config.sys 文件中 FILES 参数指定的入口数提供足够的空间。

程序首先根据表 143-1 所示的系统文件表内容定义了相对应的结构型 SystemTableEntry：

```
struct SystemTableEntry
{
    struct SystemTableEntry far *next; /*下一个系统文件表入口*/
    unsigned file_count; /*表中文件的个数*/
    unsigned handle_count; /*对此文件所进行操作的个数*/
    unsigned open_mode; /*文件的打开模式*/
    char file_attribute; /*文件的属性字节*/
    unsigned local_remote; /*本地/远端设备*/
    unsigned far *DPD; /*驱动器参数块*/
    unsigned starting_cluster; /*起始簇*/
    unsigned time_stamp; /*时间戳*/
    unsigned date_stamp; /*数据戳*/
    long file_size; /*文件大小*/
    long current_offset; /*当前偏移*/
    unsigned relative_cluster; /*相对簇*/
    long directory_sector_number; /*路径扇区数*/
    char directory_entry_offset; /*路径入口偏移*/
    char filename_ext[11]; /*可执行文件名*/
};
```

然后获取本机 DOS 版本号，这是通过 intdos() 函数执行 0x21 中断调用 ax=0x3001 函数得到的。由此版本号，通过调用 int 21H 的 0x52 中断功能获取内部 DOS 列表地址，该地址偏移 4 位的地址即是系统文件表的地址，由此便可以导出系统文件表的内容。接下来就是遍历此系统文件表，依次访问文件表中记录的每个文件，如果有对该文件的操作，那么在屏幕上输出相对应的可执行文件名，然后指向下一个文件，进行同样的操作，直到遍历完整个文件表。



## 程序代码

### 【程序 143】 显示系统文件表

//本实例源码参见光盘



## 归纳注释

intdos 函数和 intdosx 函数的原型为：

```
int intdos(union REGS *inregs, union REGS *outregs);
int intdosx(union REGS *inregs, union REGS *outregs, struct SREGS *segregs);
```

这两个函数执行 DOS 的 int 21H 中断，调用一个指定的 DOS 函数。Inregs->h.ah 的值指定了要调用的 DOS 函数。intdosx 在调用前，还要对 segregs->ds 和 segregs->es 进行赋值。

程序中用到的宏 FP\_OFF, FP\_SEG 和 MK\_FP 声明如下：

```
unsigned FP_OFF(void far *fp);
unsigned FP_SEG(boid far *p);
void far *MK_FP(unsigned seg, unsigned ofs);
```

FP\_OFF 作用于远指针\*p，得到一个无符号的整数值，即远地址偏移；FP\_SEG 作用于远指针\*p，得到一个无符号整数值，即远地址段的值；MK\_FP 由段号和偏移生成一个远指针。

## 实例 144 显示目录内容



## 实例说明

本实例通过系统调用显示一个目录下所有文件和文件夹，目的是让读者学会使用 opendir() 函数和 readdir() 函数打开并读取一个文件目录，利用链表进行排序等内容。程序运行效果如图 144-1 所示（在 Borland C++ 3.1 中编译运行）。

The screenshot shows a terminal window titled "Borland C++ for DOS". The window displays a list of files in the current directory, including various executable files (EXE) and DLL files. The files listed are: UNZIP.EXE, VESATEST.EXE, WIN31MWH.HLP, WINHELP.EXE, WINHELP.TCH, WINSIGHT.EXE, WINSIGHT.HLP, WINSPECTR.EXE, WINSPECTR.HLP, WINSTUB.EXE, WORKED1.DLL, WORKED2.DLL, WORKED3.DLL, WORKED4.DLL, WORKED5.DLL, WORKHELP.HLP, WORKLIB1.DLL, WORKLIB2.DLL, WORKOPT.DOS, WORKSHOP.EXE, WREMOTE.EXE, WRSETUP.EXE, WSIHOOK.DLL, WSIWIN.DLL. At the bottom of the list, it says "Press any key to quit...".

图 144-1 实例 144 程序运行效果



## 实例解析

程序在声明部分，首先定义了一个DIR型的指针 directory\_pointer 和 dirent 结构型的指针 entry，把这两个指针作为对象目录的入口指针，然后定义了本程序进行排序的链表结构。在程序中用 opendir 函数打开以命令行参数为名称的文件夹，返回目录指针到 directory\_pointer，然后读取该目录下文件名到 entry 中，按照名称字典将其插入到链表中。最后打印整个链表内容，也就完成了显示该目录下排序好的所有文件及子目录的名称。



## 程序代码

### 【程序 144】 显示目录内容

```
#include <stdio.h>
#include <dirent.h>
#include <alloc.h>
#include <string.h>
#include <stdlib.h>

void main(int argc, char *argv[])
{
    DIR *directory_pointer;
    struct dirent *entry;
    struct fileList
    {
        char filename[64];
        struct fileList *next;
    } start, *node, *previous, *new;

    if ((directory_pointer = opendir(argv[1]))==NULL)
        /*取 argv[1]文件夹的指针赋于*/
        printf("Error opening %s\n", argv[1]);
    else
    {
        start.next = NULL;
        /*将 directory_pointer 指向的文件名列表做成一个以 fileList 类型为结点的链*/
        while(entry=readdir(directory_pointer))
        /*读取 directory_pointer 指向的文件名*/
        {
            // Find the correct location
            previous = &start;
            node = start.next;

            while ((node) && (strcmp(entry, node->filename) > 0))
            /* 以字典顺序搜索在链表中此文件名应该插入的位置*/
            {
                node = node->next;
                previous = previous->next;
            }

            new = (struct fileList *)

```

```

    malloc(sizeof(struct FileList));
    if (new == NULL) /*内存分配失败*/
    {
        printf("Insufficient memory to store list\n");
        exit(1);
    }
    /*完成插入*/
    new->next = node;
    previous->next = new;
    strcpy(new->filename, entry);
}

closedir(directory_pointer);
node = start.next;
/*输出整个链表结点的文件名*/
while (node)
{
    printf("%s\n", node->filename);
    node = node->next;
}
printf(" Press any key to quit...");
getch();
return;
}

```



## 归纳注释

程序主要是通过链表完成查询和删除、排序等操作。链表是 C 语言中常用的一种数据结构，通常有单向、双向、单向循环、双向循环链表等类型。相对于数组，它更容易完成数据的插入和删除，也就更容易完成数据的排序和查找，并且会更经济地使用内存空间，不会造成空间闲置。

本例定义的文件链表类型为：

```
struct FileList { char filename[64]; struct FileList *next; }.
```

为了完成文件名按字典排序输出，应该在创建链表时把所选路径中的所有文件名有序地插入到链表中。

# 实例 145 读取磁盘文件



## 实例说明

本实例通过系统调用递归读取磁盘文件，并滚屏显示整个磁盘中的所有文件名，以便使读者掌握使用磁盘文件目录操作库 dirent.h 中相关函数，进行磁盘文件目录的访问和操作。程序运行效果如图 145-1 所示（在 Borland C++ 3.1 中编译运行）。



c:\ Borland C++ for DOS

BACK\_B.GIF  
BIAOSHI.GIF  
BUYDUBA.GIF  
DATAFIX.GIF  
DOWNLO^1.GIF  
FORUM.GIF  
IMGRIGHT.GIF  
IMGTOP.JPG  
MAIL.GIF  
PRODUCT.GIF  
KAI\_ESCAN.OCX  
DESKTOP.INI  
KAUSUC.EXE  
UPDLIB.DLL  
KAUPATCH.DLL  
UPDATE.ORD  
KAU00279.DAT  
UPDPATCH.DLL  
ORDER.DAT  
KAEPLAT.DLL  
KAEWEB.DAT  
KAEADAILY.DAT  
KAECORE.DAT  
THUMBS.DB

Press any key to quit...

图 145-1 实例 145 程序运行效果



## 实例解析

dirent.h 是 Turbo C 中主要针对磁盘文件进行操作的库函数头文件，包括 opendir、closedir、readdir 等常用的目录读取或操作函数，这些函数说明如下。

```
DIR *opendir(char *dirname);
```

opendir 函数用于打开一个目录串以便读取。当要从一个目录串读取连续的目录入口时，使用 readdir 函数。

```
struct dirent readdir(DIR *dirp);
```

该函数从目录串\*dirp 中读取当前目录入口，然后将入口指针移向目录串下一个目录入口。它返回一个指向 dirent 结构的指针。

```
void closedir(DIR *dirp);
```

该函数用于关闭一个目录链表。

此外，本例程序还用到了基本输入输出库 io.h 中的函数 chmod，其具体声明如下：

```
int _chmod(const char *path, int func [, int attrib]);
```

此函数获得或设置\*path 所指向文件的 DOS 文件属性，当 func=0 时，该函数返回当前 DOS 属性；当 func=1 时，函数设置 attrib 的属性。属性 FA\_DIREC 表明指向的是一个文件夹。



## 程序代码

### 【程序 145】 读取磁盘文件

```
//本实例源码参见光盘
```



## 归纳注释

程序首先由 opendir 函数得到当前目录的一个目录串指针，然后沿此指针向下搜索，判断此指针中的目录入口文件的属性，如果入口文件为文件夹，就递归调用主函数继续搜索；如果入口文件是文件，则打印文件名。

# 实例 146 删 除 目 录 树

## 实例说明

本实例使用系统调用来删除目录树，即将用户指定的整个目录连同其文件和子目录全部删除。程序运行效果如图 146-1 所示（在 Borland C++ 3.1 中编译运行）。

```
E:\BORLANDC\BIN\test>dir
Volume in drive E is FTP
Volume Serial Number is 44F6-E592

Directory of E:\BORLANDC\BIN\test

2004-04-28  01:11    <DIR>          .
2004-04-28  01:11    <DIR>          ..
2004-04-28  01:10           15,555 146.EXE
2004-04-28  01:17    <DIR>          mydir
          1 File(s)           15,555 bytes
          3 Dir(s)        181,964,800 bytes free

E:\BORLANDC\BIN\test>dir mydir
Volume in drive E is FTP
Volume Serial Number is 44F6-E592

Directory of E:\BORLANDC\BIN\test\mydir

2004-04-28  01:17    <DIR>          .
2004-04-28  01:17    <DIR>          ..
2004-04-18  19:31           2,835 i33.C
2004-04-28  00:20           1,723 144.C
2004-04-28  00:49           1,256 145.C
2004-04-28  01:10           2,108 146.C
          4 File(s)           7,922 bytes
          2 Dir(s)        181,964,800 bytes free

E:\BORLANDC\BIN\test>146 mydir
>> Delete Successful!
Press any key to quit...
E:\BORLANDC\BIN\test>dir
Volume in drive E is FTP
Volume Serial Number is 44F6-E592

Directory of E:\BORLANDC\BIN\test

2004-04-28  01:11    <DIR>          .
2004-04-28  01:11    <DIR>          ..
2004-04-28  01:10           15,555 146.EXE
          1 File(s)           15,555 bytes
          2 Dir(s)        181,985,280 bytes free

E:\BORLANDC\BIN\test>_
```

图 146-1 实例 146 程序运行效果

## 实例解析

在 MS-DOS 6.0 及以上版本中，Microsoft 引进了 Deltree 命令，允许用户直接删除目录和它的文件以及该目录的子目录。本实例实现的正是 Deltree 命令的功能。

程序首先分析输入的目录，用 fnsplit() 函数进行分解。函数 fnsplit() 的定义在 dir.h 头文件中，具体声明为：

```
int fnsplit (const char *path, char *drive, char *dir, char* name, char *ext);
```

它将一个文件的全路径名作为一个字符串进行分解，分为 4 个部分：盘符、目录、文件名和扩展名。

另外，程序中还用 getcwd 函数：

```
char *getcwd(char *buf, int buflen);
```

以获得当前工作目录的全路径名，并把它存放在 buffer 中。用 intdosx 指向 DOS int 21H 的中断功能。



## 程序代码

### 【程序 146】 删除目录树

//本实例源码参见光盘



## 归纳注释

程序在判断该目录是否存在以及是否为当前工作目录后，再进入目录中进行递归删除，直到这个目录的所有文件和子目录都被删除，然后退到上一层目录，删除此目录。由此完成对整个目录树的删除。

# 实例 147 定义文本模式



## 实例说明

本实例通过系统调用自定义文本模式，通过本实例读者可以掌握有关 BIOS INT 10H 的功能和 window(), geninterrupt(), textmode() 等函数的使用方法。程序运行效果如图 147-1 所示。



图 147-1 实例 147 程序运行效果



## 实例解析

### BIOS 中断

程序主要使用了 BIOS 中提供的 INT 10H 中断的各种功能。BIOS 的 INT 10H 中断主要提供了视频操作的各种功能，是系统提供的一组功能强大的函数。这些函数可以用以下语句来调用：

```
_AH=0xXX;  
geninterrupt(VIDEO_BIOS);
```

其中，\_AH 表示了调用的 INT 10H 中断的具体功能函数号 XX，geninterrupt() 函数执行一个

软中断，在上面的语句中，VIDEO\_BIOS 表示 10H 中断。

在本例中，主要使用其中的文本模式功能。

0FH——获得当前的视频模式，返回值 AH 中存放每行的字符数，AL 中存放视频模式。

11H——为 VGA、EGA 模式生成模式集合，并重置视频环境，生成可显示的点阵字符。

12H——为 VGA、EGA 模式选择显示器程序，执行后返回值。

BH: 0——彩色模式，1——单色模式。

BL: 0——64K, 1——128K, 2——192K, 3——256K。

CH: 适配器位。

CL: 开关设置。

1AH——获取当前正在使用的显示模式鉴别码。

另外，函数 setfont8x8() 是将显示模式设置成 8×8 像素的点阵格式，其参数位显示模式，即每行可以显示的字符数。

## 程序代码

### 【程序 147】 定义文本模式

```
#include <conio.h>
#include <dos.h>
#include <stdlib.h>
#define VIDEO_BIOS      0x10 /*int 10h 是 BIOS 中对视频函数的调用*/
int setfont8x8(int); /*设置不同的显示模式*/
void    setstdfont(int); /*恢复成系统默认的显示模式*/
void    main(void)
{
    intlines,i;
    lines = setfont8x8(C80);
    /*设置 8×8 点阵，每行 80 字符的显示模式，并获取可显示的最大行数*/
    textattr(WHITE); /*textattr()函数设置字符模式下窗口的前景色和背景色*/
    clrscr(); /*清除屏幕*/
    if (lines < 43) {
        textattr(LIGHTRED);
        cprintf("\n\r Drivers of EGA or VGA not found... \n\r");
        /*cprintf() 的功能是向窗口输出文本*/
        exit(1);
    }
    window(20,15,70,35); /*画字符模式窗口，4 个参数依次为左，上，右，下的位置*/
    textattr((RED<<4)+WHITE); /*把窗口设置成前景色为白色，背景色为红色*/
    clrscr();
    for (i=1;i<=lines;i++) { /*循环输出最多能输出的行数*/
        cprintf("\n\r No. %d ",i);
        delay(200); /*每输出一行，等待 200ms*/
    }
    getch(); /*等待用户输入一个字符*/
    window(1,1,80,lines); /*重新设置窗口*/
    textattr(LIGHTGRAY<<4); /*将窗口背景色设置为灰色*/
    clrscr();
    cprintf("\n\r Full screen 80x%d display mode.\n\r",lines);
    getch();
    lines = setfont8x8(C40); /*将窗口设置为每行 40 个字符的显示模式*/
    textattr((BLUE<<4)+LIGHTGREEN); /*设置窗口，前景为绿色，背景为蓝色*/
```

```

clrscr();
cprintf("\n\r Can be also set as 40x%d mode.\n\r",lines);
getch();
setstdfont(C80); /*重新设置成标准的每行 80 字符的显示模式*/
clrscr();
cprintf("\n\r Back to normal mode... \n\r");
printf(" Press any key to quit... ");
getch();
exit(0);
}

int setfont8x8(mode)
int mode;
{
int maxlines,maxcol;
char vtype,displaytype;
textmode(mode); /*设置文本格式, mode 含义为每行可以显示的字符数*/
_AH = 0x0F; /*int 10h 的 0fh 功能为获取当前的显示模式, 执行后每行可显示字符数保存在 ah 中*/
geninterrupt(VIDEO_BIOS); /*geninterrupt() 函数执行一个软中断, 调用 int 10h*/
maxcol = _AH; /*获取每行可以显示的字符数*/
_AX = 0x1A00; /*int 10h 的 1ah 功能为获取当前的显示代码*/
geninterrupt(VIDEO_BIOS);
displaytype = _AL; /*int 10h 返回后, al 中为显示类型*/
vtype = _BL; /*bl 中为显示器的类型*/
if (displaytype == 0x1A) { /*可以直接获取最大行数*/
    switch (vtype) {
        case 4:
        case 5: maxlines = 43;
                  break;
        case 7:
        case 8:
        case 11:
        case 12: maxlines = 50;
                  break;
        default: maxlines = 25;
                  break;
    }
}
else { /*无法读取显示器的类型 */
    _AH = 0x12; /*int 10h 的 12h 功能为选择显示器程序*/
    _BL = 0x10;
    geninterrupt(VIDEO_BIOS);
    if (_BL == 0x10)
        maxlines = 25;
    else
        maxlines = 43;
}
if (maxlines > 25) { /*如果可以设置更多的行*/
_AX = 0x1112; /*以下的部分都是 int 10h 的 11h 号功能调用, 作用是生成相应的显示字符*/
_BL = 0;
geninterrupt(VIDEO_BIOS);
_AX = 0x1103;
_BL = 0;
geninterrupt(VIDEO_BIOS);
}

```

```

*((char *) &directvideo - 8) = maxlines; /*设置显示行数*/
window(1,1,maxcol,maxlines); /*画出相应大小的窗口*/
return(maxlines); /*返回可以设置的最大行数*/
}
void setstdfont(mode)
int mode;
{
if (mode != LASTMODE)
    _AL = mode;
else {
    _AH = 0x0F; /*获取当前显示模式*/
    geninterrupt(VIDEO_BIOS);
    mode = _AL;
}
_AH = 0; /*恢复成系统标准模式*/
geninterrupt(VIDEO_BIOS);
*((char *) &directvideo - 8) = 25; /*行数设置成 25 行*/
textmode(mode);
}

```



## 归纳注释

程序先把显示模式设置为每行 80 个字符，并将屏幕设置成白色，然后清屏，画一个窗口，并将其前景色设为白色，背景色设为红色，在窗口中输出能够显示的最大行数，按任意键后，把显示模式改为每行 80 个字符，刷新屏幕，再改成每行 40 个字符的显示模式，最后还原成系统最初的标准设置。

# 实例 148 设计立体窗口



## 实例说明

本实例通过系统调用设计立体投影窗口，即在不同的位置上显示 3 个不同颜色的立体投影窗口，以便读者掌握 window() 函数的使用方法。程序运行效果如图 148-1 所示。

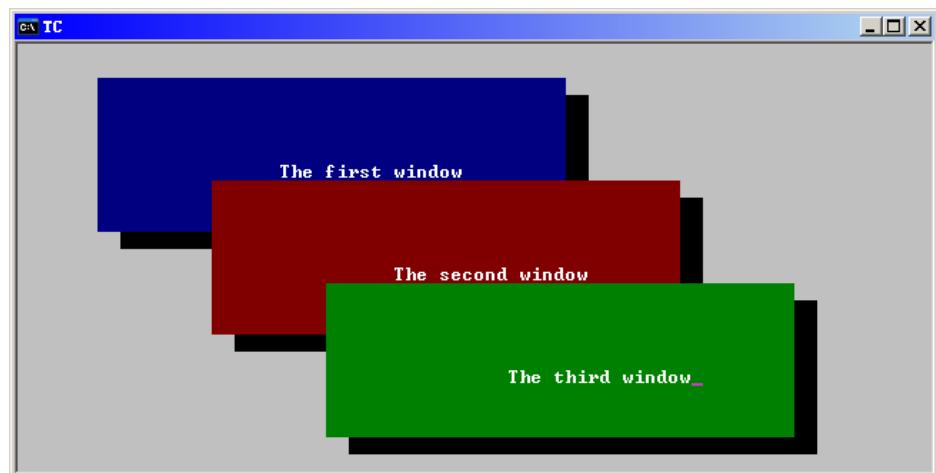


图 148-1 实例 148 程序运行效果



## 实例解析

### 立体投影窗口

立体投影窗口的原理是在设定的窗口区域内以投影色（一般为黑色）进行涂色，然后在原窗口位置上错位后，再用一种不同的颜色画出一个窗口，将这个新画的窗口叠加在原投影窗口上，就形成了立体投影窗口的效果。

Turbo C 提供了窗口函数 window(int left, int top, int right, int bottom)，允许在屏幕范围内画出任意大小的矩形窗口。其中 left、top 是窗口左上角的坐标，right、bottom 是窗口右下角的坐标。该函数在屏幕上定义一个文本窗口，并且使屏幕上这个区域成为激活窗口。

函数 cprintf()，cputs()，clschr()是窗口 I/O 函数，它们自动激活定义的窗口，并在该窗口中完成输出、清屏等操作。但是因为 Turbo C 默认显示方式是直接传送字符串到屏幕显示缓冲区，因此必须设置全局变量 directvideo=0，以通过 BIOS 调用使用这些窗口 I/O 函数。



### 程序代码

#### 【程序 148】 设计立体窗口

```
#include <conio.h>
void window_3d( int, int, int, int, int, int );
int main(void)
{
    directvideo = 0;
    textmode(3);
    textbackground( WHITE );
    textcolor( BLACK );
    clrscr();
    window_3d( 10,4,50,12, BLUE, WHITE );
    gotoxy( 17,6 );
    cputs("The first window");
    window_3d(20,10,60,18,RED, WHITE );
    gotoxy(17,6);
    cputs("The second window");
    window_3d(30,16,70,24, GREEN, WHITE );
    gotoxy(17,6);
    cputs("The third window");
    getch();
    return 0;
}
void window_3d( int x1, int y1, int x2, int y2, int bk_color, int fo_color )
/*立体投影窗口的显示
   x1,y1, x2, y2 是窗口的大小
   bk_color 是背景色
   fo_color 是前景色，即文本的颜色*/
{
    textbackground(BLACK);
    window(x1, y1,x2, y2); /*画背景窗口*/
    clrscr();
    textbackground(bk_color); /*设置背景色*/
```

```
textcolor(fo_color); /*设置前景色*/  
window(x1-2, y1-1, x2-2, y2-1); /*画实际的窗口*/  
clrscr();  
}
```



## 归纳注释

显示3个立体投影窗口的操作主要由函数 `window_3d(int x1, int y1, int x2, int y2, int bk_color, int fo_color)` 来完成，其参数 `x1, y1, x2, y2` 分别是投影窗口的左上角和右下角坐标，`bk_color`, `fo_color` 是投影窗口的背景色和前景色（即窗口中显示字符的颜色）。该函数先设置背景色为黑色，然后根据坐标参数开一个窗口（此时屏幕上无窗口显示），并调用清屏函数 `clsscr()`，使得屏幕上显示一个黑色窗口，接着设置立体投影窗口的背景色和前景色，再把黑色投影窗口的 `x` 轴和 `y` 轴坐标分别减 2 和减 1，得到的坐标值作为新的窗口坐标打开一个窗口，即将此窗口叠加到黑色窗口上，再清屏，就得到一个立体投影窗口。

# 实例 149 彩色弹出菜单



## 实例说明

本实例通过系统调用实现彩色弹出式菜单的设计。程序运行效果如图 149-1 所示。



图 149-1 实例 149 程序运行效果



## 实例解析

### 弹出式菜单设计

弹出式菜单被广泛应用于各类软件或操作系统中，作为显示某些功能的工具。弹出式菜单被设计成“弹出”并显示的一个窗口，功能选项在窗口中垂直排列，并且第一个选项为高亮反显。用户可以用快捷键或单击方式进行菜单功能选择。

设计弹出式菜单有以下几个要点。

(1) 保存菜单出现前的部分屏幕的函数 `save_video`。要保存屏幕内容，必须读取并存储屏幕上各位置的当前值。用 INT 10H 的 08 号子功能从屏幕上某一个位置读一个字符，返回该字符的 ASCII 字符码及其在当前光标处的属性。其调用参数为“AH=0x08，BH=显示页”，返回时“AH=属性字节，AL=ASCII 字符码”。

(2) 显示菜单边框的函数 `chineseputs`。由于本例开发的彩色汉字弹出式菜单函数基于中文文本操作系统，所以菜单的边框是中文方式下的双字节汉字制表符。构成菜单边框的制表符有 6 种，系统把这些制表符作为汉字对待，因而需先构造一个显示汉字字符串的函数 `chineseputs()`，调用 BIOS INT 10H 中断的 09 号功能显示汉字字符串，该调用的入口参数为 AH=09H，AL=ASCII 字符码，BH=显示页，BL=属性（字符方式下）或颜色（图形方式下），CX=写字符计数。

(3) 显示菜单正文的函数 `display_menu`，为显示单个字符串，只要把指针像数组一样编程索引，数组的每一项，均为指向相应菜单项的指针。

(4) 等待用户响应的函数 `get_resp`。

(5) 处理用户响应的主函数 `main`。

(6) 恢复屏幕为原始状态的函数 `restore_video`。

(7) `popup` 函数，用于弹出所选中的菜单项。

## 程序代码

### 【程序 149】 彩色弹出菜单

//本实例源码参见光盘

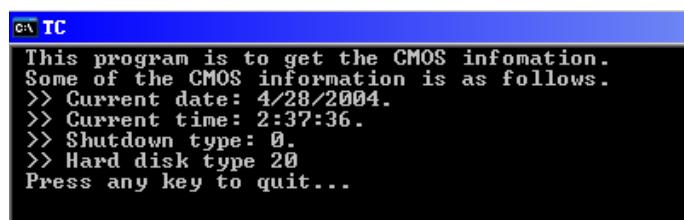
## 归纳注释

本实例中，`popup` 函数是弹出式菜单程序中最主要的函数，它负责调用其他函数，`popup` 首先判断菜单起始点坐标是否超出屏幕范围，然后根据菜单正文最长项来决定菜单终止点的坐标。基于菜单起始点和终止点的坐标，`popup` 函数计算出保存将要被弹出式菜单覆盖的部分屏幕所需的内存，然后把屏幕内容保存到该内存区域中。如果边框标志置位，则显示边框；否则不显示边框。接着显示菜单正文项，并准备接受用户的选择。一旦用户选择了某选项，则 `popup` 函数恢复屏幕，并释放保存屏幕所需的那块内存区域，返回用户的选项值给调用 `popup` 函数的主函数 `main()`。

## 实例 150 读取 CMOS 信息

## 实例说明

本实例通过系统调用来读取 CMOS 的信息。程序运行效果如图 150-1 所示。



```
CM TC
This program is to get the CMOS infomation.
Some of the CMOS information is as follows.
>> Current date: 4/28/2004.
>> Current time: 2:37:36.
>> Shutdown type: 0.
>> Hard disk type 20
Press any key to quit...
```

图 150-1 实例 150 程序运行效果



## 实例解析

### CMOS 信息与 CMOS 结构

PC 把系统配置信息存放在 CMOS 存储器中，包括用户驱动器类型、系统日期等。PC 机不使用标准的段加偏移地址方式访问 CMOS，而是通过 PC 端口地址与 CMOS 联系。

本实例程序通过调用 `inportb` 和 `outportb` 函数获取 CMOS 信息，并显示有关内容。程序先定义了 CMOS 结构（详见程序代码，各项含义可通过其名称得知），接着声明了一个此结构的 CMOS 变量，然后通过 `outportb` 和 `inportb` 函数对 CMOS 内容进行以 byte 为单位的访问，依次读取 CMOS 信息，并显示其中的部分内容。



### 程序代码

#### 【程序 150】 读取 CMOS 信息

//本实例源码参见光盘



### 归纳注释

程序用到的 `inportb` 和 `outportb` 函数在头文件 dos.h 中定义如下：

```
unsigned char inportb(int portid);
```

实现从硬件端口读取一个字节。

```
void outportb(int portid, unsigned char value);
```

实现向硬件端口输出一个字节的 value 值。

在本实例中，`70H` 和 `71H` 分别是针对 CMOS 进行写入和读取的端口，以便返回系统的 CMOS 信息，将其赋予 CMOS 结构变量中。

## 实例 151 获取 BIOS 设备列表



### 实例说明

本实例通过系统调用来获取 BIOS 设备列表。程序运行效果如图 151-1 所示（在 Borland C++ 3.1 中编译运行）。

```
Borland C++ for DOS
This program is to get BIOS equipments list.
>> Math coprocessor available.
>> No game adapters.
>> System board memory: 16.
>> Video card memory: 48.
>> Number of floppies: 1.
>> Number of printers: 3.
>> Number of serial ports: 4.
Press any key to quit...
```

图 151-1 实例 151 程序运行效果



## 实例解析

### \_bios\_equiplist 函数

本实例程序通过调用 bios.h 头文件中定义的 \_bios\_equiplist 函数来获取系统的 BIOS 设备列表并输出。此函数的原型为：

```
unsigned _bios_equiplist(void);
```

此函数通过使用 BIOS INT11H 中断来返回一个整型数据，描述连接在系统上外围设备的信息。此整型数据的具体位（Bit）含义如下。

Bit 15 14——并行打印机数目，00=0, 01=1, 10=2, 11=3。

Bit 13——附加的打印机序列号。

Bit 12——附加的游戏输入输出设备。

Bit 11 10 9——COM 口的个数，000=0, 001=1, …, 111=7。

Bit 8——系统是否可以 DMA，0——可以，1——不可以。

Bit 7 6——磁盘驱动器数目，00=0, 01=1, 10=2, 11=3。

Bit 5 4——初始化视频格式，00=未使用，01=40×25BW 彩色，10=80×25BW 彩色，11=80×25BW 黑白。

Bit 3 2——主板 RAM 大小，00=16K, 01=32K, 10=48K, 11=64K。

Bit 1——是否具有浮点运行协处理器。

Bit 0——是否从磁盘启动。

程序中定义了与此相对应的位域型作为列表信息的存储类型（具体声明见程序中 struct Equip 的定义），然后定义了 Equipment 的联合类型，并声明了变量 equip 以存储 BIOS 设备列表。



### 程序代码

#### 【程序 151】 BIOS 设备列表

```
//本实例源码参见光盘
```



### 归纳注释

程序中使用了“联合”类型（union）来实现对设备列表信息的位访问。在“union Equipment { unsigned list; struct Equip list\_bits; }”中，list 和 list\_bits 存放的地址是相同的，只是前者只能作为无符号整型变量访问，后者则可以访问其中的某些位，这样方便于操作。

## 实例 152 锁住硬盘



### 实例说明

本实例通过系统调用实现对硬盘加软锁。程序运行效果如图 152-1 所示。

注意：因为对硬盘分区表的操作关系到系统的安全问题，比较危险，本实例程序仅作为了解和学习使用，上机实践时应有专业人员指导。

```

This is a hard disk lock program.
Do you want to lock your hard disk? (Y/N): y
Writing main boot sector successfully!
Press any key to quit...

```

图 152-1 实例 152 程序运行效果



## 实例解析

### 硬盘主引导扇区

给硬盘加锁的方法很多，最简单的方法就是改变主引导记录或者改变分区表中某些关键字，以达到不能使用硬盘的目的。硬盘主引导扇区内容如表 152-1 所示。

表 152-1 硬盘主引导扇区内容

000H	主引导记录 (240 字节)	1CEH	第二分区表 (16 字节)
0F0H	全 0 (206 字节)	1DEH	第三分区表 (16 字节)
1BEH	第一分区表 (16 字节)	1EEH	第四分区表 (16 字节)
			55H AAH

主引导扇区最后两个字节 55H 和 AAH 是硬盘自举记录的有效标志。每个分区表为 16 个字节，具体内容见表 152-2。其中，Boot ind 是自举标志字节，其值为 80H 时，表示可自举分区，其值为 00H 时，表示不可自举分区。SYS ind 是 DOS 系统标志字节，其值为 01H 时，表示分区是 DOS 分区，为 00H 时，表示未知。H.S.CYL 表示分区的起始地址：H 是磁头号，S 是扇区号，CYL 是柱面号的低 8 位，高 2 位在 S 字节的高 2 位。REL seet 表示该分区的相对扇区号，#of sects 表示该分区使用的扇区数。例如，某硬盘的一个分区表为 80 00 02 00 01 03 51 30 01 00 00 00 03 51 00 00。第一字节 80H 是自举分区标志，第五字节 01H 是 DOS 分区标志，若改变 01H 为 00H，可达到加密硬盘的目的。

表 152-2 分区表字节内容

Boot ind	H	S	CYL
SYS ind	H	S	CYL
REL seet			
# of sects			

程序仅仅修改主引导扇区最后两个字节 55H 和 AAH，修改后，如果用硬盘启动，则系统提示：Disk Boot Failure, Insert System Disk AND Press Enter。



## 程序代码

### 【程序 152】 锁住硬盘

```

#include<bios.h>
#include<stdio.h>
#include<conio.h>
int main(void)
{
    int result;
    char a='N';
    char buffer[512];
    clrscr();

```

```

printf(" This is a hard disk lock program.\n");
printf(" Do you want to lock your hard disk? (Y/N): ");
scanf("%c",&a);
if (a == 'Y'||a=='y')
{
result = biosdisk(2,0x80,0,0,1,1,buffer);
if(result)
{
    buffer[510] = 0x0;
    buffer[511] = 0x0;
    printf(" Fail to read main boot sector!\n");
}
if(!result) result = biosdisk(3,0x80,0,0,1,1,buffer);
(!result)?(printf(" Writing main boot sector successfully!\n")):(printf(" Fail
to write main boot sector!\n"));
}
printf(" Press any key to quit...");getch();
return 0;
}

```



## 归纳注释

程序中用到的主要函数是 biosdisk 函数，其定义为“char biosdisk(int cmd, int drive, int head, int track, int sector, int nsects, void \*buffer)”。该函数使用中断 13H，把磁盘操作直接转给 BIOS。cmd 参数指示待执行的操作，cmd=2 时是读盘操作，cmd=3 时是写盘操作。drive 为 0 时代表第一个软驱，为 1 时代表第二个软驱，为 0x80 则表示第一个硬盘驱动器。函数返回值为 00 时，表示操作成功。

# 实例 153 备份/恢复硬盘分区表



## 实例说明

本实例通过系统调用实现对硬盘分区表的备份和恢复。程序运行效果如图 153-1 所示（在 Borland C++ 3.1 中编译运行）。

注意：由于程序是对硬盘的分区表进行操作，对系统来说比较危险，容易产生丢失文件甚至系统崩溃的情况，所以必须谨慎再谨慎。

```

C:\TC>153
The correction method using this program is : program S or program R
S---save partition table to file part.doc in c disk
R---restore partition table from file part.doc in c disk

C:\TC>153 s
Read partition table successfully!
Partition table save successfully!

C:\TC>_

```

图 153-1 实例 153 程序运行效果



## 实例解析

为了实现对硬盘分区表的保存与恢复，本例采用了 biosdisk() 函数对硬盘的分区表进行了读取和写入。一般来说，硬盘的分区表都保存在硬盘 0 柱面 0 磁道 0 扇区 0 簇，biosdisk 函数通过 INT 13H 中断直接向 BIOS 发出磁盘操作命令，从指定的该盘区位置读取 512 字节数据并保存在 buffer 中。然后使用文件操作函数 fopen, fwrite 等进行保存。恢复分区表则相反，调用 biosdisk 函数将以前保存过的分区表文件的内容恢复到分区表即可。



## 程序代码

### 【程序 153】 备份/恢复硬盘分区表

//本实例源码参见光盘



## 归纳注释

本实例程序中用到的文件读写操作函数 fread 和 fwrite 定义如下：

```
fread (void *buffer, int size, int count, FILE *fp);  
fwrite (const void *buffer, int size, int count, FILE *fp);
```

其中，buffer 是一个指针，用于指向读/写数据的地址，size 是要读写的字节数，count 是要进行读写 size 字节的数据项，fp 为指向文件的文件指针。

# 实例 154 设计口令程序



## 实例说明

本实例通过系统调用实现口令程序设计。程序运行效果如图 154-1 所示。

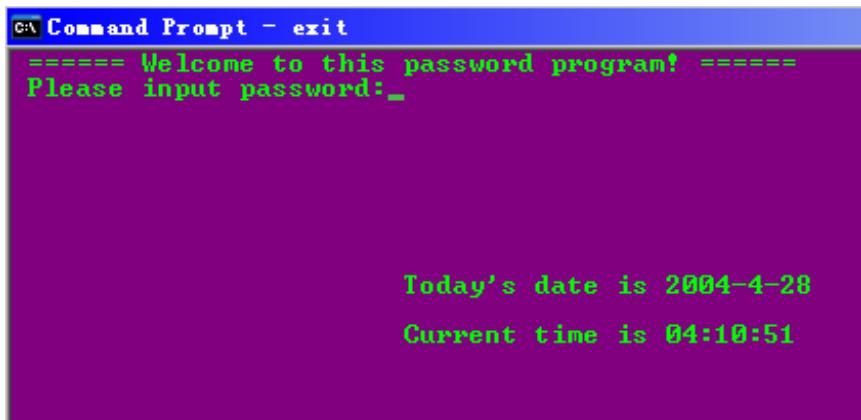


图 154-1 实例 154 程序运行效果



## 实例解析

在许多程序中要求输入用户名和密码，本实例程序采用输入并禁止回显的系统控制台，实现了一个简单的密码输入程序。程序采用 Turbo C 提供的 getpass() 函数，实现从控制台读入口令，并禁止回显，最多可以支持 8 个字符的输入。

程序中用到的主要函数如下。

(1) char \*getpass(char \*prompt)，该函数从键盘读入一个口令并禁止回显，返回一个字符指针指向获得的字符串，最多为 8 个字符（不包括作为该字符串结束标志的空终结符）。

(2) void getdate(struct date \*datep) 和 void gettime(struct time \*timep)，这两个函数包含在 dos.h 头文件中，分别用来获得系统当前的日期和时间。其指针型参数 datep 和 timep 指向取到的结果。调用这两个函数后，返回值分别使用 datep.da\_year, datep.da\_mon, datep.da\_day 以及 timep.ti\_hour, timep.ti\_min, timep.ti\_sec 来表示具体的日期和时间。



## 程序代码

### 【程序 154】 设计口令程序

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

void main(void)
{
    struct date today;
    struct time now;
    getdate(&today); /*把系统当前日期存入 today 所指向的 date 结构中*/
    gettime(&now); /*把系统当前时间存入 now 所指向的 time 结构中*/
    /*设定字符颜色和背景色*/
    textcolor(LIGHTGREEN);
    textbackground(MAGENTA);
    /*当输入口令不对时，反复进行以下循环*/
    do{
        clrscr(); /*调用清屏函数*/
        gotoxy(25,10);
        printf("Today's date is %d-%d-%d\n",today.da_year,today.da_mon,today.da_day);
        gotoxy(25,12);
        printf("Current time is %02d:%02d:%02d\n",now.ti_hour,now.ti_min,now.ti_sec);
        gotoxy(1,1);
        printf(" ===== Welcome to this password program! =====\n Please input ");
    }
    while (atoi((char *) getpass("password:")) != today.da_mon+now.ti_hour);
    /*如果输入正确，则显示正确信息并退出*/
    textcolor(WHITE);
    textbackground(BLACK);
    clrscr();
    gotoxy(1,1);
    printf(" Password Correct!!\n");
    printf(" Press any key to quit...\n");
    getch();
}
```



## 归纳注释

程序先在屏幕上显示当前日期和时间，并要求用户输入 password（本例设定为屏幕上显示的月和小时之和），如果输入不对，则不停地循环。当然，也可以设计成输入 3 次密码不对，即退出程序，这是程序设计中经常用到，以便防止非授权人恶意使用程序。

# 实例 155 程序自我保护



## 实例说明

本实例通过系统调用来实现程序的自我保护——“程序自杀”。程序运行效果如图 155-1 所示（在 Borland C++ 3.1 中编译运行）。

```
cx Command Prompt
E:\BORLANDC\BIN>time
The current time is: 4:21:19.13
Enter the new time:

E:\BORLANDC\BIN>155
You are a legitimate user to run this program!

E:\BORLANDC\BIN>time
The current time is: 4:21:31.30
Enter the new time: 4:41:31

E:\BORLANDC\BIN>155
You are a illegal user to run this program!
System run error!
```

图 155-1 实例 155 程序运行效果



## 实例解析

在编制软件时，一般使用的安全检测保护技术是在系统开始运行前，对用户加以口令校对或使用期限检测，在发现非法使用时，系统立即终止运行，以此来限制软件的使用权限。但这一方法也有缺陷，就是非法使用者可以在系统退出之后，用其他软件工具对系统运行期间留在内存中的数据进行跟踪分析并修改，然后得到跳过这一安全检测保护的方法，从而可以直接进入系统。

为了增加非法使用者的破解难度，可以使系统在一旦检测到非法使用时，立即把自身的内存数据毁灭，这样就能大大加强软件的自我保护能力。



## 程序代码

### 【程序 155】 程序自我保护

```
#include <dos.h>
#include <dir.h>
#include <stdio.h>
#include <io.h>
```

```
#include <sys\stat.h>
void main( int argc, char* argv[ ] )
{
    struct time now;
    FILE* fp;
    int errno;
    gettime( &now );
    if( now.ti_min > 30 ) /*如果非法使用系统则删除程序*/
    {
        errno = chmod( argv[0], S_IWRITE );
        if((errno)&& ( fp != NULL ))
        {
            fclose( fp );
            /*将文件长度截为 0*/
            unlink( argv[0] );
            exit(0);
            /*删除本文件退出*/
        }
        else
        {
            /*如果不能删除，则打印错误退出*/
        printf( "\n You are an illegal user to run this program!\n System run error!\n" );
        exit(1);
    }
}
printf(" You are a legitimate user to run this program!\n");
getch();
return;
}
```



## 归纳注释

程序实现了简单的自我保护技术——“程序自我删除”。该程序规定必须在机器时间处在每个小时的前半小时才能启动系统，否则该使用为非法。一旦检测到非法使用时，它立即将该程序删除。在删除前，该程序首先将它本身的执行程序的长度截止为 0，这样即使非法使用者恢复了被删掉的程序，得到的也只是一个长度为 0 的空文件，毫无用处。

# 06

## 第六部分 常见试题 解答篇

### 精彩导读

- 水果拼盘问题
- 计算方差
- 字符串倒置
- 部分排序
- 求解三角方程
- 统计选票

# 实例 156 水果拼盘

## 实例说明

现有苹果、桔子、香蕉、菠萝、梨 5 种水果用来做水果拼盘，每个水果拼盘一定有 3 个水果，且这 3 个水果的种类不同，问可以制作出多少种水果拼盘？本实例使用 C 语言中的“枚举类型”来解决水果拼盘问题。程序运行后显示出可能的拼盘类型，效果如图 156-1 所示。



The terminal window shows the following output:

```
40  pineapple orange   apple
41  pineapple orange   banana
42  pineapple orange   pear
43  pineapple banana   apple
44  pineapple banana   orange
45  pineapple banana   pear
46  pineapple pear    apple
47  pineapple pear    orange
48  pineapple pear    banana
49  pear   apple    orange
50  pear   apple    banana
51  pear   apple    pineapple
52  pear   orange   apple
53  pear   orange   banana
54  pear   orange   pineapple
55  pear   banana  apple
56  pear   banana  orange
57  pear   banana  pineapple
58  pear   pineapple apple
59  pear   pineapple orange
60  pear   pineapple banana

There are 60 kinds of fruit plates.
Press any key to quit...
```

图 156-1 实例 156 程序运行效果

## 实例解析

使用枚举类型定义变量 x、y、z，其中，x、y、z 是 5 种水果中的任一种，设定由水果 x、y、z 制作一种水果拼盘，并满足  $x \neq y \neq z$ 。使用三重循环语句来组合它们，使用穷举法来计算总共有多少种水果拼盘可以制作。

## 程序代码

### 【程序 156】 水果拼盘

```
#include <stdio.h>
void main()
{
    enum fruit {apple, orange, banana, pineapple, pear}/* 定义枚举结构*/
    enum fruit x,y,z,pri;/*定义枚举变量*/
    int n,loop;
    n=0;
    for(x=apple;x<=pear;x++)
        for(y=apple;y<=pear;y++)
            if(x!=y)
```

```

{
    for(z=apple;z<=pear;z++)
        if((z!=x)&&(z!=y))
        {
            n=n+1;
            printf ("\n %-4d",n);
            for(loop=1;loop<=3;loop++)
            {
                switch(loop)
                {
                    case 1:pri=x;break;
                    case 2:pri=y;break;
                    case 3:pri=z;break;
                    default: break;
                }
                switch(pri)
                {
                    case apple: printf(" %-9s","apple");break;
                    case orange: printf(" %-9s","orange");break;
                    case banana: printf(" %-9s","banana");break;
                    case pineapple: printf(" %-9s", "pineapple");break;
                    case pear: printf(" %-9s", "pear");break;
                    default: break;
                }
            }
        }
    printf("\n\n There are %d kinds of fruit plates.\n",n);
    puts(" Press any key to quit... ");
    getch();
    return;
}

```

## 实例 157 小孩吃梨

### 实例说明

小孩买了一些梨，当即吃了一半，还不过瘾，又多吃了两个；第二天早上又将剩下的梨吃掉一半，又多吃了两个。以后每天早上都吃了前一天剩下的一半，并又多吃一个。到第 18 天只剩下一个梨了，问小孩共买了多少个梨？本实例使用“倒推法”来解决小孩吃梨问题。程序运行效果如图 157-1 所示。

```

This program is to solve
The Problem of Child's Eating Pears.
>> In the first day, the child bought 786430 pears.
Press any key to quit...

```

图 157-1 实例 157 程序运行效果



## 实例解析

解本题用到一种重要的计算方法，即“倒推法”。知道最后一天的梨数，可以一天一天倒推到第一天的梨的个数。假设第  $n$  天的梨个数为  $X_n$ ，则前一天的梨个数为  $X_{n-1}$ ，那么， $X_n=X_{n-1}-((X_{n-1})/2+1)=(X_{n-1})/2-1$ 。因此，迭代公式  $X_{n-1}=2(X_n+1)$ ，初始条件  $X_{18}=1$ 。据此，可以一步一步倒推到第一天的梨的个数。



## 程序代码

### 【程序 157】 小孩吃梨

```
#include <stdio.h> /*小孩吃梨问题*/
void main()
{   long x=1,y=18; /* 定义两个变量，并赋值*/
    clrscr(); puts("      This program is to solve");
    puts(" The Problem of Child's Eating Pears.\n");
    while(y>0) /*y>0 是循环控制条件*/
    {
        x=2*(x+1);     y--;
    }
    printf(" In the first day, the child bought %ld pears.\n",x);
    puts("\n Press any key to quit..."); getch(); return;
}
```

## 实例 158 删 除字符串中的特定字符



## 实例说明

编写函数 func 实现从字符串中删除指定的字符。同一字母的大、小写按不同字符处理。例如原字符串为“turbo c and borland c++”，从键盘上输入字符“n”，则输出后变为“turbo c ad borlad c++”。如果输入的字符在字符串中不存在，则字符串照原样输出。程序运行效果如图 158-1 所示。

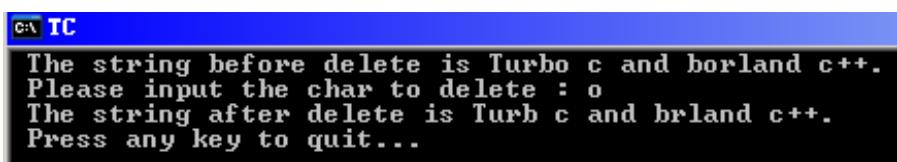


图 158-1 实例 158 程序运行效果



## 程序代码

### 【程序 158】 删 除字符串中的特定字符

```
#include <stdio.h>
int func(char s[],int c)
{
    char *q=s;
```

```

for( ; *q; q++)
if(*q != c) *(s++) = *q;
*s = '\0';
}
void main()
{
    static char str[ ]="Turbo c and borland c++";
    char ch;
    clrscr(); printf(" The string before delete is %s.\n",str);
    printf(" Please input the char to delete : ");
    scanf("%c",&ch);
    func(str,ch);
    printf(" The string after delete is %s.\n",str);
    printf(" Press any key to quit... ");
    getch();
    return;
}

```

## 实例 159 求解符号方程



### 实例说明

要求根据输入的字符方程求出未知字符，即“ $97ab=8ab\times cd+1$ ”，其中 a、b 代表 0~9 的数字，cd 为两位数， $8\times cd$  的结果为两位数， $9\times cd$  的结果为 3 位数，求 a、b、cd 代表的数，及  $8ab\times cd$  的结果。程序可以利用 i 来表示 cd ( $10 \leq i < 100$ )，再根据  $8\times cd$  为两位数和  $9\times cd$  为 3 位数来进行限定，求出 cd 的值。程序运行效果如图 159-1 所示。

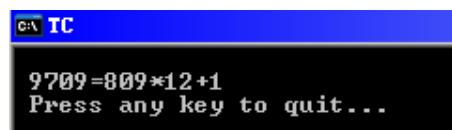


图 159-1 实例 159 程序运行效果



### 程序代码

#### 【程序 159】 求解符号方程

```

#include <stdio.h>
void main()
{
    long int a,b,c,i,j,k;
    clrscr();
    for(i=10;i<100;i++)
    {
        for(j=0;j<=9;j++)
            for(k=0;k<=9;k++)
            {
                a=8*100+j*10+k;
                b=97*100+j*10+k;
                if((b==i*a+1)&&b>=1000&&b<=10000&&8*i<100&&9*k>=100)
                    printf("\n %ld=%ld*%ld+%ld",97*100+j*10+k,800+j*10+k,i,b%j);
            }
    }
    printf("\n Press any key to quit... ");
    getch();
}

```

```
        return;  
}
```

## 实例 160 计算方差



### 实例说明

本实例要求编写函数计算并输出给定 10 个数的方差  $s = \sqrt{\frac{1}{10} \sum_{k=1}^{10} (X_k - X_0)^2}$ ，其中  $X_0 = \frac{1}{10} \sum_{k=1}^{10} X_k$  为 10 个数的平均值。例如，给定的 10 个数为 95.0、89.0、76.0、65.0、88.0、72.0、85.0、81.0、90.0、56.0，则输出为  $s=11.730729$ 。程序运行效果如图 160-1 所示。

```
TC  
The original data is:  
95.0 89.0 76.0 65.0 88.0 72.0 85.0 81.0 90.0 56.0  
The variance of the data is 11.730729.  
Press any key to quit..._
```

图 160-1 实例 160 程序运行效果



### 程序代码

#### 【程序 160】 计算方差

```
#include <stdio.h>  
#include <math.h>  
double fun(double x[10])  
{  
    int i;  
    double fc, avg=0.0, sum=0.0, abs=0.0;  
    for (i=0;i<10;i++)  
        sum+=x[i];  
    avg=sum/10;  
    for (i=0;i<10;i++)  
        abs+=(x[i]-avg)*(x[i]-avg);  
    fc=sqrt(abs/10) ;  
    return fc;  
}  
void main()  
{  
    double s, x[10]={95.0,89.0,76.0,65.0,88.0,72.0,85.0,81.0,90.0,56.0};  
    int i;  
    clrscr();  
    printf(" The original data is:\n");  
    for(i=0;i<10;i++)  
        printf(" %3.1f",x[i]);  
    printf("\n The variance of the data is %lf.\n",fun(x));
```

```
    printf(" Press any key to quit...");  
    getch();  
    return;  
}
```

## 实例 161 求取符合特定要求的素数



### 实例说明

编写函数 num(int m,int k,int xx[])将大于整数 m 且紧靠 m 的 k 个素数存入数组中，最后调用函数 readwriteDAT 从 IN161.DAT 文件中读取 10 对 m、k，分别求取符合要求的素数，并把结果输出到文件 OUT161.DAT 中。例如若输入 17，5，则应输出 19，23，29，31，37。程序运行效果如图 161-1 所示。

```
C:\TC  
This program is to get k prime numbers which are larger than m.  
>> Please input two integers to m and k : 17 5  
>> The 5 prime numbers which are larger than 17 are:  
19 23 29 31 37  
Press any key to quit...
```

图 161-1 实例 161 程序运行效果



### 程序代码

#### 【程序 161】 求取符合特定要求的素数

```
#include <conio.h>  
#include <stdio.h>  
int isP(int m)  
{  
    int i ;  
    for(i = 2 ; i < m ; i++)  
        if(m % i == 0) return 0 ;  
    return 1 ;  
}  
void num(int m,int k,int xx[ ])  
{  
    int i=0;  
    for(m=m+1;k>0;m++)  
        if(isP(m))  
    {  
        xx[i++]=m;  
        k--;  
    }  
}  
void readwriteDAT()  
{  
    int m, n, xx[1000], i;  
    FILE *rf, *wf ;
```

```

rf = fopen("IN161.DAT", "r");
wf = fopen("OUT161.DAT", "w");
for(i = 0 ; i < 10 ; i++)
{
    fscanf(rf,"%d%d",&m,&n);
    num(m,n,xx) ;
    for(m=0;m < n ; m++)
        fprintf(wf, "%d ", xx[m]);
    fprintf(wf, "\n");
}
fclose(rf);
fclose(wf);
}

void main()
{
    int m, n, xx[1000] ;
    clrscr() ;
    puts(" This program is to get k prime numbers which are larger than m.");
    printf(" >> Please input two integers to m and k : ") ;
    scanf("%d%d", &m, &n) ;
    num(m, n, xx) ;
    printf(" >> The %d prime numbers which are larger than %d are:\n ",n,m);
    for(m = 0 ; m < n ; m++)
        printf(" %d ", xx[m]);
    readwriteDAT();
    printf("\n Press any key to quit...") ;
    getch();
    return;
}

```

## 实例 162 统计符合特定条件的数



### 实例说明

已知数据文件 IN162.DAT 中存有 200 个 4 位数，并已调用读函数 readDat() 把这些数存入数组 a 中。编制一个函数 jsVal()，其功能是如果 4 位数各位上的数字均是 0 或 2 或 4 或 6 或 8，则统计出满足此条件的个数 cnt，并把这些 4 位数按从大到小的顺序存入数组 b 中。最后 main() 函数调用写函数 writeDat( ) 把结果 cnt 以及数组 b 中符合条件的 4 位数输出到 OUT162.DAT 文件中。程序运行效果如图 162-1 所示。

```

TC
The number of the satisfied integers is 25.
8688 8686 8684 8682 8680 8668 8666 8664 8662 8660
8648 8646 8644 8642 8640 8628 8626 8624 8622 8620
8608 8606 8604 8602 8600
Press any key to quit...

```

图 162-1 实例 162 程序运行效果



## 程序代码

### 【程序 162】 统计符合特定条件的数

```
#include <stdio.h>
#define MAX 200
int a[MAX], b[MAX], cnt = 0;
void jsVal()
{
    int bb[4];
    int I, j, k, flag;
    for (I=0; I<200; I++)
    {
        bb[0]=a[I]/1000;
        bb[1]=a[I]%1000/100;
        bb[2]=a[I]%100/10;
        bb[3]=a[I]%10;
        for (j=0; j<4; j++)
        {
            if (bb[j]%2==0)
                flag=1;
            else
            {
                flag=0;
                break;
            }
        }
        if (flag==1)
        {
            b[cnt]=a[I];
            cnt++;
        }
    }
    for(I=0; I<cnt-1; I++)
        for(j=I+1; j<cnt; j++)
            if (b[I]<b[j])
            {
                k=b[I];
                b[I]=b[j];
                b[j]=k;
            }
}
void readDat()
{
    int i ;
    FILE *fp ;
    fp = fopen("IN162.DAT", "r") ;
    for(i = 0 ; i < MAX ; i++)
        fscanf(fp, "%d", &a[i]) ;
    fclose(fp) ;
}
void main()
{
```

```

int i,j;
clrscr();
readDat() ;
jsVal() ;
printf(" The number of the satisfied integers is %d.\n ", cnt) ;
for(i = 0,j=0 ; i < cnt ; i++)
{
    printf("%d ", b[i]) ;
    if(j==9)
    {
        printf("\n ");
        j=0;
    }
    else
        j++;
}
printf("\n") ;
writeDat() ;
printf(" Press any key to quit...") ;
getch();
}
writeDat()
{
    FILE *fp ;
    int i ;
    fp = fopen("OUT162.DAT", "w") ;
    fprintf(fp, "%d\n", cnt) ;
    for(i = 0 ; i < cnt ; i++)
        fprintf(fp, "%d\n", b[i]) ;
    fclose(fp) ;
}

```

## 实例 163 字符串倒置



### 实例说明

编写函数 READDAT()实现从文件 IN163.DAT 中读取一篇英文文章存入到字符串数组中，并编写函数 STROR(), 以行为单位把字符串中的所有小写字母 o 左边的字符串内容移到该串的右边存放，然后把小写字母 o 删除，最后把已处理的字符串仍按行重新存入字符串数组中，最后调用函数 WRITEDAT()把结果输出到文件 OUT163.DAT 中。

例如原文为：

```
You can create an index on any field.  
you have the correct record.
```

结果为：

```
n any field.Yu can create an index  
rd.yu have the crrect rec
```

原始数据文件存放的格式是每行的宽度均小于 80 个字符，含标点符号和空格。

程序运行效果如图 163-1 所示。

```
r ecpert and execute, F  
f particulars, and perhaps judge  
unsels, ne by ne; but the general c  
f affairs, and the plts and marshallng  
se that are learned. cme best frm th  
th; I spend to much time in studies is sl  
rnament, t use them to much fr  
n; is affectati  
lly by their rules, t make judgement wh  
lar. is the humur f a sch  
They perfect nature,  
and are perfectec by experience:  
r natural abilities are like natural plants, f  
yning by study; that need pr  
ns and studies themselves d give frth directi  
o much at large, t  
unded in by experience. except they be b  
ntemn studies, Crafty men c  
simple men admire them,  
and wise men use them;  
wn use; fr they teach nt their  
ut them, but that is a wisdm with  
n. and above them, wn by bservati  
Press any key to continue . . .
```

图 163-1 实例 163 程序运行效果



## 程序代码

### 【程序 163】 字符串倒置

```
#include "stdio.h"  
#include "string.h"  
#include "conio.h"  
char xx[50][80];  
int maxline=0;  
int ReadDat(void);  
void WriteDat(void);  
void StrOR(void)  
{  
    int I,j,k,index,strl;  
    char ch;  
    for(I=0;I<maxline;I++)  
    {  
        strl=strlen(xx[I]);  
        index=strl;  
        for(j=0;j<strl;j++)  
            if(xx[I][j]=='o')  
            {  
                for(k=j;k<strl-1;k++)  
                    xx[I][k]=xx[I][k+1];  
                xx[I][strl-1]= ' ';  
                index=j;  
            }  
        for(j=strl-1;j>=index;j--)  
        {  
            ch=xx[I][strl-1];  
            for(k=strl-1;k>0;k--)
```

```

        xx[I][k]=xx[I][k-1];
        xx[I][0]=ch;
    }
}
void main()
{
    clrscr();
    if(ReadDat())
    {
        printf("Can't open the file!\n");
        return;
    }
    StrOR();
    WriteDat();
    system("pause");
}
int ReadDat(void)
{
    FILE *fp; int i=0; char *p;
    if((fp=fopen("in163.dat", "r"))==NULL) return 1;
    while(fgets(xx[i], 80, fp)!=NULL)
    {
        p=strchr(xx[i], '\n');
        if(p)
            *p=0;
        i++;
    }
    maxline=i;
    fclose(fp);
    return 0;
}
void WriteDat(void)
{
    FILE *fp;
    int i;
    fp=fopen("out163.dat", "w");
    for(i=0;i<maxline;i++)
    {
        printf("%s\n", xx[i]);
        fprintf(fp,"%s\n", xx[i]);
    }
    fclose(fp);
}

```

## 实例 164 部分排序



### 实例说明

在文件 in164.dat 中有 200 个正整数，且每个数均在 1000~9999 之间。函数 ReadDat() 用于读

取这 200 个数存放到数组 aa 中。要求编制函数 jsSort(), 按每个数的后 3 位的大小进行升序排列, 然后取出满足此条件的前 10 个数依次存入数组 bb 中, 如果后 3 位的数值相等, 则按原先的数值进行降序排列。最后调用函数 WriteDat() 把结果数组 bb 输出到文件 out164.dat 中。

例如处理前:

6012 5099 9012 7025 8088

处理后:

9012 6012 7025 8088 5099

程序运行效果如图 164-1 所示。

```
No. 1: 2001  
No. 2: 1001  
No. 3: 1012  
No. 4: 3016  
No. 5: 5017  
No. 6: 7026  
No. 7: 7031  
No. 8: 8034  
No. 9: 1034  
No.10: 8036  
Press any key to quit....
```

图 164-1 实例 164 程序运行效果



## 程序代码

### 【程序 164】 部分排序

```
#include<stdio.h>
#include<string.h>
#include<conio.h>
int aa[200],bb[10];
void jsSort()
{
    int I,j,data;
    for(I=0;I<199;I++)
        for(j=I+1;j<200;j++)
            {if (aa[I]%1000>aa[j]%1000)
             {data=aa[I];aa[I]=aa[j];aa[j]=data;}
             else if(aa[I]%1000==aa[j]%1000)
             if(aa[I]<aa[j])
                 {data=aa[I];aa[I]=aa[j];aa[j]=data;}
             }
    for(I=0;I<10;I++)
        bb[I]=aa[I];
}
void PressKeyToQuit()
{
    printf("\n Press any key to quit... ");
    getch();
    return;
}
void main()
{
    readDat();
    jsSort();
    writeDat();
    PressKeyToQuit();
}
readDat()
{
    FILE *in;
    int i;
    in=fopen("in164.dat","r");
    for(i=0; i<200; i++) fscanf(in,"%d",&aa[i]);
}
```

```

fclose(in);
}
writeDat()
{
    FILE *out;
    int i;
    clrscr();
    out=fopen( "out164.dat" , "w" );
    for(i=0; i<10; i++){
        printf(" No.%2d: %d\n",i+1,bb[i]);
        fprintf(out,"%d\n",bb[i]);
    }
    fclose(out);
}

```

## 实例 165 产品销售记录处理



### 实例说明

已知在文件 IN165.DAT 中存有 100 个产品销售记录，每个产品销售记录由产品代码 dm（字符型 4 位）、产品名称 mc（字符型 10 位）、单价 dj（整型）、数量 sl（整型）和金额 je（长整型）4 部分组成。其中金额=单价×数量。

函数 ReadDat() 用于读取这 100 个销售记录并存入结构数组 sell 中。

要求编制函数 SortDat()，按产品代码从大到小进行排列，若产品代码相同，则按金额从大到小进行排列，最终排列结果仍存入结构数组 sell 中，最后调用函数 WriteDat() 把结果输出到文件 OUT165.DAT 中。

程序运行效果如图 165-1 所示（运行前后两个文件的比较）。

M100	machine	500	69	34500	P219	power	1190	1400	1666000
M101	machine	510	139	70890	P218	power	1180	1330	1569400
M102	machine	520	209	108680	P217	power	1170	1260	1474200
M103	machine	530	279	147870	P216	power	1160	1190	1380400
M104	machine	540	349	188460	P215	power	1150	1120	1288000
M105	machine	550	419	230450	P214	power	1140	1050	1197000
M106	machine	560	489	273840	P213	power	1130	980	1107400
M107	machine	570	559	318630	P212	power	1120	910	1019200
M108	machine	580	629	364820	P211	power	1110	840	932400
M109	machine	590	699	412410	P210	power	1100	770	847000
M110	machine	600	769	461400	P209	power	1090	700	763000
M111	machine	610	839	511790	P208	power	1080	630	680400
M112	machine	620	909	563580	P207	power	1070	560	599200
M113	machine	630	979	616770	P206	power	1060	490	519400
M114	machine	640	1049	671360	P205	power	1050	420	441000
M115	machine	650	1119	727350	P204	power	1040	350	364000
M116	machine	660	1189	784740	P203	power	1030	280	288400
M117	machine	670	1259	843530	P202	power	1020	210	214200
M118	machine	680	1329	903720	P201	power	1010	140	141400
M119	machine	690	1399	965310	P200	power	1000	70	70000
P200	power	1000	70	70000	M519	motor	2690	1403	3774070
P201	power	1010	140	141400	M518	motor	2680	1333	3572440
P202	power	1020	210	214200	M517	motor	2670	1263	3372210
P203	power	1030	280	288400	M516	motor	2660	1193	3173380
P204	power	1040	350	364000	M515	motor	2650	1123	2975950

IN165.DAT

OUT165.DAT

图 165-1 实例 165 程序运行效果



## 程序代码

### 【程序 165】 产品销售记录处理

```
#include<stdio.h>
#include<mem.h>
#include<string.h>
#include<conio.h>
#include<stdlib.h>
#define MAX 100
typedef struct{
char dm[5]; /*产品代码*/
char mc[11]; /*产品名称*/
int dj; /*单价*/
int sl; /*数量*/
long je; /*金额*/
}PRO;
PRO sell[MAX];
void ReadDat();
void WriteDat();
void SortDat()
{
    int I,j;
    PRO xy;
    for(I=0;I<99;I++)
        for(j=I+1;j<100;j++)
            if(strcmp(sell[I].dm,sell[j].dm)<0)
                {xy=sell[I];sell[I]=sell[j];sell[j]=xy;}
            else if(strcmp(sell[I].dm,sell[j].dm)==0)
                if(sell[I].je<sell[j].je)
                    {xy=sell[I]; sell[I]=sell[j]; sell[j]=xy;}
}
void main()
{
    memset(sell,0,sizeof(sell));
    ReadDat();
    SortDat();
    WriteDat();
}
void ReadDat()
{
    FILE *fp;
    char str[80],ch[11];
    int i;
    fp=fopen("IN165.DAT","r");
    for(i=0;i<100;i++){
        fgets(str,80,fp);
        memcpy(sell[i].dm,str,4);
        memcpy(sell[i].mc,str+4,10);
        memcpy(ch,str+14,4);ch[4]='\0';
        sell[i].dj=atoi(ch);
        memcpy(ch,str+18,5);ch[5]='\0';
        sell[i].sl=atoi(ch);
    }
}
```

```

sell[i].je=(long)sell[i].dj*sell[i].sl;
fclose(fp);
}
void WriteDat(void)
{
    FILE *fp;
    int i;
    fp=fopen("OUT165.DAT", "w");
    for(i=0;i<100;i++){
        fprintf(fp,"%s %s %4d %5d %10ld\n", sell[i].dm,sell[i].mc,sell[i].dj,
sell[i].sl,sell[i].je);
    }
    fclose(fp);
}

```

## 实例 166 特定要求的字符编码



### 实例说明

函数 ReadDat()用于实现从文件 IN166.DAT 中读取一篇英文文章，存入到字符串数组 xx 中。要求编制函数 encryptChar()，按给定的替代关系对数组 xx 中的所有字符进行替代，仍存入数组 xx 的对应的位置上，最后调用函数 WriteDat()把结果 xx 输出到文件 OUT166.DAT 中。

替代关系 “ $f(p)=p*11 \bmod 256$ ”，其中 p 是数组中某一个字符的 ASCII 值， $f(p)$ 是计算后新字符的 ASCII 值。如果计算后  $f(p)$ 值小于等于 32 或大于 130，则该字符不变，否则将  $f(p)$ 所对应的字符进行替代。原始数据文件存放的格式是每行的宽度均小于 80 个字符。

程序运行效果如图 166-1 所示。

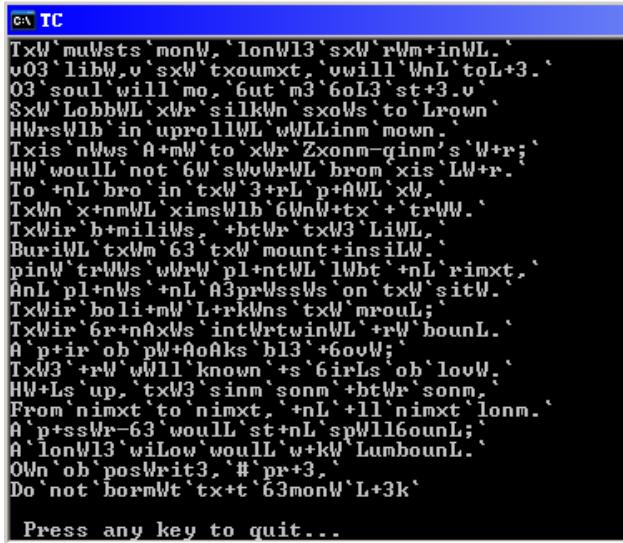


图 166-1 实例 166 程序运行效果



### 程序代码

#### 【程序 166】 特定要求的字符编码

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

```

#include<string.h>
#include<conio.h>
#include<ctype.h>
unsigned char xx[50][80];
int maxline=0; /*文章的总行数*/
int ReadDat(void);
void WriteDat(void);
void encryptChar()
{
    int I;
    char *pf;
    for(I=0;I<maxline;I++)
    {pf=xx[I];
    while(*pf!=0)
    {if(*pf*11%256>130||*pf*11%256<=32);
    else
    *pf=*pf*11%256;
    pf++; }
    }
}
void PressKeyToQuit()
{
    printf("\n Press any key to quit... ");
    getch();
    return;
}
void main()
{
    clrscr();
    if(ReadDat())
    {
        printf(" Can't open file IN166.DAT! \n");
        return;
    }
    encryptChar();
    WriteDat();
    PressKeyToQuit();
}
int ReadDat(void)
{
    FILE *fp;
    int i=0;
    unsigned char *p;
    if((fp=fopen("in166.dat","r"))==NULL) return 1;
    while(fgets(xx[i],80,fp)!=NULL)
    {
        p=strchr(xx[i],'\n');
        if(p)*p=0;
        i++;
    }
    maxline=i;
    fclose(fp);
    return 0;
}

```

```

void WriteDat(void)
{
    FILE *fp;
    int i;
    fp=fopen("out166.dat", "w");
    for(i=0;i<maxline;i++)
    {
        printf("%s\n",xx[i]);
        fprintf(fp,"%s\n",xx[i]);
    }
    fclose(fp);
}

```

## 实例 167 求解三角方程



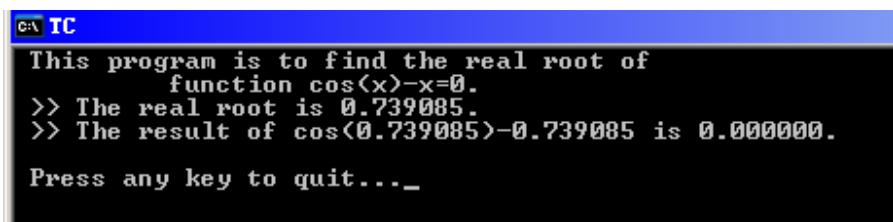
### 实例说明

利用简单迭代方法求方程  $\cos(X)-X=0$  的一个实根,  $X_{n+1}=\cos(X_n)$ 。

迭代步骤如下:

- (1) 取  $X_1$  初值为 0.0;
- (2)  $X_0=X_1$ , 把  $X_1$  的值赋给  $X_0$ ;
- (3)  $X_1=\cos(X_0)$ , 求出一个新的  $X_1$ ;
- (4) 若  $X_0-X_1$  绝对值小 0.000001, 执行步骤 (5), 否则执行步骤 (2);
- (5) 所求  $X_1$  就是方程  $\cos(X)-X=0$  的一个实根, 可以作为函数值返回。

编写函数 countvalue() 实现程序的要求, 最后调用函数 WRITEDAT() 把结果输出到文件 OUT167.DAT 中。程序运行效果如图 167-1 所示。



```

C:\ TC
This program is to find the real root of
      function cos(x)-x=0.
>> The real root is 0.739085.
>> The result of cos(0.739085)-0.739085 is 0.000000.
Press any key to quit...

```

图 167-1 实例 167 程序运行效果



### 程序代码

#### 【程序 167】 求解三角方程

```

#include <conio.h>
#include <math.h>
#include <stdio.h>
float countvalue()
{
    float x0,x1=0.0;
    while(1)
    {

```

```

x0=x1;
x1=cos(x0);
if(fabs(x0-x1)<1e-6)
    break;
}
return x1;
}
void PressKeyToQuit()
{
    printf("\n Press any key to quit... ");
    getch();
    return;
}
main()
{
    clrscr();
    puts(" This program is to find the real root of");
    puts("      function cos(x)-x=0.");
    printf(" >> The real root is %f.\n",countvalue());
    printf(" >> The result of cos(%f)-%f is %f.\n",countvalue(),countvalue(),
cos(countvalue())-countvalue());
    writeDat();
    PressKeyToQuit();
}
writeDat()
{
    FILE *wf;
    wf=fopen("OUT167.DAT", "w");
    fprintf(wf, "%f\n", countvalue());
    fclose(wf);
}

```

## 实例 168 新完全平方数



### 实例说明

本实例实现在 3 位整数（100~999）中寻找符合条件的整数并依次从小到大存入数组中，特定条件为它既是完全平方数，又有两位数字相同，例如 144、676 等。要求满足该条件的整数的个数通过所编制的函数返回。最后调用函数 writeDat() 把结果输出到文件 out168.dat 中。

程序运行效果如图 168-1 所示。

```

C:\ TC
This program is to find the Perfect Square Numbers.
which have 3 digits, and 2 of them are the same.
These numbers are as follows:
100 121 144 225 400 441 484 676 900
Press any key to quit...

```

图 168-1 实例 168 程序运行效果



## 程序代码

### 【程序 168】 新完全平方数

```
#include <stdio.h>
int jsvalue(int bb[])
{
    int I,j,k=0;
    int hun,ten,data;
    for(I=100;I<=999;I++)
    {
        j=10;
        while(j*j<=I)
        {
            if (I==j*j)
            {
                hun=I/100;data=I-hun*100;
                ten=data/10;data=data-ten*10;
                if(hun==ten||hun==data||ten==data)
                {
                    bb[k]=I;
                    k++;
                }
            }
            j++;
        }
    }
    return k;
}
void PressKeyToQuit()
{
    printf("\n Press any key to quit...");
    getch();
    return;
}
main()
{
    int b[20],num;
    clrscr();
    puts(" This program is to find the Perfect Square Numbers.");
    puts(" which have 3 digits, and 2 of them are the same.");
    puts(" These numbers are as follows:");
    num=jsvalue(b);
    writeDat(num,b);
    PressKeyToQuit();
}
writeDat(int num,int b[])
{
    FILE *out;
    int i;
    out=fopen("out168.dat","w");
    fprintf(out,"%d\n",num);
    for(i=0;i<num;i++)

```

```

{
    fprintf(out,"%d\n",b[i]);
    printf(" %d",b[i]);
}
fclose(out);
}

```

## 实例 169 三重回文数



### 实例说明

寻找并输出 11~999 之间的数 m，它满足  $m$ 、 $m^2$  和  $m^3$  均为回文数。所谓回文数是指其各位数字左右对称的整数，例如 121，676，94249 等。满足上述条件的数如  $m=11$ ， $m^2=121$ ， $m^3=1331$  皆为回文数。要求编制函数 int svalue(long m)实现此功能，如果是回文数，则函数返回 1，反之则返回 0。最后把结果输出到文件 out169.dat 中。

程序运行效果如图 169-1 所示。

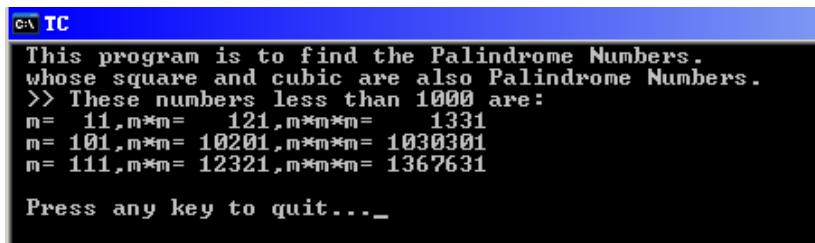


图 169-1 实例 169 程序运行效果



### 程序代码

#### 【程序 169】 三重回文数

```

#include <stdio.h>
int jsvalue(long n)
{
    int I,strl,half;
    char xy[20];
    ltoa(n,xy,10);
    strl=strlen(xy);
    half=strl/2;
    for(I=0;I<half;I++)
        if(xy[I]!=xy[--strl])
            break;
    if(I>=half) return 1;
    else return 0;
}
void PressKeyToQuit()
{
    printf("\n Press any key to quit...");
    getch();
    return;
}

```

```

main()
{
    long m;
    FILE *out;
    clrscr();
    puts(" This program is to find the Palindrome Numbers.");
    puts(" whose square and cubic are also Palindrome Numbers.");
    puts(" >> These numbers less than 1000 are:");
    out=fopen("out169.dat", "w");
    for(m=11;m<1000;m++)
    {
        if(jsvvalue(m)&&jsvalue(m*m)&&jsvalue(m*m*m))
        {
            printf(" m=%4ld,m*m=%6ld,m*m*m=%8ld \n",m,m*m,m*m*m);
            fprintf(out,"m=%4ld,m*m=%6ld,m*m*m=%8ld \n",m,m*m,m*m*m);
        }
    }
    fclose(out);
    PressKeyToQuit();
}

```

## 实例 170 奇数方差

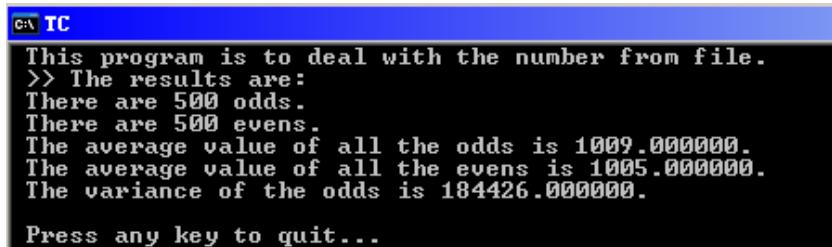
### 实例说明

函数 ReadDat()用于从文件 IN170.DAT 中读取 1000 个十进制整数到数组 xx 中。要求编制函数 Compute()分别计算出 xx 中奇数的个数 odd, 奇数的平均值 ave1, 偶数的平均值 ave2 以及所有奇数的方差 totfc 的值, 最后调用函数 WriteDat()把结果输出到 OUT170.DAT 文件中。

计算方差的公式为  $\text{totfc} = \frac{1}{N} \sum_{i=1}^N (\text{xx}[i] - \text{ave1})^2$ , 其中 N 为奇数的个数, xx[i] 为奇数, ave1 为

奇数的平均值。

原始数据文件存放的格式为每行存放 10 个数, 并用空格隔开 (每个数均大于 0 且小于等于 2000)。程序运行效果如图 170-1 所示。



```

C:\TC
This program is to deal with the number from file.
>> The results are:
There are 500 odds.
There are 500 evens.
The average value of all the odds is 1009.000000.
The average value of all the evens is 1005.000000.
The variance of the odds is 184426.000000.

Press any key to quit...

```

图 170-1 实例 170 程序运行效果

### 程序代码

#### 【程序 170】 奇数方差

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

```

#include <stdlib.h>
#include <string.h>
#define MAX 1000
int xx[MAX],odd=0,even=0;
double avel=0.0,ave2=0.0,totfc=0.0;
void WriteDat(void);
int ReadDat(void)
{
    FILE *fp;
    int i,j;
    if((fp=fopen("IN170.DAT", "r" ))==NULL)
        return 1;
    for(i=0;i<100;i++)
    {
        for(j=0;j<10;j++)
            fscanf(fp, "%d ", &xx[i*10+j]);
        fscanf(fp, "\n");
        if(feof(fp))
            break;
    }
    fclose(fp);
    return 0;
}
void Compute(void)
{
    int I, yy[MAX];
    for(I=0;I<1000;I++)
        if(xx[I]%2)
        {
            odd++;
            avel+=xx[I];
            yy[odd-1]=xx[I];
        }
        else
        {
            even++;
            ave2+=xx[I];
        }
    avel/=odd;
    ave2/=even;
    for(I=0;I<odd;I++)
        totfc+=(yy[I]-avel)*(yy[I]-avel)/odd;
}
void PressKeyToQuit()
{
    printf("\n Press any key to quit...");
    getch();
    return;
}
void main()
{
    int i;
    clrscr();
    puts(" This program is to deal with the number from file.");
}

```

```

puts(" >> The results are:");
for(i=0;i<MAX;i++)xx[i]=0;
if(ReadDat())
{
    printf(" Can't open data file IN170.DAT!\n");
    return;
}
Compute();
printf(" There are %d odds.\n There are %d evens.\n The average value of all
the odds is %lf.\n The average value of all the evens is %lf.\n The variance of
the odds is %lf.\n",odd,even,ave1,ave2,totfc);
WriteDat();
PressKeyToQuit();
}
void WriteDat(void)
{
    FILE *fp;
    int i;
    fp=fopen("OUT170.DAT", "w");
    fprintf(fp,"%d\n%d\n%lf\n%lf\n%lf\n",odd,even,ave1,ave2,totfc);
    fclose(fp);
}

```

## 实例 171 统计选票



### 实例说明

对 10 个候选人进行选举，现有一个 100 条记录的选票数据文件 IN171.DAT，其数据存放的格式是每条记录的长度均为 10 位，第一位表示第一个人的选中情况，第二位表示第二个人的选中情况，依此类推，内容均为字符 0 和 1，1 表示此人被选中，0 表示此人未被选中，全选或不选均为无效的选票。

给定函数 ReadDat() 的功能是把选票数据读入到字符串数组 xx 中。要求编制函数 CountRs() 来统计每个人的选票数并把得票数依次存入 yy[0]~yy[9] 中。把结果 yy 输出到文件 OUT171.DAT 中。程序运行效果如图 171-1 所示。

```

C:\ TC
NO. 1 notes=48
NO. 2 notes=50
NO. 3 notes=55
NO. 4 notes=49
NO. 5 notes=50
NO. 6 notes=51
NO. 7 notes=48
NO. 8 notes=44
NO. 9 notes=61
NO.10 notes=55
Press any key to continue . . .

```

图 171-1 实例 171 程序运行效果



### 程序代码

#### 【程序 171】 统计选票

```

#include<stdio.h>
char xx[100][11];
int yy[10];
int ReadDat(void);
void WriteDat(void);
void CountRs(void)

```

```

{
    int i,j,k;
    for(i=0;i<100;i++)
    {
        k=0;
        for(j=0;j<10;j++)
            if(xx[i][j]=='1')
                k++;
        if(k==0||k==10)
            continue;
        for(j=0;j<10;j++)
            if(xx[i][j]=='1')
                yy[j]++;
    }
}
void main()
{
    int i;
    for(i=0;i<10;i++) yy[i]=0;
    if(ReadDat())
    {
        printf("Can't open file IN171.DAT!\n\007");
        return;
    }
    CountRs();
    WriteDat();
    system("pause");
}
int ReadDat(void)
{
    FILE *fp;
    int i;
    char tt[13];
    clrscr();
    if((fp=fopen("in171.dat","r"))==NULL) return 1;
    for(i=0;i<100;i++)
    {
        if(fgets(tt,13,fp)==NULL) return 1;
        memcpy(xx[i],tt,10);
        xx[i][10]=0;
    }
    fclose(fp);
    return 0;
}
void WriteDat(void)
{
    FILE *fp;
    int i;
    fp=fopen("out171.dat","w");
    for(i=0;i<10;i++)
    {
        fprintf(fp,"%d\n",yy[i]);
        printf(" NO.%2d notes=%d\n",i+1,yy[i]);
    }
    fclose(fp);
}

```

# 实例 172 同时整除

## 实例说明

要求编写函数 void countvalue(int \*a, int \*n), 求出 1~1000 之内能被 7 或 11 整除, 但不能同时被 7 和 11 整除的所有整数, 将它们放在数组 a 中, 并通过 n 返回满足条件的数的个数。程序运行效果如图 172-1 所示。

```
which can be divided exactly by 7 or 11,
but can not be divided exactly both by 7 and 11.
These numbers less than 1000 are:
    7   11   14   21   22   28   33   35   42
    49   55   56   63   66   70   84   88   91
    99  105  110  112  119  121  126  132  133
   143  147  161  165  168  175  176  182  187
   196  198  203  209  210  217  220  224  238
   245  252  253  259  264  266  273  275  280
   287  294  297  301  315  319  322  329  330
   341  343  350  352  357  363  364  371  374
   392  396  399  406  407  413  418  420  427
   434  440  441  448  451  455  469  473  476
   484  490  495  497  504  506  511  517  518
   528  532  546  550  553  560  561  567  572
   581  583  588  594  595  602  605  609  623
   630  637  638  644  649  651  658  660  665
   672  679  682  686  700  704  707  714  715
   726  728  735  737  742  748  749  756  759
   777  781  784  791  792  798  803  805  812
   819  825  826  833  836  840  854  858  861
   869  875  880  882  889  891  896  902  903
   913  917  931  935  938  945  946  952  957
   966  968  973  979  980  987  990  994
Press any key to quit...
```

图 172-1 实例 172 程序运行效果

## 程序代码

### 【程序 172】 同时整除

```
#include <conio.h>
#include <stdio.h>
void countvalue(int *a,int *n)
{
    int I;
    *n=0;
    for(I=1;I<=1000;I++)
        if(I%7==0&&I%11)
        {
            *a=I;
            *n=*n+1;
            a++;
        }
        else if(I%7&&I%11==0)
        {
            *a=I;
            *n=*n+1;
            a++;
        }
}
```

```

void PressKeyToQuit()
{   printf("\n Press any key to quit... ");
    getch();
    return;
}
main()
{   int aa[1000],n,k;
    clrscr();
    puts(" This program is to find numbers\n which can be divided exactly by 7
or 11,\n but can not be divided exactly both by 7 and 11.");
    puts(" These numbers less than 1000 are:");
    countvalue(aa,&n);
    for(k=0;k<n;k++)
        if((k+1)%10==0)
            printf("\n");
        else
            printf("%5d",aa[k]);
    writeDAT();
    PressKeyToQuit();
}
writeDAT()
{
    int aa[1000],n,k;
    FILE *fp;
    fp=fopen( "out172.dat" , "w" );
    countvalue(aa,&n);
    for(k=0;k<n;k++)
        if((k+1)%10==0)
            fprintf(fp, "\n");
        else
            fprintf(fp, "%5d",aa[k]);
    fprintf(fp, "\n");
    fclose(fp);
}

```

## 实例 173 字符左右排序



函数 ReadDat()用于从文件 in186.dat 中读取 20 行数据存放到字符串数组 xx 中（每行字符串长度均小于 80）。

要求编制函数 jsSort(), 以行为单位对字符串按给定的条件进行排序, 排序后的结果仍按行重新存入字符串数组 xx 中, 最后调用函数 WriteDat()把结果 xx 输出到文件 out173.dat 中。

**排序条件:** 将字符串从中间一分为二, 左边部分按字符的 ASCII 值升序排序, 排序后左边部分与右边部分进行交换。如果原字符串长度为奇数, 则最中间的字符不参予处理, 仍放在原位置上。

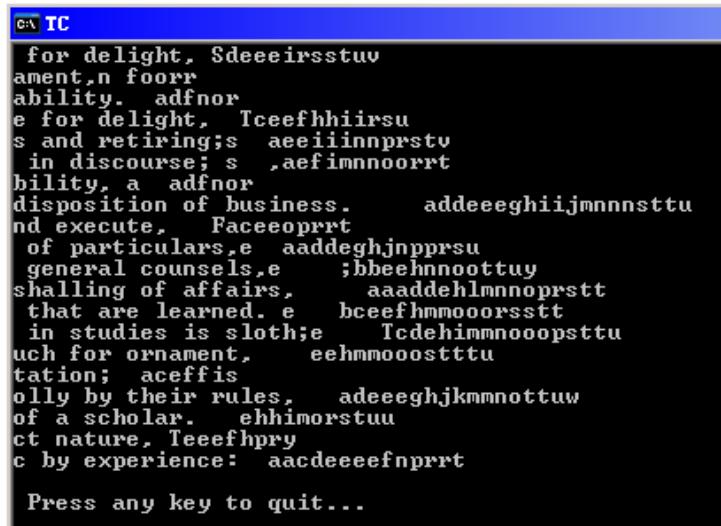
例如, 源字符串:

```
dcbahgfe
432198765
```

则处理后字符串：

```
hgfe abcd  
876591234
```

程序运行效果如图 173-1 所示。



```
for delight, Sdeeeirsstuv
ament.n foorr
ability. adfnor
e for delight, Tceefhhirsu
s and retiring;s aeeiiinnprstv
in discourse; s ,aefimnnoorrt
bility, a adfnor
disposition of business. addeeeghijmnnnstu
nd execute, Faceeoprrt
of particulars,e aaddeghjnppls
general counsels,e ;bbeehnnoottuy
shalling of affairs, aaaddehlmnnoprstt
that are learned. e bceefhmooorsstt
in studies is sloth;e Tcdehimmnnoopsttu
uch for ornament, eehmmooostttu
tation; aceffis
olly by their rules, adeeeghjkmmnottuw
of a scholar. ehhimorstu
ct nature, Teeefhpry
c by experience: aacdeeeefnprrt

Press any key to quit...
```

图 173-1 实例 173 程序运行效果



## 程序代码

### 【程序 173】 字符左右排序

```
#include<stdio.h>
#include<string.h>
#include<conio.h>
char xx[20][80];
void jsSort()
{
    int i,strl,half,j,k;
    char ch;
    for(i=0;i<20;i++) /*行循环*/
    {
        strl=strlen(xx[i]); /*每行长度*/
        half=strl/2;
        for(j=0;j<half-1;j++) /*每行的第 j 个位置*/
            for(k=j+1;k<half;k++)
                if(xx[i][j]>xx[i][k])
                {
                    ch=xx[i][j]; /*每次将最小数赋给 xx[i][j]*/
                    xx[i][j]=xx[i][k];
                    xx[i][k]=ch;
                }
        for(j=half-1,k=strl-1;j>=0;j--,k--)
        {
            ch=xx[i][j];
            xx[i][j]=xx[i][k];
            xx[i][k]=ch;
        }
    }
}
```

```

void PressKeyToQuit()
{
    printf("\n Press any key to quit...");
    getch();
    return;
}
void main()
{
    readDat();
    jsSort();
    writeDat();
    PressKeyToQuit();
}
readDat()
{
    FILE *in;
    int i=0;
    char *p;
    in=fopen("in173.dat","r");
    while(i<20&&fgets(xx[i],80,in)!=NULL)
    {
        p=strchr(xx[i],'\n');
        if(p)
            *p=0;
        i++;
    }
    fclose(in);
}
writeDat()
{
    FILE *out;
    int i;
    clrscr();
    out=fopen("out173.dat","w");
    for(i=0;i<20;i++)
    {
        printf("%s\n",xx[i]);
        fprintf(out,"%s\n",xx[i]);
    }
    fclose(out);
}

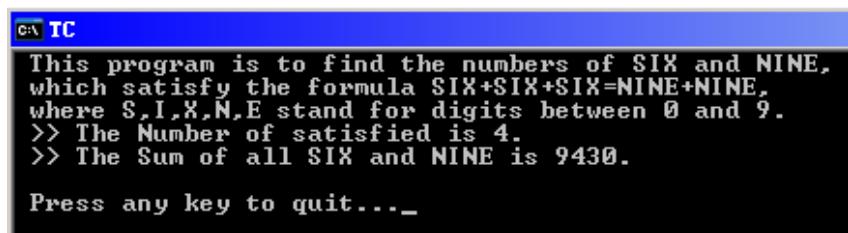
```

## 实例 174 符号算式求解

### 实例说明

计算出满足的条件 SIX+SIX+SIX=NINE+NINE 的自然数 SIX 和 NINE 的个数 cnt，以及满足此条件的所有 SIX 与 NINE 的和 sum。编写函数 countvalue() 实现程序的要求，最后调用函数 writedat() 把结果 cnt 和 sum，输出到文件 out187.dat 中，其中 S, I, X, N, E 各代表一个十进制数字。

程序运行效果如图 174-1 所示。



```
This program is to find the numbers of SIX and NINE,
which satisfy the formula SIX+SIX+SIX=NINE+NINE,
where S,I,X,N,E stand for digits between 0 and 9.
>> The Number of satisfied is 4.
>> The Sum of all SIX and NINE is 9430.

Press any key to quit...
```

图 174-1 实例 174 程序运行效果

## 程序代码

### 【程序 174】 符号算式求解

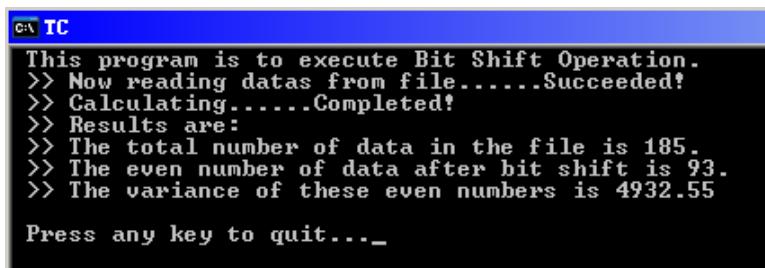
```
#include <stdio.h>
int cnt,sum;
void countvalue()
{
    int i;
    for(i=666;i<=999;i=i+2)
        if((i/10%10==(3*i/2)/100%10)&&((3*i/2)/1000==(3*i/2)%100/10))/*以 I 为准*/
    {
        cnt++;
        sum+=i+3*i/2;
    }
}
void PressKeyToQuit()
{
    printf("\n Press any key to quit...");
    getch();
    return;
}
void main()
{
    cnt=sum=0;
    clrscr();
    puts(" This program is to find the numbers of SIX and NINE,");
    puts(" which satisfy the formula SIX+SIX+SIX=NINE+NINE,");
    puts(" where S,I,X,N,E stand for digits between 0 and 9. ");
    countvalue();
    printf(" >> The Number of satisfied is %d.\n",cnt);
    printf(" >> The Sum of all SIX and NINE is %d.\n",sum);
    writeDat();
    PressKeyToQuit();
}
writeDat()
{
    FILE *fp;
    fp=fopen("OUT174.DAT","w");
    fprintf(fp,"%d\n%d\n",cnt,sum);
    fclose(fp);
}
```

# 实例 175 数字移位

## 实例说明

已知在文件 in175.dat 中存有若干个（小于 200 个）4 位数字的正整数，函数 readdat() 读取这若干个正整数并存入数组 xx 中。要求编写函数 calvalue()，①求出文件中共有多少个正整数 totnum；②求这些数右移 1 位后，产生的新数中为偶数的数的个数 totcnt，以及满足此条件的这些数（右移前的值）的算术平均值 totpjz，最后调用函数 wriedat() 把所求的结果输出到文件 out175.dat 中。

程序运行效果如图 175-1 所示。



```
This program is to execute Bit Shift Operation.  
>> Now reading datas from file.....Succeeded!  
>> Calculating.....Completed!  
>> Results are:  
>> The total number of data in the file is 185.  
>> The even number of data after bit shift is 93.  
>> The variance of these even numbers is 4932.55  
Press any key to quit....
```

图 175-1 实例 175 程序运行效果

## 程序代码

### 【程序 175】 数字移位

```
#include<stdio.h>  
#include<conio.h>  
#define MAXNUM 200  
int xx[MAXNUM];  
int totnum=0;  
int totcnt=0;  
double totpjz=0.0;  
int readdat(void);  
void wriedat(void);  
void calvalue(void)  
{  
    int i,data;  
    for(i=0;i<MAXNUM;i++)  
    {  
        if(!xx[i]) break;  
        if(xx[i]>0) totnum++;  
        data=xx[i]>>1;  
        if(data%2==0)  
        {  
            totcnt++;  
            totpjz+=xx[i];  
        }  
    }  
    totpjz/=totcnt;  
}
```

```

void PressKeyToQuit()
{
    printf("\n Press any key to quit... ");
    getch();
    return;
}
void main()
{
    int i;
    clrscr();
    puts(" This program is to execute Bit Shift Operation.");
    puts(" >> Now reading datas from file.....Succeeded!");
    puts(" >> Calculating.....Completed!");
    puts(" >> Results are:");
    for(i=0;i<MAXNUM;i++) xx[i]=0;
    if(readdat())
    {
        printf(" Can't open the data file in175.dat!\007\n");
        return;
    }
    calvalue();
    printf(" >> The total number of data in the file is %d.\n",totnum);
    printf(" >> The even number of data after bit shift is %d.\n",totcnt);
    printf(" >> The variance of these even numbers is %.2lf\n",totpjz);
    writedat();
    PressKeyToQuit();
}
int readdat(void)
{
    FILE *fp;
    int i=0;
    if((fp=fopen("in175.dat","r"))==NULL) return 1;
    while(!feof(fp))
        fscanf(fp,"%d",&xx[i++]);
    fclose(fp);
    return 0;
}
void writedat(void)
{
    FILE *fp;
    fp=fopen("out175.dat","w");
    fprintf(fp,"%d\n%d\n%.2lf\n",totnum,totcnt,totpjz);
    fclose(fp);
}

```

## 实例 176 统计最高成绩



### 实例说明

已知学生的记录由学号和学习成绩构成，N 名学生的数据已存入 a 数组中。找出成绩最高的学

生记录(假定最高成绩的记录是惟一的)并通过形参返回。

要求编写函数 `MMM(STU a[], int n, STU *s)` 实现程序的要求，并编写 `sort(STU a[], int n)` 函数对记录数组 `a` 的成绩进行排序(从高到低)，最后调用函数 `READWRITEDAT()` 把结果和排序后的成绩记录输出到文件 `OUT176.DAT` 中。

例如：

```
KS01 87  
KS09 97  
KS11 67
```

则最后输出：

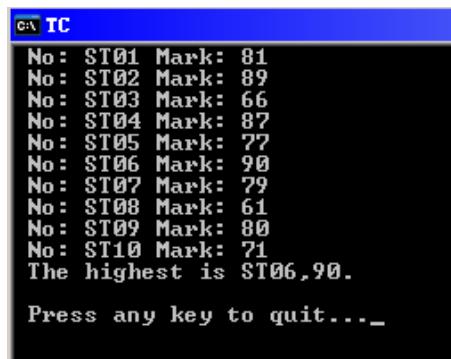
```
KS09 97
```

程序运行效果如图 176-1 所示。

## 程序代码

### 【程序 176】 统计最高成绩

```
#include <stdio.h>  
#include <string.h>  
  
#define N 10  
void readwritedat();  
typedef struct ss{  
char num[10];  
int s;  
}STU;  
mmm(STU a[],int n,STU *s)  
{  
    int i;  
    s->s=a[0].s;  
    for(i=1;i<n;i++)  
        if(a[i].s>s->s)  
            *s=a[i];  
}  
void PressKeyToQuit()  
{  
    printf("\n Press any key to quit...");  
    getch();  
    return;  
}  
void sort(STU a[],int n)  
{  
    STU t;  
    int i,j;  
    for(i=0;i<n-1;i++)  
        for(j=i+1;j<n;j++)  
            if(a[i].s<a[j].s)  
            {  
                strcpy(t.num,a[i].num);  
                t.s=a[i].s;  
                strcpy(a[i].num,a[j].num);  
                a[i].s=a[j].s;  
                strcpy(a[j].num,t.num);  
            }  
}
```



```
c:\TC  
No: ST01 Mark: 81  
No: ST02 Mark: 89  
No: ST03 Mark: 66  
No: ST04 Mark: 87  
No: ST05 Mark: 77  
No: ST06 Mark: 90  
No: ST07 Mark: 79  
No: ST08 Mark: 61  
No: ST09 Mark: 80  
No: ST10 Mark: 71  
The highest is ST06,90.  
Press any key to quit...
```

图 176-1 实例 176 程序运行效果

```

        a[j].s=t.s;
    }

}

main()
{
    STU a[N]={ {"ST01",81}, {"ST02",89}, {"ST03",66}, {"ST04",87}, {"ST05",77},
    {"ST06",90}, {"ST07",79}, {"ST08",61}, {"ST09",80}, {"ST10",71} },m;
    int i;
    clrscr();
    for(i=0;i<N;i++)
        printf(" No: %s Mark: %2d\n",a[i].num,a[i].s);
    mmm(a,10,&m);
    printf(" The highest is %s,%d.\n",m.num,m.s);
    readwritedat();
    PressKeyToQuit();
}
void readwritedat()
{
    FILE *rf,*wf;
    STU a[100],m;
    int i;
    rf=fopen("in176.dat","r");
    wf=fopen("out176.dat","w");
    for(i=0;i<100;i++)
        fscanf(rf,"%s%d",a[i].num,&a[i].s);
    mmm(a,100,&m);
    fprintf(wf,"the top: %s,%d\n",m.num,m.s);
    sort(a,100);
    for(i=0;i<100;i++)
        fprintf(wf,"%s,%d\n",a[i].num,a[i].s);
    fclose(rf);
    fclose(wf);
}

```

# 07

## 第七部分 游戏篇

### 精彩导读

- 商人过河游戏
- 吃数游戏
- 打字训练游戏
- 迷宫探险游戏
- 迷你撞球游戏
- 模拟扫雷游戏
- 推箱子游戏
- 五子棋游戏

# 实例 177 商人过河游戏

## 实例说明

有 3 个商人带者 3 个随从和货物过河，船每次最多只能载两个人，由他们自己划行，并且如何乘船渡河的大权由商人们掌握。要求保证在过河期间的任一岸上商人的人数要大于或等于随从的人数，否则随从会杀死商人抢走货物。设计一个符合上述要求的商人过河的简单游戏。程序运行效果如图 177-1 和图 177-2 所示。

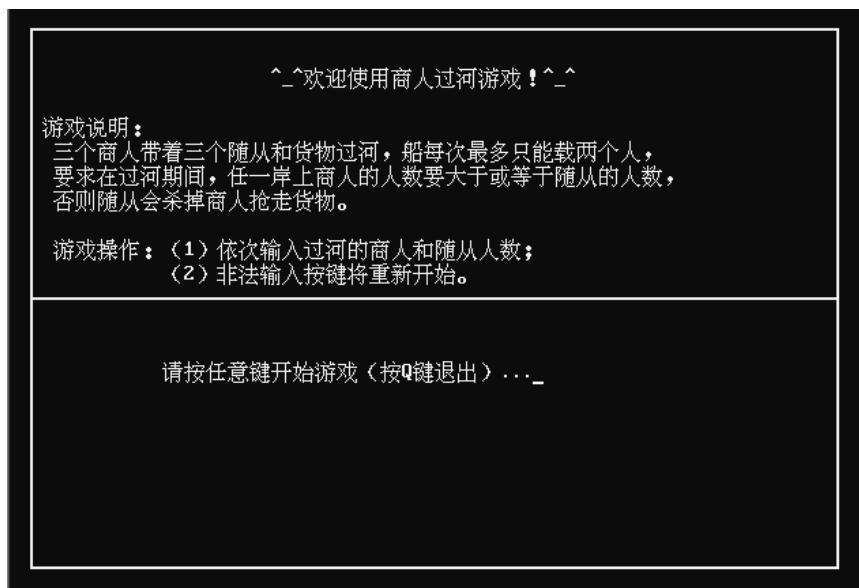


图 177-1 商人过河游戏运行的初始界面

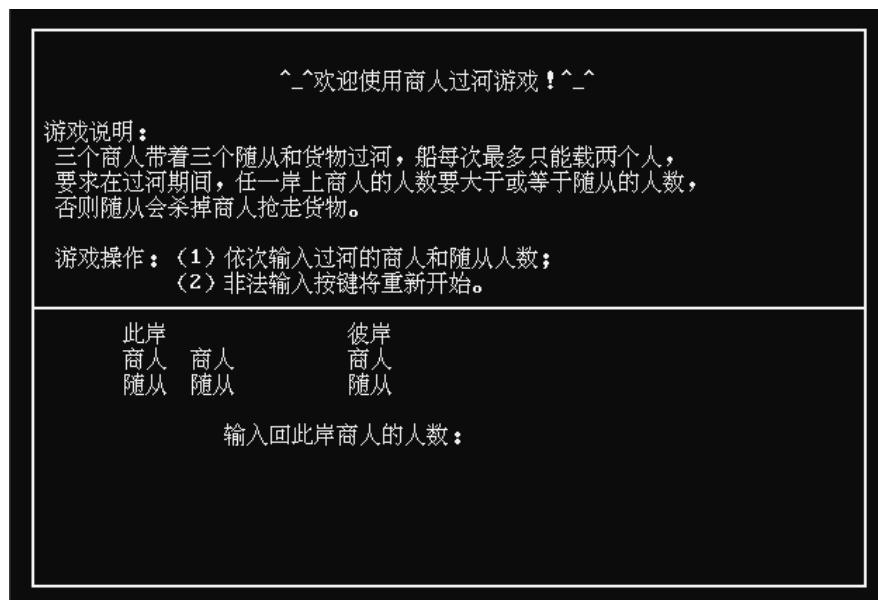


图 177-2 商人过河游戏的输入界面



## 实例解析

用两个数组来表示两岸的商人和随从，其中“**A**”表示商人，“**B**”表示随从，数组 **a** 表示此岸，数组 **b** 表示彼岸；通过输入过河的人数，来改变两个数组中的“**A**”和“**B**”的个数，再用文本输出两岸的情况。

游戏规则：①依次输入过河的商人和随从人数；②非法输入按键将重新开始游戏（例如一次输入的人数不得多于 2 个，过河的人数不超过 2 个，不得输入负数，字母等）；③游戏开始，或重新开始时，可输入 Q 或 q 键退出游戏。

程序设计要点。

(1) 为提供友好的游戏界面，本程序虽然在文本模式下实现，但是提供了汉字界面，游戏说明和规则，操作提示均采用汉字显示，为此，游戏必须在 UCDOS 等汉字环境下运行 (UCDOS 程序见光盘，具体说明参考 UCDOS 下的 readme.txt 文件的说明)。

(2) 程序算法主要是根据规则判断数组 **a** 和 **b** 的状态是否满足要求，若不满足要求，输出提示信息，重新开始游戏；若满足要求，继续显示状态，输入人数。

(3) **helpf** 函数通过调用 **dwframe** 函数来进行游戏窗口的定义，背景色和文本色的设置，界面框架的绘制，并显示游戏提示和规则说明。

(4) **printcase** 函数将数组 **a** 和数组 **b** 所代表的状态进行显示，并判断若商人和随从全部到达彼岸，则成功过河，游戏完成。

(5) 主函数 **main** 完成调用 **helpf** 函数，判断开始或重新开始时是否输入 Q 或 q 键，若是，则恢复背景色和文本色的设置，退出游戏；若不是，则在 **while** 循环中提示输入，判断情况，并做相应处理。

//本实例源码参见光盘



## 归纳注释

为了找到求解这类问题的规律性，建立数学模型，用以解决更为广泛的问题，对游戏的数学模型建立如下：此问题可视为一个状态转移问题，每一步就是一次渡河，每次渡河就是一次状态转移。用三维变量 ( $X$ ,  $Y$ ,  $Z$ ) 表示状态： $X$ ——商人数， $Y$ ——随从人数， $Z$ ——船的状态，其中  $Z$  的取值范围 {0, 1}， $X$ ,  $Y$  的取值范围 {0, 1, 2, 3}，那么安全状态可表示为  $\{Y=X; X=1, 2\}$ ，即 (3,3,1) (3,2,1) (3,1,1) (2,2,1) (3,0,1) (0,3,1) (0,2,1) (1,1,1) (0,1,1) (3,2,0) (3,1,0) (2,2,0) (3,0,0) (0,3,0) (0,2,0) (1,1,0) (0,1,0) (0,0,0)。用线表示决策，坐标表示状态，此线将决策前后两个状态相连，这样问题变成求由 (3,3,1) 变到 (0,0,0) 状态变化的一条道路。

另外，状态转移时要满足一定的规则：①  $Z$  从 1 变为 0 与从 0 变为 1 交替进行；②当  $Z$  从 1 变为 0 时，即船从此岸到对岸，此岸人数减少 1 或 2 个；即  $(x,y,1) \rightarrow (u,v,0)$  时， $u \leq x$ ,  $v \leq y$ ,  $u+v=x+y-1$  或者  $u+v=x+y-2$ ；③当  $Z$  从 0 变为 1 时，即船从对岸到此岸，此岸人数增加 1 或 2 个，即  $(x,y,0) \rightarrow (u,v,1)$  时， $u \geq x$ ,  $v \geq y$ ,  $u+v=x+y+1$  或者  $u+v=x+y+2$ ；④不重复已出现过的状态，如 (3,3,1)  $\rightarrow$  (3,1,0)  $\rightarrow$  (3,3,1)。本问题的求解线路如图 177-3 所示。

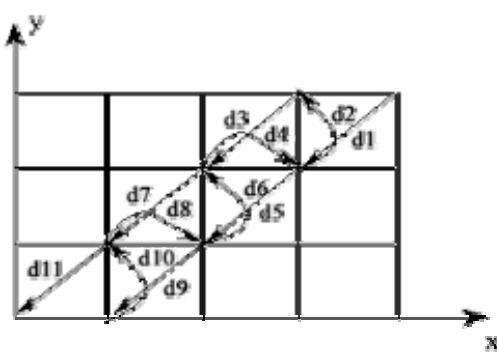


图 177-3 状态转移解法

# 实例 178 吃数游戏

## 实例说明

路边吃数游戏主要是要求在宫殿中找出几个数之和是 7 的数并把它们吃掉，再走到 7 的位置就获胜了。该游戏是比较简单的游戏。运行环境是在文本模式下进行的。程序运行效果如图 178-1 和图 178-2 所示。



图 178-1 路边吃数游戏开始界面



图 178-2 路边吃数游戏打贏游戏的界面

## 实例解析

程序设计中最重要的思想是，先在屏幕上画一个宫殿（包含墙壁、通道和数字以及小人），再在循环中判断键盘输入，如果输入为上下左右 4 个键，则根据宫殿的地图情况依次处理（碰到墙壁不动，碰到通道按方向前进，碰到数字吃掉并前进），吃掉数字时，将所吃数字计入总和。当所

到位置为指定位置且所吃数字总和满足要求时，就赢了游戏，否则，所吃数字不满足要求，则输掉游戏，提示重来。

K 是二维数组，用于存储宫殿的地图，0 代表该位置为墙壁，1 代表该位置为通道，大于 1 的数字表示该位置为通道且有某个数可以吃。

int en(int \*b,int n)函数用于判断数组 b 中的元素是否与 n 相同，若有相同的元素，则返回 0，否则返回 1。

fu(int \*b)函数用于将数组 b 中的 20 个元素置初值 0。

lostgame(int num)函数用于在固定位置输出 num 的值，并提示已经输掉游戏了，按任意键重新开始游戏，按任意键后，将其输出的两行提示信息用空格覆盖。

wingame()函数用于输出已经赢得游戏的提示信息，按任意键退出游戏，在按任意键后，恢复文本模式的前景色和背景色的设置，并清屏，退出程序。

主函数 main 中要完成的内容是初始化，画地图，判断按键输入，并做相应处理。

//本实例源码参见光盘

## 归纳注释

在设计游戏时，要规划好程序的结构是怎样的，如何处理各种消息和输入，并尽可能的将相同的代码模块函数化，这样可以使程序简明易懂，在调试和修改时也会很方便。

# 实例 179 解救人质游戏

## 实例说明

本实例实现解救人质游戏的设计。游戏要求在一定的时间内走到右下角的小人处解救人质。在每一行要往下走时都有一定的条件，碰到该条件，就可以实现往下走一行，否则只能待在本行。程序运行效果如图 179-1~图 179-5 所示。

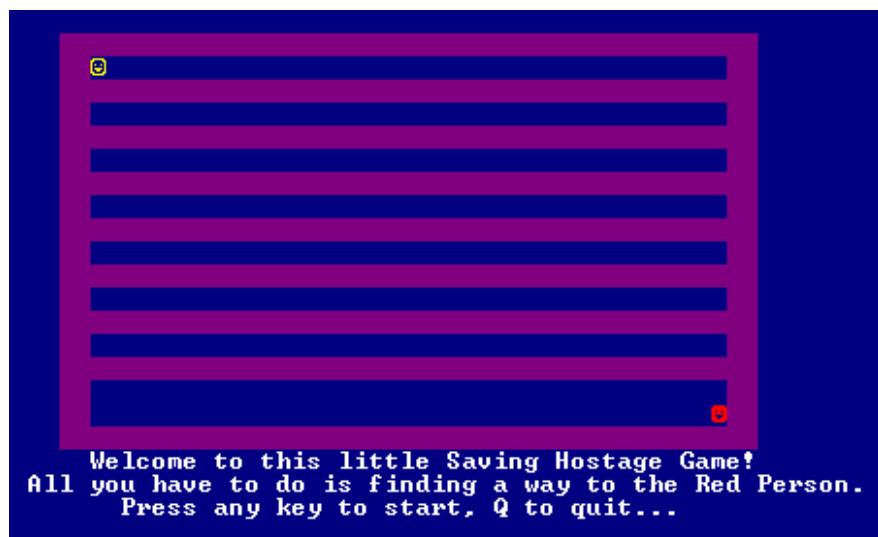


图 179-1 解救人质游戏运行初始界面

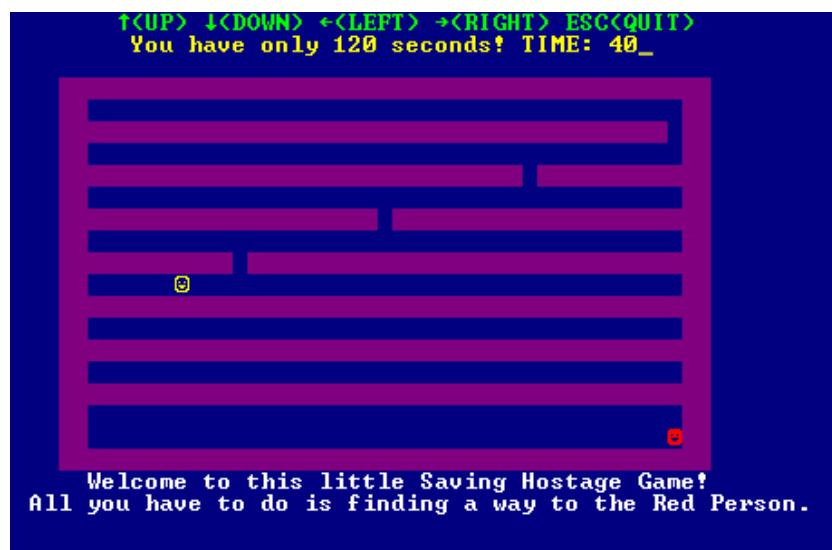


图 179-2 寻找途径界面

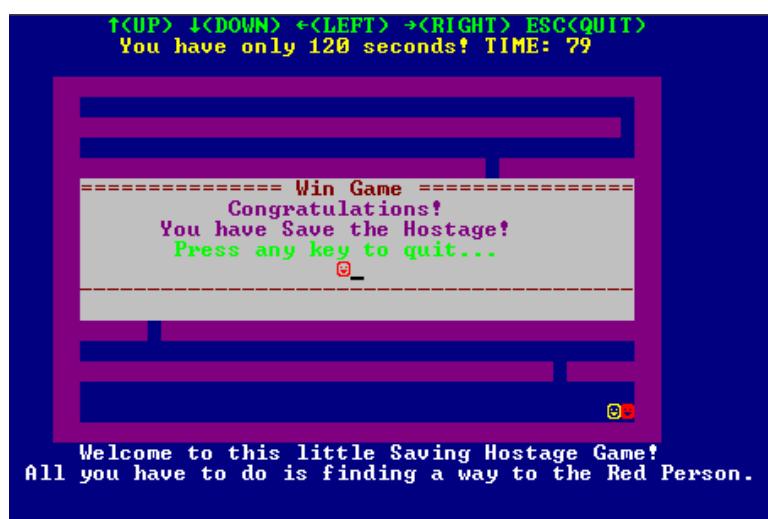


图 179-3 完成任务界面

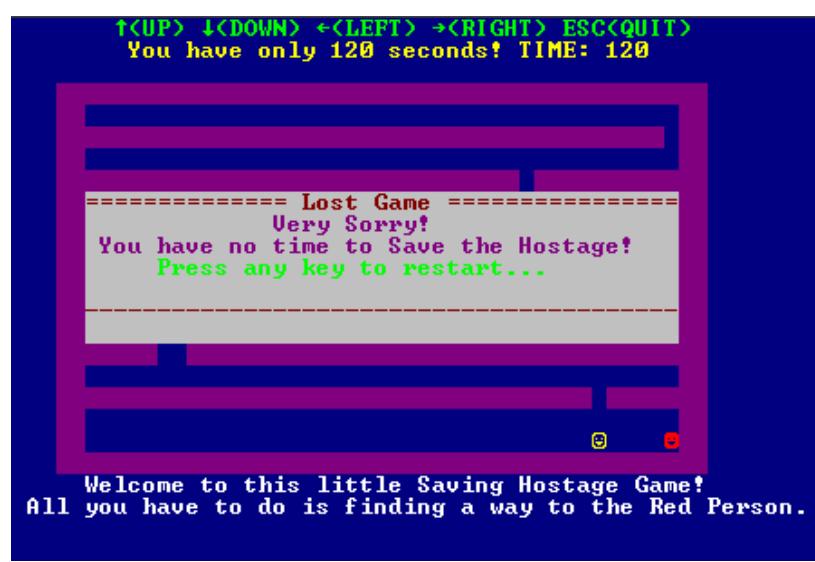


图 179-4 输掉游戏界面

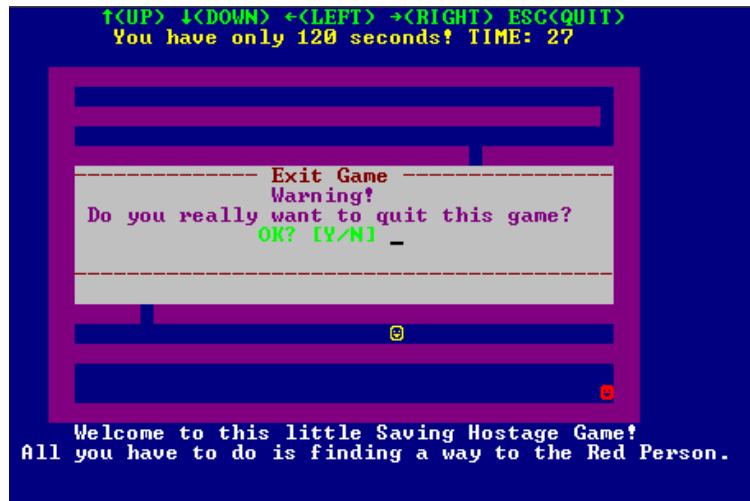


图 179-5 在游戏进行中按 ESC 键退出的确认界面

## 实例解析

游戏设计思路：采用文本模式进行，先在屏幕上显示一座 7 层的大楼示意图，人质在底层，解救人（玩家）从顶楼开始搜索。开始时，提示游戏规则，并按任意键开始游戏，按 Q 键退出游戏。

开始游戏时，提示按上下左右键进行移动，ESC 键退出游戏。每一层的楼梯位置未知，要求玩家自行查找。提示时间为 120s（可通过程序中的宏 MAXTIME 定义修改），并实时显示所用时间。时间到而未完成任务，则显示小窗口，提示失败，按任意键重新开始游戏。按时到达底层红色小人那里则解救成功，显示小窗口提示获胜，按任意键退出游戏。在游戏已经开始后，按 ESC 键要退出游戏时，显示小窗口提示确认是否要退出，按 Y 键退出，按 N 键则回到游戏初始界面。

程序中，全局变量 t 为定时计数变量，x, y 为当前玩家的坐标（文本模式下的行和列号），以便于初始化。程序由 initScreen, littlewin, JudgeKey 和 quitgame 等函数组成。其中，JudgeKey 完成程序的所有功能，包括调用 initScreen 等函数。而主函数 main 中，仅有一个语句，即调用 JudgeKey 函数。之所以这样，可以使程序简明易懂，也可以在 JudgeKey 中递归调用 JudgeKey，避免了采用 goto 语句造成语序的混乱，降低程序的可读性。

initScreen 函数用于初始化屏幕，并输出提示信息。包括设置屏幕背景色和前景色，画大楼示意图，画人质和解救人，输出游戏提示，按 Q 键则调用 quitgame 退出游戏，否则按其他任意键后，初始化计时变量 t，提示操作键和时间限制。

quitgame (int Conf) 函数用于退出游戏，当 Conf=0 时，恢复屏幕背景色和前景色的设置，清屏，退出游戏。当 Conf=1 时，需要确认信息，此时显示小窗口，并提示是否确认退出，按 Y 则恢复屏幕设置清屏退出，按 N 则调用 JudgeKey 函数，开始重新游戏。

littlewin (int WinGam) 函数用于显示小窗口，提示赢了游戏 (WinGam=1) 或输了游戏 (WinGam=0)，赢了则提示按任意键退出游戏，输了则提示按任意键重新开始游戏。

JudgeKey 函数用于判断输入的按键，并进行相应操作（仅对上下左右和 ESC 做处理）。首先调用 initScreen 初始化屏幕，接着在循环中不断判断按键，对其做相应操作。并有一个计时循环，用于定时显示所用的时间，判断如果到达时间，则调用 littlewin (0)，说明已经输掉游戏，按任意键，重新调用 JudgeKey 开始新游戏。如果按键为 ESC，则调用 quitgame (1)，显示提示信息，根据回应，做相应处理。如果按键为左右键，则玩家位置做相应变化，并判断是否已经解救了人质，如是则跳出循环，调用 littlewin (1)，说明赢了，并调用 quitgame (0)，退出游戏。如果为上下键，则根据一定的条件判断是否是楼梯，是的话，则打开楼梯。每个按键，都有提示音，通

过 PC 喇叭发声。

//本实例源码参见光盘



## 归纳注释

程序主要是调用 window(int x1, int y1, int x2, int y2) (其中参数 (x1,y1) 和 (x2,y2) 为窗口的左上角和右下角坐标, 即行列号) 来定义一个文本窗口, 用函数 textbackground 和 textcolor 分别设定背景色和文本色, 并用 clrscr 函数来显示这个窗口。此时就可用 cprintf 和 cputs 函数在该窗口中进行输出。用 gotoxy(x,y) 函数可以在该窗口的指定位置输出, 注意此时 (x,y) 坐标是以该窗口的左上角点为参考原点 (1,1), 即为相对坐标。

# 实例 180 打字训练游戏



## 实例说明

本实例实现打字训练游戏的设计。本打字训练游戏可以进行英文、其他字符和全部字符的打字训练, 每次训练 100 个字符, 并进行打字时间、打字速度、正确率的统计显示。程序欢迎界面为图形模式, 打字界面为文本模式, 模仿 Turbo C 2.0 程序的界面设计 (没有完成菜单功能、快捷键等设计)。程序运行效果如图 180-1~图 180-4 所示。



图 180-1 打字训练游戏运行的欢迎界面

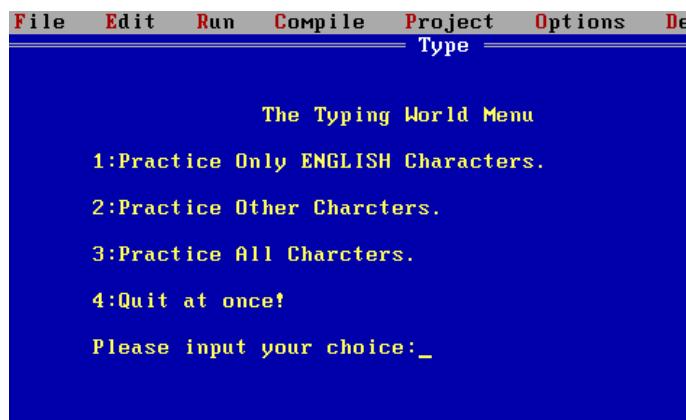


图 180-2 打字训练游戏的主菜单界面

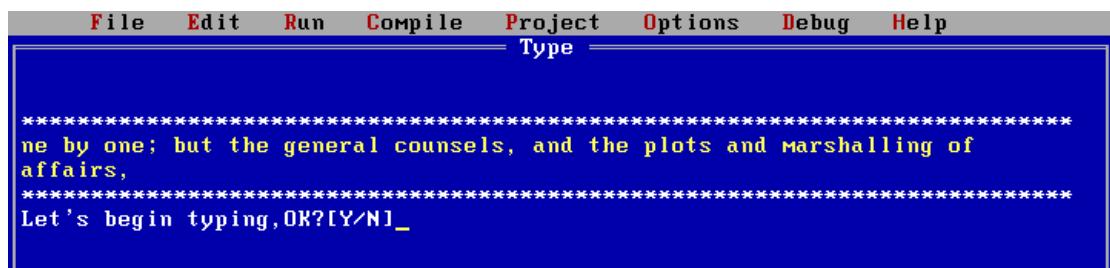


图 180-3 打字训练游戏的英文打字界面

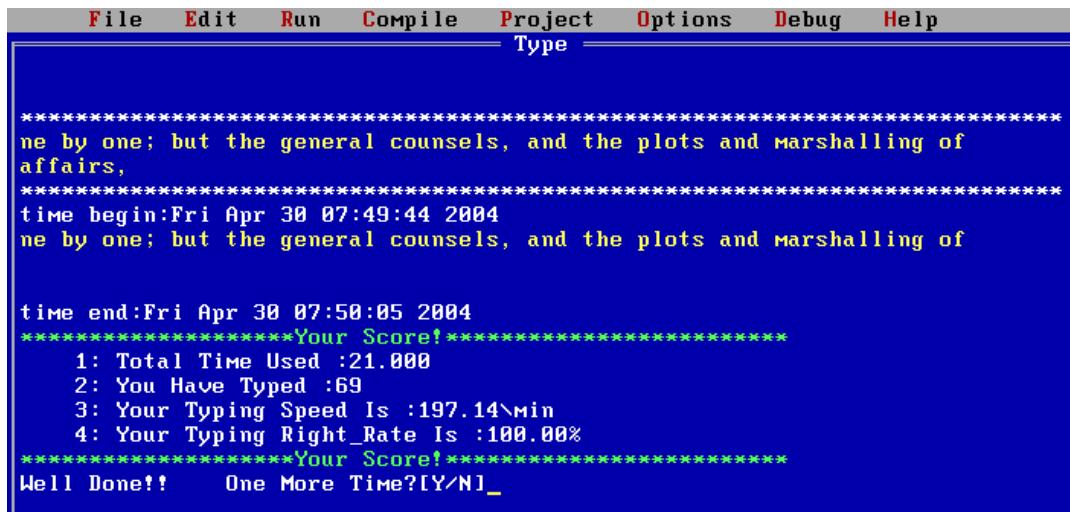


图 180-4 打字训练游戏的打字结果统计界面



图 180-5 打字训练游戏的退出确认界面

## 实例解析

程序采用模块化设计模式。开始时，采用图形方式显示欢迎界面，并采用逐一显示字符的方式，每显示一个字符，发出一次声音（PC 的喇叭声）。欢迎界面显示完之后，按任意键进入主菜单。主菜单和打字界面为文本模式，仿 TC 界面。主菜单采用文字显示，要求选择 1~4 按键，1~3 分别为英文、其他字符和全部字符的打字练习。在选择了 1~3 之后，就相应的打开 english.dat、others.dat 和 typeall.dat 文件，从中随机选取 MAXCHAR 个字符进行打字训练。打字训练结束，显示统计信息，提示是否继续打字，是则回到主菜单，否则退出。在退出时，界面采用小窗口显示

提示信息，按 Y 则确认退出，按 N 则返回主菜单。

程序中的字符数组 `string` 用于存储从文件中选取的字符，其大小可以通过程序中的宏 `MAXCHAR` 设置。函数 `LittleWin` 用于显示退出确认窗口，函数 `quitgame` 用于退出程序，函数 `Welcome` 完成图形方式下的欢迎界面的显示，函数 `drawframe` 完成文本模式下的框架的显示，函数 `Frame` 用于显示主菜单，函数 `GetCharacter` 用于根据选择从相应文件中提取字符，函数 `Typing` 用于进行打字训练。程序中的主函数 `main` 调用 `Welcome` 函数进行欢迎界面显示和调用 `Frame` 函数显示主菜单。

`LittleWin (int WinType)` 函数用于根据 `WinType` 进行小窗口的显示，本程序中仅实现了 `WinType=1` 的情况，此时显示的是退出程序的确认窗口，显示此操作将退出程序，按 Y 则返回 1，按 N 则返回 0。

`quitgame()` 函数用于退出程序，先调用 `LittleWin(1)` 进行确认，若返回 1，则恢复背景色和前景色的设置，清屏，退出程序，若返回 0，则调用 `Frame()` 函数，返回主菜单。

`Welcome()` 函数用于显示欢迎画面，该画面是图形模式下的。首先初始化图形方式，接着设置背景色和前景色，并画一系列不同填充图案的圆，同时，每画一个圆，就发出一阵声音。接着逐个字符的输出欢迎信息，并发出声音，显示完毕，要求按任意键进入主菜单。

`drawframe()` 函数用于显示类似 TC 的程序界面，主要是在文本模式的屏幕（25 行 80 列）的第一行设置一个 `window`，设置相应的背景色和前景色，并输出菜单，在第 25 行设置一个同样的 `window`，输出提示文本，在剩下的屏幕区域设置一个 `window`，输出边框字符。

`Frame()` 函数用于显示主菜单。该函数首先调用 `drawframe` 函数显示边框，再在边框的内部设置一个窗口，用于显示文本和打字，然后输出主菜单，供选择。

`GetCharacter()` 函数用于根据用户的选择（1~4）选取字符，供打字联系。在完成字符选择后，调用 `Typing` 函数进行打字训练。

`Typing()` 函数用于接受用户的打字输入，并统计相关的数据，打字完成，显示统计数据，并询问是否继续，若是则调用 `Frame` 函数返回主菜单，若不是则调用 `quitgame` 函数。

//本实例源码参见光盘



## 归纳注释

根据需要，可以在 `english`、`others`、`typeall` 3 个 `dat` 函数中修改文本，进行相应的打字训练。

本程序只是简单的打字训练程序，功能还不是很完善。比如菜单功能，仅仅参考 TC 界面，用户可以自行添加功能，并调整菜单的名称；快捷键的功能也没有完成。另外，程序的打字训练功能也不够完善，可参考 `tt` 程序添加相应的功能。

开始的欢迎界面是图形方式，必须全屏显示。如果在 `main` 主函数中，不调用 `Welcome` 显示欢迎信息，或者将欢迎信息在文本方式下显示，则该程序可以在 windows 操作系统下进行窗口模式的显示。

# 实例 181 双人竞走游戏



## 实例说明

本实例实现双人竞走游戏的设计。双人竞走游戏是一个双人玩的策略游戏，游戏有两个点，

第一个游戏者可以按 R, G, F, D 控制第一个点的走向，第二个游戏者可以按 4 个光标键控制第二个点的走向，谁先碰到边界或已走的路线便输。其他功能键：F1 查看帮助信息；F2 设定游戏速度等级，0 为最快，1 为正常，2 最慢；F3 设定游戏者 1 的颜色，有 15 种颜色可以选择；F4 设定游戏者 2 的颜色；F5 继续游戏。程序运行效果如图 181-1~图 181-4 所示。



图 181-1 双人竞走游戏的初始界面



图 181-2 游戏开始界面

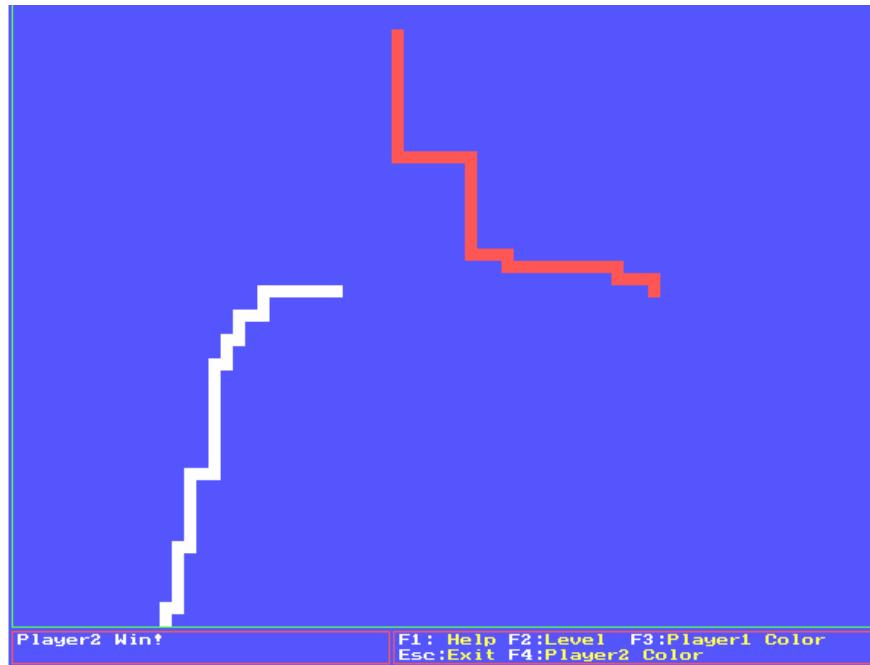


图 181-3 玩家 2 赢得界面

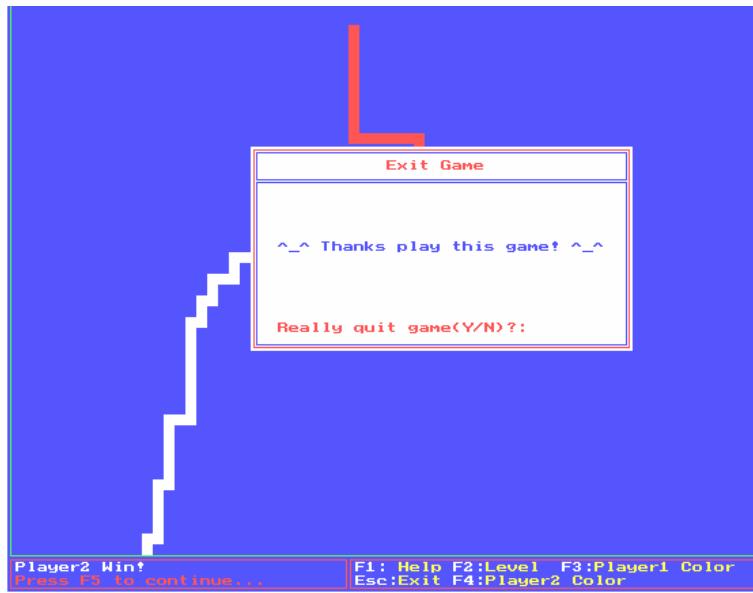


图 181-4 退出游戏的确认界面

## 实例解析

结构 information 定义画方框的坐标 X, Y, 以及颜色 Color。Draw 为是否已走过, Draw 为 1 已经走过, 为 0 则未走过。

playerxy 结构用于记录游戏者所在位置, X 与 Y 就是全局数组 coordinate[][] 的两个下标。

全局数组 INFOR coordinate[80][60] 是记录画每个框的坐标及颜色以及是否已走过。

全局变量 Timeout=1 用于限制游戏的快慢, 可用 Set\_Level() 设定。

全局变量 size 定义画框的大小, 单位为像素, maxX 和 maxY 为数组 coordinate 下标的最大值。

全局变量 BackColor 为游戏背景色, Player1Color 与 Player2Color 为游戏者默认颜色, 可调用 Set\_Color() 函数设定。

函数 InitialGraphics 用于初始化图形模式, GoodBye 用于在退出游戏显示提示信息, 不能初始化图形界面时才会调用此函数。函数 InitFace 用于初始化游戏界面。函数 InitCoordinate 用于初始化全局数组 coordinate[][]。函数 InitPlayerPlace 用于初始化游戏者开始位置。Drawbar(CurrentCoor player,int who) 为画框函数, player 为画框位置, who 判断是哪一个游戏者。函数 HelpMassage 提供帮助信息, 按 F1 调用此函数。函数 GetKey 用于获取按键, 如果低 8 位非 0 则为 ASCII 码, 如为 0 则为控制键。函数 Initfx 用于初始化开始方向, 只有两个游戏者都按下了方向键才开始游戏。ManageMove 为最主要函数, 控制画框方向以及判断是否已输。Set\_Level 函数用于设定游戏等级。Set\_Color (int who) 函数用于设置游戏者颜色, 其实就是设置全局变量 Player2Color 与 Player1Color, 根据 who 来判断是哪一个游戏者。int Exit\_Game (void) 是确认退出函数, 返回 1 (退出) 与 0 (继续游戏)。GameManage 为游戏控制函数, 完成一系列初始化操作, 最后调用 ManageMove 函数, 完后便显示谁胜及一些快捷键的功能调用。程序中的主函数 main 调用 InitialGraphics() 函数进行图形的初始化, 并调用 GameManage() 函数对游戏进行控制, 最后调用 closegraph() 函数关闭图形模式, 退出游戏。

//本实例源码参见光盘



## 归纳注释

由于程序是在图形方式显示的, 需要 EGAVGA.BGI 文件才能运行, 该文件在 Turboc2 目录下

可找到, 将 EGAVGA.BGI 文件与可执行文件放在相同目录下, 即可运行游戏。

本程序是图形方式下运行的游戏, 通过该程序的设计, 可以加深读者对图形方式下程序的理解, 进一步掌握有关 DOS 图形方式编程的技巧, 为开发大型的游戏程序打下基础。

对于在文本模式下的显示提示、确认等信息的窗口, 也可以在图形模式下实现。

该游戏的玩法虽然比较简单, 但是功能齐全, 可以作为 DOS 下编写 C 语言游戏程序和其他软件的参考范例。

## 实例 182 迷宫探险游戏

### 实例说明

本实例实现迷宫探险游戏的设计。迷宫探险游戏主要是完成小人从左上角闯过迷宫到达右下角的路径探险。开始时, 按 M 键进入人工控制模式; 按其他键则由计算机演示自动走迷宫的功能。在人工模式时, 按 w、s、a、d 分别代表上、下、左、右方向, 按 Q 键可退出游戏。程序运行结果如图 182-1~图 182-3 所示。

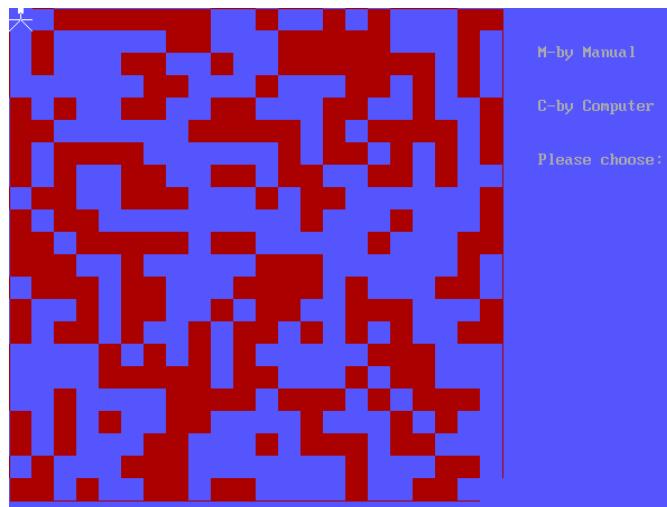


图 182-1 迷宫探险游戏初始界面

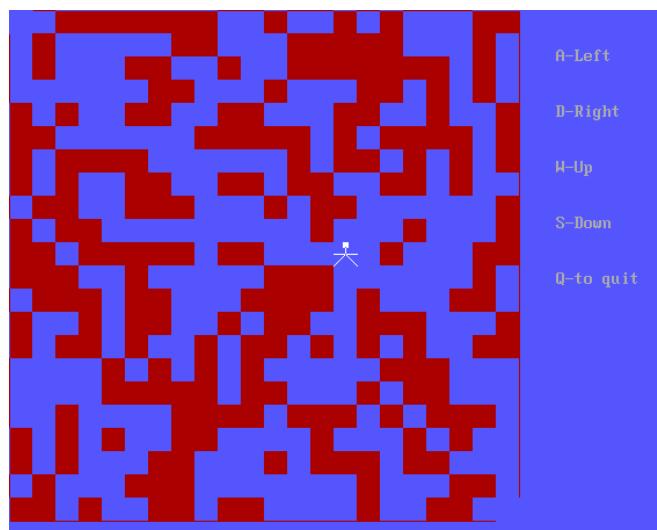


图 182-2 开始探险游戏界面

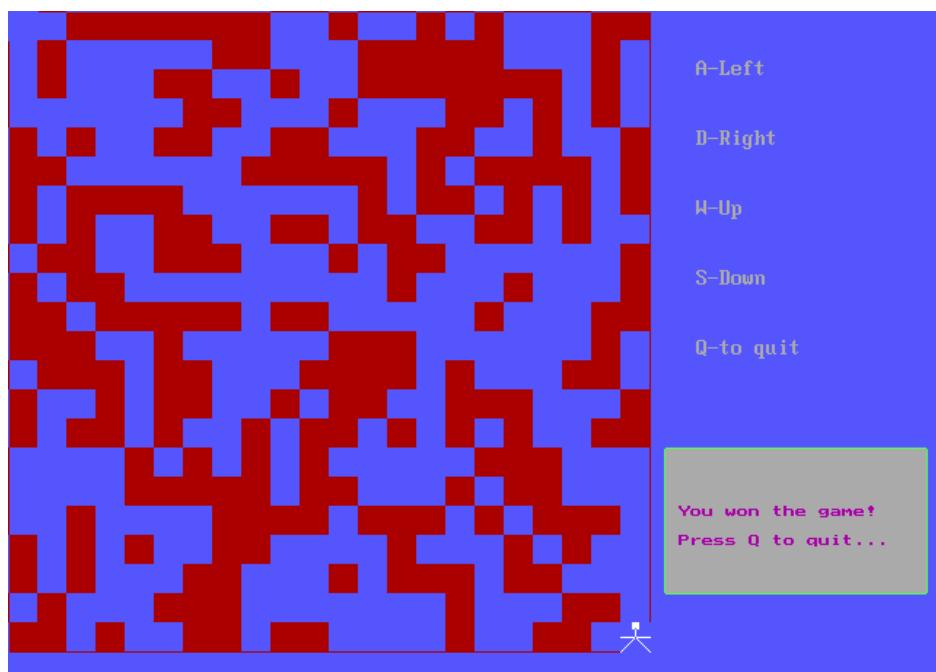


图 182-3 完成探险任务界面

## 实例解析

一个走迷宫的诀窍：顺着墙沿一侧走（一直沿左侧或一直沿右侧）。本程序实现了这一思想，在由计算机演示自动走迷宫时，小人一直沿左侧走碰到不能走，则返回，一直到到达出口为止。程序中的迷宫是随机生成的，没有固定的模式。每次都不相同。

程序设计思路如下。

宏定义 M、N 用于设定迷宫的大小（长和宽），程序中定义为  $22 \times 22$  方块。

全局变量数组 bg[M][N] 用于存储迷宫的信息，1 代表墙壁，0 代表通道。

程序主要控制在 main 函数中。首先，main 函数对一些全局变量进行初始化，接着调用 makebg 随机生成迷宫的地图。然后初始化图形方式，并在画迷宫的旁边输出提示信息（是文本模式下的 printf，所以其坐标是行号和列号），接着设定写模式为 XOR\_PUT，设置背景色和前景色，设置填充模式，调用 drawbg 进行迷宫地图的绘制，调用 drawman 在指定位置画一个小人，然后从键盘读入一个字符，若是 M 键则进行人工控制，其他键则是计算机自动演示。

人工控制时，对输入的按键进行判断，有效的键是 W、S、A、D 为方向键，Q 为退出键。若为方向键，则根据迷宫地图进行判断，是通道前进一步，否则，只能在当前位置。若走出迷宫，则显示提示信息，按任意键退出。

在计算机演示模式下，按靠墙的左侧一直走，进行探险，如果碰到“死胡同”则返回，这样可以一直走到出口，按任意键退出游戏。

函数 rect (int x0,int y0,int x1,int y1) 用于绘制左上角坐标为 (x0,y0)，右下角坐标为 (x1,y1) 的实心矩形（采用一个像素一条直线的方式填充起来）。

makebg 函数用于随机生成代表迷宫地图的数组。

drawbg 函数用于根据 bg 数组中的数据绘制迷宫地图。

drawman (int x,int y,int len) 用于在指定的坐标 (x,y) 处绘制小人。

// 本实例源码参见光盘



## 归纳注释

本程序由于代码不多，所以控制部分在 main 主函数中完成。也可以将许多代码从 main 函数中剥离出来，做成函数进行调用。比如，初始化，可以用函数实现，然后调用 makebg 产生迷宫地图，调用 drawbg 画迷宫地图，调用函数（比如 OutInfo）输出提示信息，调用 getinput 函数对输入的按键进行处理，在 getinput 函数中，若输入 M 则调用 ManualCon 函数进行人工控制，若输入 C，则调用 ComputerCon 函数进行计算机走迷宫的操作演示。在 ManualCon 中，对各种输入进行相应的操作，走出迷宫，则调用 quitgame 函数进行退出游戏的提示，按任意键退出游戏。所有这些，实现起来都不会很难，读者可以自行尝试实现。

# 实例 183 迷你撞球游戏



## 实例说明

迷你撞球游戏是根据 DxBall 撞球游戏进行设计的具有简单的撞球功能的游戏。在游戏运行后，根据提示，按任意键开始游戏（Q 键退出游戏）。游戏开始后，用鼠标可以移动挡板，使掉下来的撞球反弹上去，碰到的砖块将消失，这样一直到所有砖块都消失，就赢了游戏。程序运行结果如图 183-1~图 183-3 所示。

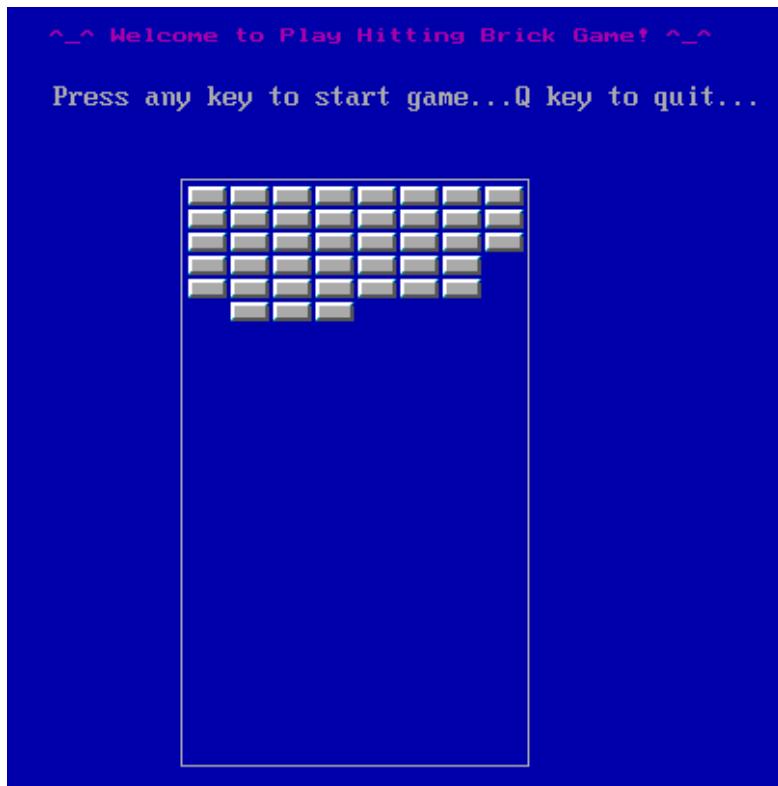


图 183-1 迷你撞球游戏初始界面

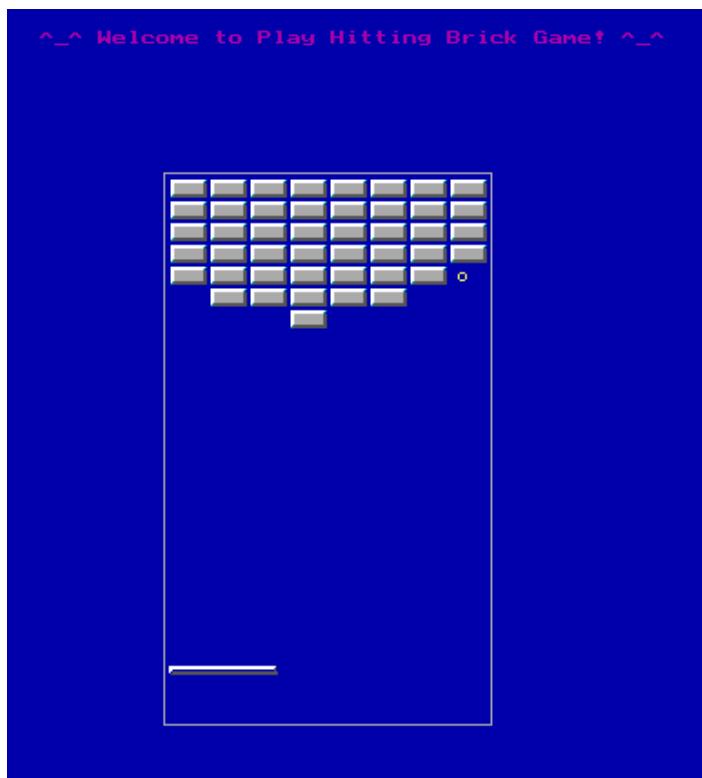


图 183-2 开始撞球的界面

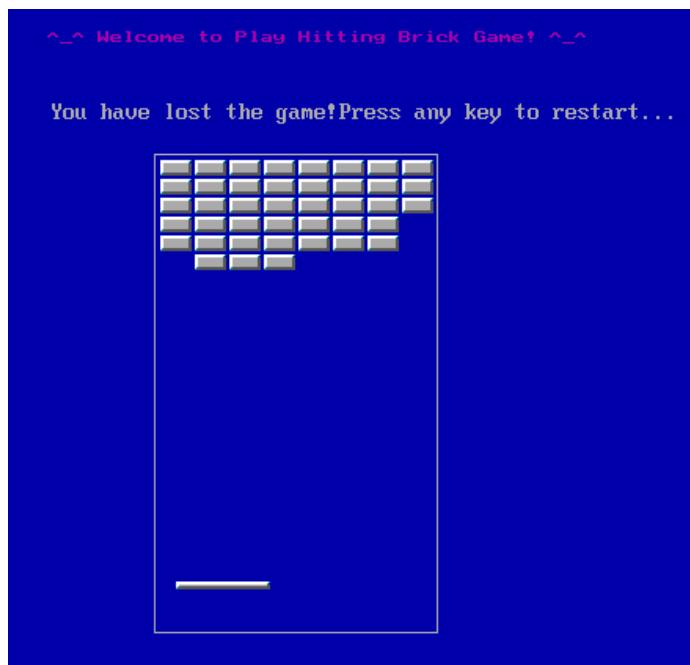


图 183-3 输掉游戏的界面

## 实例解析

联合 REGS regs 为全局变量，用于存储通过 DOS 系统调用返回的寄存器的值（对鼠标的操作和有关信息）。

程序主要在 main 主函数中完成。main 函数首先进行一些变量的初始化，接着初始化图形模式，调用 Msinit 函数对鼠标进行初始化，并调用 Setmouse 函数对鼠标的范围进行设置，绘制砖块，输出提示信息，要求按任意键开始游戏（Q 键则调用 quitgame 函数，关闭图形模式，退出程序）。开始游

戏后，通过 Msread 不断读取鼠标的位置信息，并绘制挡板，判断反弹是否碰到砖块并对圆球的位置进行绘制。当砖块都打完时，玩家胜利，当球没碰到挡板时，则输掉游戏，提示按任意键重新开始。

全局变量 zhuan[5]数组用于保存盘面情况。

quitgame()函数用于关闭图形模式，清屏，退出程序。

wingame()函数用于提示已经赢了游戏，按任意键调用 quitgame 退出游戏。

Msinit()函数用于实现鼠标的初始化。

Setmouse (int left,int right,int top,int bottom)函数用于实现设置鼠标的移动范围。

Msread()函数用于实现鼠标的读取。

Draw(int x,int y,int sizex,int sizey)函数用于在指定(x,y)位置画一个指定大小(长 sizex, 宽 sizey)的矩形块。

Mycircle(int x,int y,intr,int color)函数用于绘制一个圆心在 (x,y)，半径为 r，线条颜色为 color 的红色填充的圆。

//本实例源码参见光盘



### 归纳注释

本实例是比较简单的撞球游戏。当然，更复杂的游戏可以通过设定更大、更多，种类更齐全的砖块和挡板以及小球的各种变化模式来进行，而且可以设定更多的关卡模式，对游戏的功能进行扩充。另外，还可以设计在小球碰到某些砖块时，掉下一些东西（比如可以使挡板属性变化，改变游戏速度，改变砖块属性等）使游戏更加好玩。感兴趣的读者可以参考 DxBall 游戏自行编写相应功能。

## 实例 184 模拟扫雷游戏



### 实例说明

模拟扫雷游戏是模仿 Windows 自带的扫雷游戏而设计的。功能与 Windows 的扫雷游戏相当。通过在图形模式下绘制各种窗口，操作控制鼠标来进行游戏。可以选初级、中级、高级 3 种游戏模式，也可以重新开始游戏，可以查看最高分记录，也可以查看帮助信息等。程序运行结果如图 184-1~图 184-7 所示。

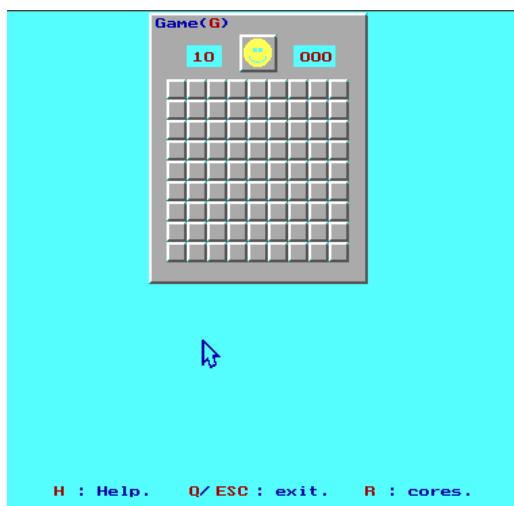


图 184-1 模拟扫雷游戏的初始界面

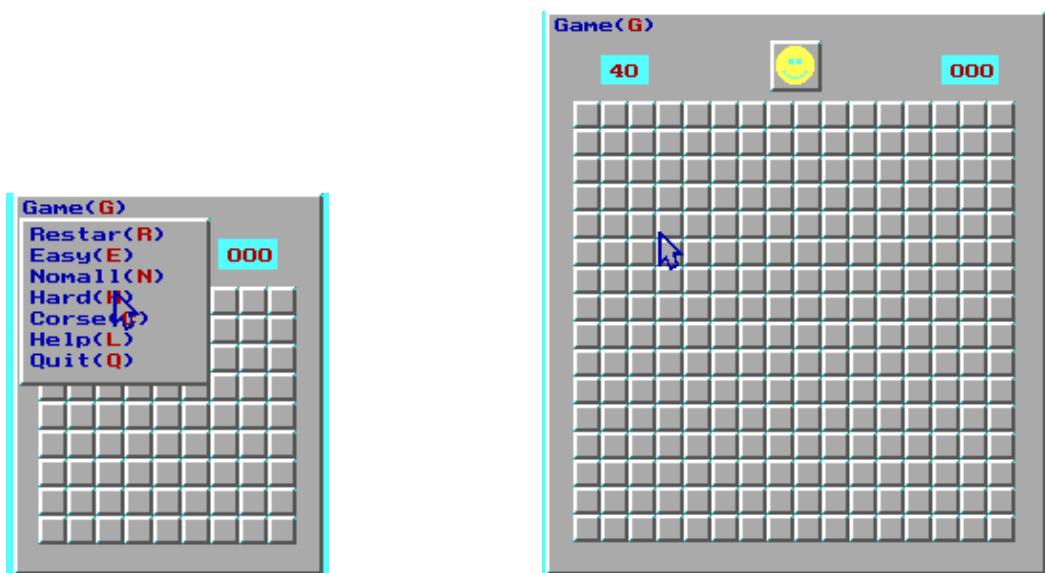


图 184-2 选择菜单中的 Normal 模式时进入中级游戏的界面

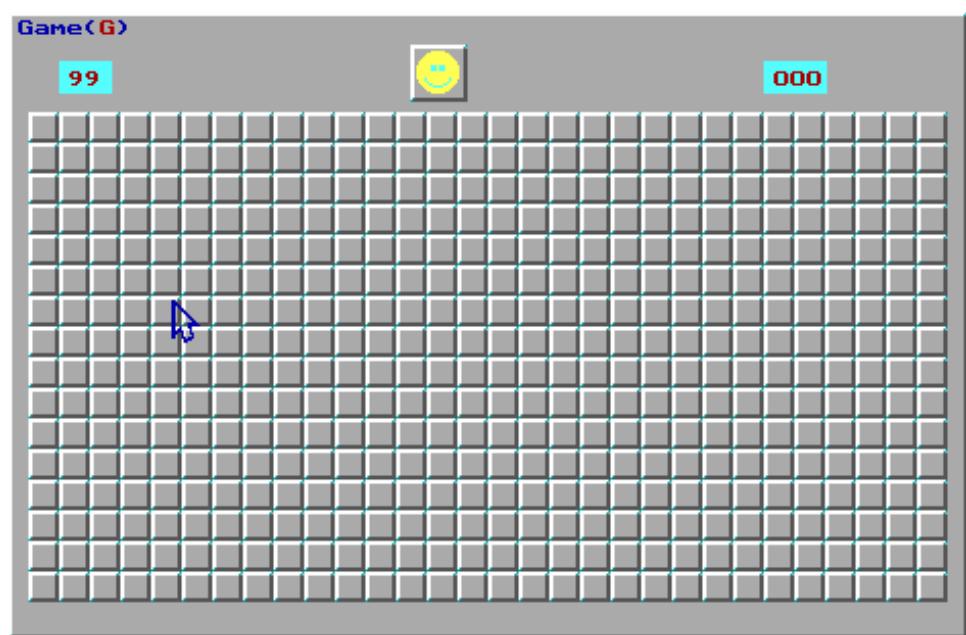


图 184-3 高级游戏的界面

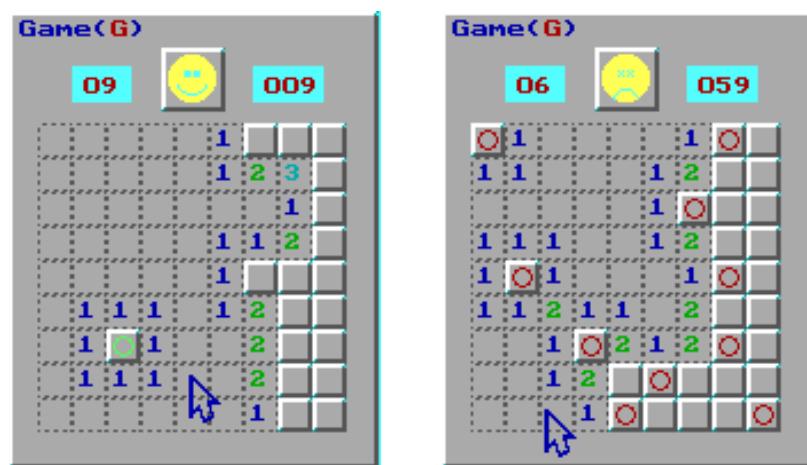


图 184-4 初级模式下的扫雷和碰到地雷的界面

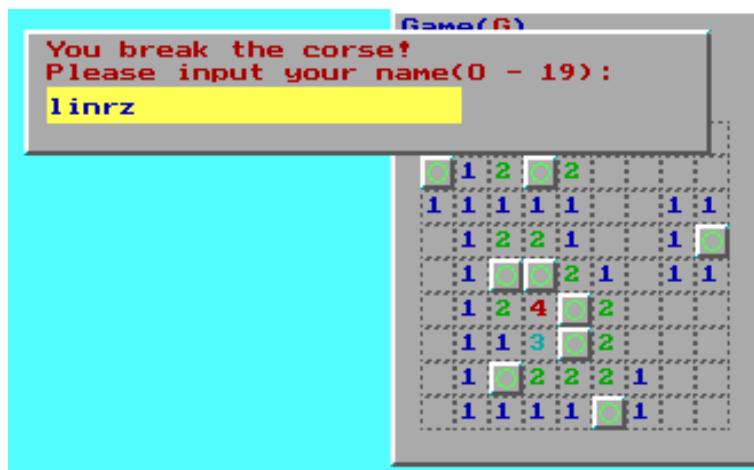


图 184-5 初级模式下的扫雷成功的界面

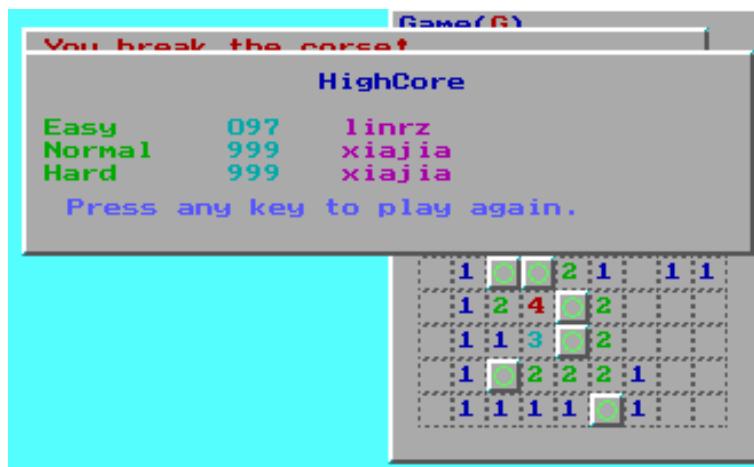


图 184-6 显示最高分数的界面

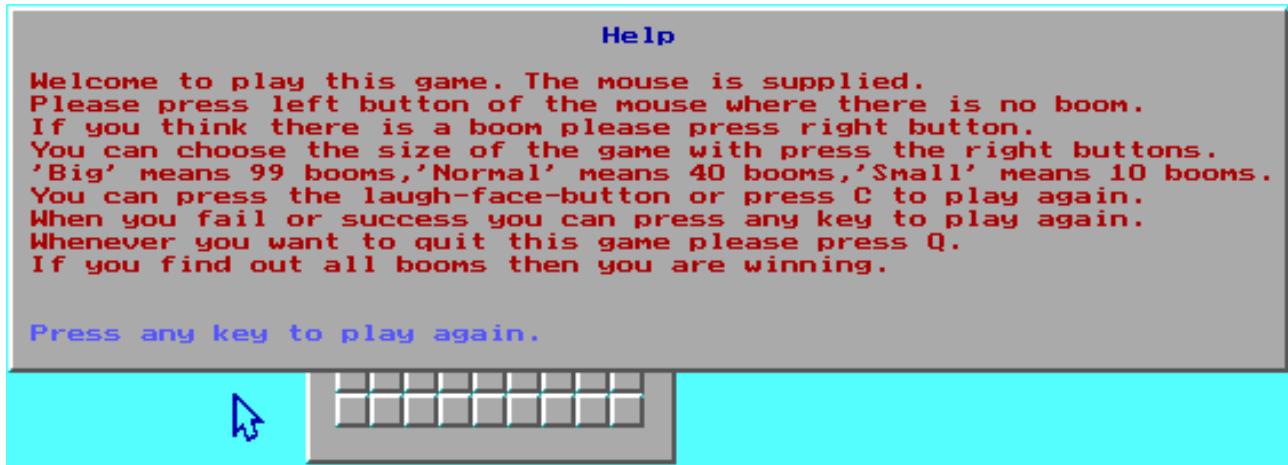


图 184-7 显示帮助信息的界面

## 实例解析

联合 REGS regs 为全局变量，用于存储通过 DOS 系统调用返回的寄存器的值（对鼠标的操作和有关信息）。

全局变量 size 用于表示每个方块的大小（正方形的边长）。

全局变量 pix、piy 是矩阵的偏移量。

全局数组 b 用于显示方格周围的雷的个数。

全局二维数组 pan[30][16] 用于记录盘面的情况：0 为没有、9 为有雷、1~8 为周围雷的个数。

全局二维数组 pan1[30][16] 用于记录当前的挖雷情况，0 为没有操作、1 为已打开、2 为已标记。

全局变量 tt 用于记录时间参数。

全局变量 Eflags 用于标记鼠标按钮的有效性，0 为有效，1 为无效，2 为鼠标的任意键等于重新开始。

程序主要功能都在主函数 main 中实现。main 函数首先定义并初始化一些操作变量，比如图形模式初始化变量，鼠标操作中的标志变量和状态变量，循环变量，矩阵大小，雷的总数，当前光标位置，菜单标志变量，时间变量，最高记录，用于保存名字的数组，用于保存鼠标图片的变量等。

接着，调用 Msinit 函数对鼠标进行初始化，并初始化图形模式，为图片指针分配内存，打开最高分数的记录文件（没有则创建一个），写入最高记录的初始值。然后设置鼠标的范围，清屏，初始化盘面，生成盘面情况（即地雷的位置），处理地雷周围的方格的数字（表明其旁边的地雷的个数）。画底座窗口，显示提示信息，绘制笑脸控制按钮，绘制地雷的盘面，显示各项从初始值（未扫的地雷个数，已扫地雷个数和所用时间等）。程序进入循环中，不断读取鼠标状态，接收用户的控制，并做出反应。这些反应包括激活菜单，对菜单的各项功能进行响应操作，对笑脸的单击操作则重新开始游戏，对于地雷区域，左键单击若无雷则打开方块，有雷则进行触雷处理，右击则标记有雷，并做相应处理，比如已扫雷、未扫雷等的计数，当扫雷完毕时，笑脸控制按钮变化，并要求输入名字，打开最高记录文件，写到记录文件中，并重新开始游戏。

Msinit() 函数用于实现鼠标的初始化。

Setmouse(int xmax,int ymax,int x,int y) 函数用于实现设置鼠标的移动范围，其中 (xmax,ymax) 为左上角的坐标，x, y 为区域的大小。

Msread(int \*xp,int \*yp,int \*bup,struct time t1,int k) 函数用于实现鼠标的读取，xp, yp, bup 分别为鼠标的位置和按键情况，t1, k 是时间参数，t1 为开始时间，k 为开始标记。

Draw(int x,int y,int sizex,int sizey) 函数用于在指定(x,y)位置画一个指定大小(长 sizex, 宽 sizey) 的矩形块 (或方块)。

Facedraw(int x,int y,int size1,int k) 用于画各种圆形脸图案，形参 x, y 表示笑脸的中心坐标，size1 是脸的大小 (半径)，k 为要描述的脸型，1 为哭、2 为平常、3 为笑。

Dead 函数用于触雷后的处理。

Draw1(int x,int y) 函数用于在 x, y 的位置描绘开始后的情况。

Open(int x,int y) 为地雷盘面的自开函数。

Random() 为随机数生成函数，加入了时间变量目的在于加强它的随机性。

Have(int sum,int x,int y,int xx,int yy) 是一个显示剩余雷数目的函数，游戏中雷的总数目不能多于 99 颗，sum 为雷的总数目，(x,y) 为显示的坐标点。

Help() 是帮助函数。

Coread() 函数用于读取记录并显示。

Ddraw2(int x,int y) 这个函数用来画鼠标光标，(x,y) 为左上角坐标，大小为 20×20。

//本实例源码参见光盘



## 归纳注释

本游戏是模仿 Windows 自带的扫雷游戏而设计的，通过本程序，读者可以学习有关在 DOS 下设计 Window 窗口界面，菜单以及通过鼠标对窗口进行操作的方法。由于在 DOS 下要显示汉字有难度，因此本程序的界面都是英文界面。通过对汉字库的调用，可以在 DOS 下显示汉字，实现

中文界面。感兴趣的读者可以查阅相关资料进行设计。

## 实例 185 推箱子游戏

### 实例说明

推箱子游戏要求通过小人在指定的房间内将所有的箱子推到指定的位置。推箱子游戏有各种不同版本，一般也有很多关可以玩。本游戏就是根据这些功能来设计简单的 DOS 下的推箱子游戏。该游戏有 4 关可以玩，第 1 关获胜，进入第 2 关，依此类推。程序采用 DOS 下的文本模式进行显示，根据各关的地图进行房间和箱子初始位置、目标位置的显示，以及人的显示。然后通过判断输入的按键进行相应的响应，有箱子在前方，则推动箱子，如果所有箱子都推到指定位置，则赢了游戏，显示相应的提示信息。程序运行结果如图 185-1~图 185-3 所示。

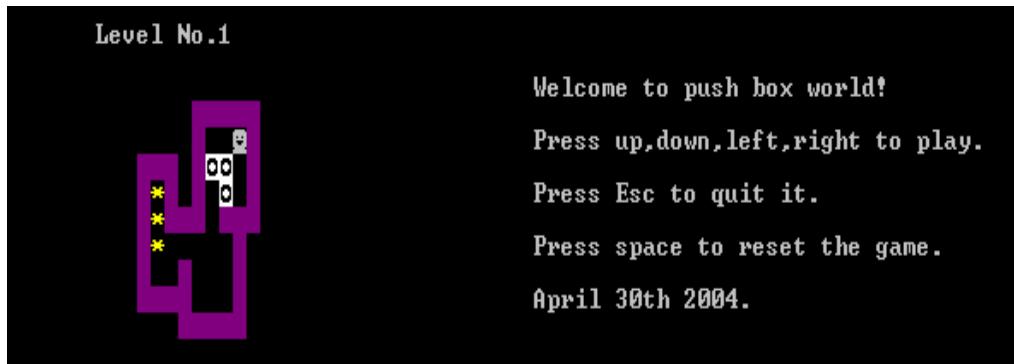


图 185-1 推箱子游戏的初始界面



图 185-2 完成第一关后的提示界面



图 185-3 完成全部的 4 关后的提示界面



## 实例解析

确定软件的功能：让玩家通过按上下左右键推箱子，当箱子全部推到目的地后出现过关信息，并显示下一关。推错了，玩家还可以按空格键重新开始。

定义核心数据结构：定义一个二维数组 `ghouse` 来记录屏幕上各点的状态，其中，0 表示什么都没有，“b”表示箱子，“w”表示墙壁，“m”表示目的地，“i”表示箱子在目的地。

对整个软件进行功能模块划分：①初始化——在屏幕上输出欢迎信息，把 `ghouse` 数组的元素初始化为 0，并根据各关的要求在屏幕上输出墙、箱子、目的地和人。用 `ghouse` 数组记录各点的状态。②进入游戏循环——这个游戏主循环是等待按键。当接收到上下左右键时执行相关操作；接收到 ESC 键时退出游戏；接收到空格键时返回本关开头；接收到无效按键时做忽略处理。③判断是否过关——用一个链表 `win` 由每关的初始化函数传给 `main` 函数。`Win` 链表主要记录屏幕上的哪些点是目的地，并记录目的地的位置。`main` 函数每执行一次操作就判断屏幕上的目的地是不是有箱子了。

结构 `winer` 为判断是否胜利的数据结构。

结构 `boxes` 为定义箱子位置的数据结构。

`putchxy()` 函数为直接写屏函数

`printwall(int x,int y)` 是在指定的坐标上画墙壁并用数组记录状态。

`Printbox(int x,int y)` 是在指定的坐标上画箱子并用数组记录状态。

`printwhither1(int x,int y,winer**win,winer **pw)` 是在特定的坐标上画目的地并用数组记录状态。

`Printwhither(int x,int y)` 是在指定的坐标上画目的地并用数组记录状态。

`Printman(int x,int y)` 是在指定的坐标上绘制人。

`Printboxin(int x,int y)` 是在指定的坐标上画箱子并用数组记录状态。

`init()` 函数为初始化函数，用于初始化数组和屏幕。

`winer *inithouse1()` 函数用于第 1 关的图像初始化。

`winer *inithouse2()` 函数用于第 2 关的图像初始化。

`winer *inithouse3()` 函数用于第 3 关的图像初始化。

`winer *inithouse4()` 函数用于第 4 关的图像初始化。

`movebox(int x,int y,char a)` 函数将在空地上的箱子移动到空地上。

`moveinbox(int x,int y,char a)` 函数将在目的地上的箱子移动到空地上。

`moveboxin(int x,int y,char a)` 函数将在空地上的箱子移动到目的地。

`moveinboxin(int x,int y,char a)` 函数将在目的地上的箱子移动到目的地。

`int judge(int x,int y)` 函数用于判断特定的坐标上的状态。

`move(int x,int y,char a)` 函数用于处理按下键盘后，人物的移动。

`reset(int i)` 函数用于处理在按下空格键后，重新开始本关的游戏。

// 本实例源码参见光盘



## 归纳注释

本程序实现在 DOS 下的推箱子游戏的简单设计。相比 Windows 下的推箱子游戏，本游戏的功能还有待完善，比如游戏的关数还不够多，游戏无法进行选关等操作。对于第一个问题，可以通过增加各关的地图来进行，对于第二个问题，可以通过增加功能键进行设置，选择游戏的关。此外，还可以增加计时功能对完成每关游戏的时间进行计数，并将最高记录保存到文件中。这些功能的设计实现，读者去可自行编写程序。

# 实例 186 五子棋游戏

## 实例说明

五子棋游戏是常见的经典游戏，在一个方阵上通过两人对弈的形式，依次在棋盘上放置两种颜色的棋子，哪一方先让五个棋子形成一条直线（包括横、竖、对角线 3 个方向）即为获胜。五子棋游戏也有许多版本，本程序在 DOS 下实现用键盘控制的五子棋游戏。程序运行结果如图 186-1~图 186-4 所示。

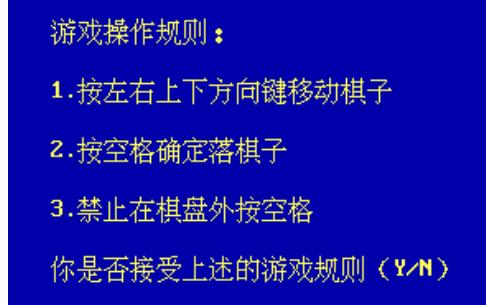


图 186-1 五子棋游戏的规则说明界面

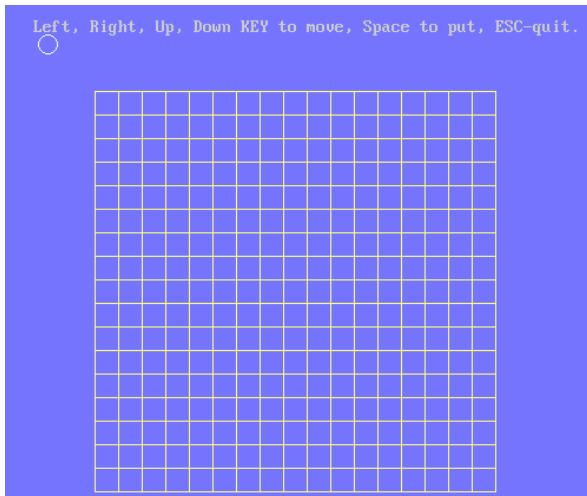


图 186-2 五子棋游戏的开始界面

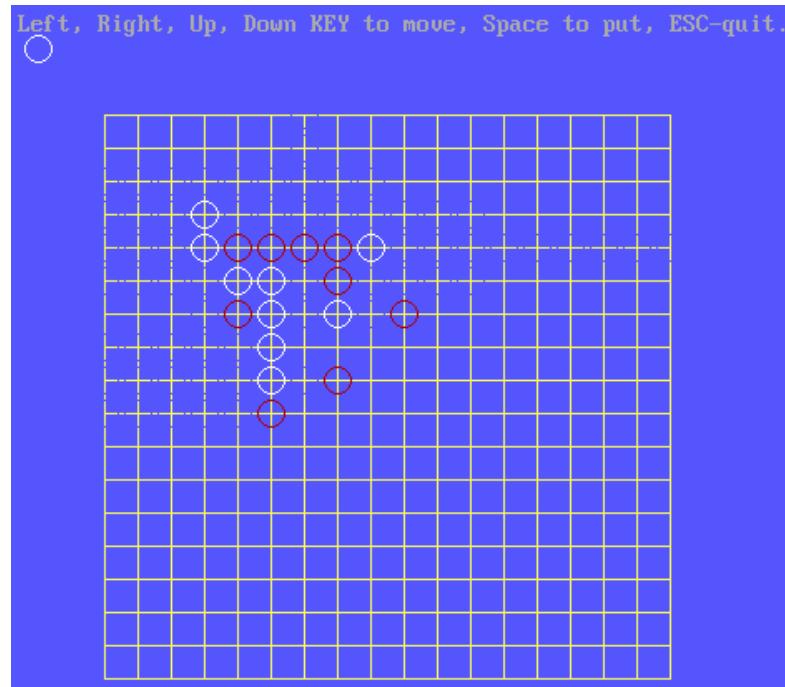


图 186-3 对弈界面



图 186-4 白方获胜的提示界面

## 实例解析

程序主要控制功能都在主函数 main 中实现。main 函数首先清屏，调用 attention 函数，输出提示信息，说明游戏规则，按 Y 继续，按 N 则退出程序。接着初始化图形模式，绘制棋盘和初始棋子的位置，输出按键提示。

然后在循环中不断从 BIOS 读取一个按键并做出判断和相应处理。在输入按键为 Space 空格键时，调用 judgewho 判断是白方还是红方的棋子，从而下一个棋子，按键为 ESC 键，则关闭图形模式并退出程序，为其他键则调用 judgekey 函数来判断输入的按键并做出相应处理。

全局数组 box[N][N] 用于保存当前棋盘上的棋子信息。

全局变量 step\_x, step\_y 用于保存白红双方的步数。

全局变量 key 用于保存输入的按键。

函数 draw\_box() 用于绘制棋盘。

函数 draw\_cicle(int x,int y,int color) 用于以指定的颜色在以 (x,y) 坐标处绘制一个圆 (相当于棋子)。

函数 change() 用于改变标志状态。

函数 judgewho(int x, int y) 用于判断是哪方的棋子。

函数 judgekey() 用于判断输入的非 ESC 和空格键的按键，并做出处理。

函数 judgeresult(int x,int y) 用于判断输赢的结果。

函数 attentoin() 用于在游戏开始时显示文本模式下的游戏规则提示信息。

//本实例源码参见光盘



## 归纳注释

本程序只是通过键盘实现简单的控制，通过初始化鼠标，设置鼠标的范围，读取鼠标状态等。另外，还可以设计相应的菜单，实现比如重新开始游戏，与计算机对弈，设置背景色，设置棋子颜色等功能。这些功能留待读者去自行编写程序。

由于游戏开头输出的提示信息是中文，所有要求在运行游戏时，要有中文环境。为了解决汉字在 DOS 下无法显示的问题，可以在输出中文的同时，在其下一行输出相应的英文，这样即使在没有中文的环境也可以显示游戏的提示信息。

# 08

## 第八部分 综合实例篇

### 精彩导读

- 综合 CAD 系统
- 功能强大的文本编辑器
- 图书管理系统
- 进销存管理系统

# 实例 187 综合 CAD 系统

## 实例说明

本实例说明如何设计综合 CAD 系统。该系统提供了对直线、矩形、圆、圆弧、图形文本和其他诸如“组”之类的对象的绘制、擦除、移动、拷贝等操作。还提供了只显模式，可以当做一个简单的图形浏览器使用。此外，CAD 系统还提供了一个打印工具包来打印已绘制的图形。系统运行界面如图 187-1 所示。

要运行 MIRCOCAD，需要配置一只鼠标和一个 VGA 显卡（工作在分辨率 640×480）。如果要打印，需要安装打印机。当然设计者也可以很容易地修改代码，让程序运行在其他的硬件平台上。

该系统主要提供了以下命令。

(1) A)rc——提示选择一个中心点、一个起点、一个终点。顺时针的从起点到终点绘制一段圆弧。这个圆弧是圆的一部分。

(2) B)o(x)——这个命令提示选择两个顶点，当第二个顶点被选择后系统将以这两个顶点为对角线绘制一个矩形。

(3) C)ircle——提示选择一个中心点和圆弧上的任意一点。系统将以中心点为圆心，以中心点到另一点的距离为半径，绘制一个圆。

(4) D)up——提示选择一个对象，之后要求选择一个新的插入点。对象将被拷贝到已选择的插入点。这个功能对复制一个刚刚从其他文件插入的组或者精确的复制一个尺寸一致的对象非常有用。复制对象在做图文件中仅仅占用 7 个字节，不管源对象有多复杂。

(5) E)r(ase)——提示选择一个对象。为了选择一个对象，用户必须把鼠标放置在对象的“基点”上。通常，对象的基点是定义对象时候选择的第一个点。

为了让选择的过程更加简单，当这个命令被触发的时候，系统将实时监控鼠标光标的位置，并且选择任何基点处于鼠标光标下的对象。一旦一个对象被选择，系统通过暂时隐藏它来表示选中状态。用户可以按下鼠标的左键来确定自己的选择，这时候就把对象从做图文件中删除了。

如果用户不小心选择到了不想被删除的对象，只要把鼠标移开，并且按鼠标右键即可取消选择。这时系统将重新绘制屏幕，用户可以继续选择要删除的对象。

如果几个的对象基点重合在同一个点，系统将选择第一个被定义的对象，用户可以按左键来确认这个选择，或者按右键来切换对象。如果用户不想选择这几个对象的任何一个，那么可以连续按鼠标的右键直到所有的对象都被跳过。

(6) S)etup / B)ase-markers——显示做图文件中所有对象的基点。

(7) F)unc——触发参数设置功能菜单，它包含以下子菜单。

① F)ont——提示选择一个新的字体字符文件。如果用户给出了一个新的文件，系统将装载这个文件并且用新字体重新绘制图形。

② I)nsert——提示输入一个做图文件名，之后选择一个插入点。把这个做图文件的图形作为一个单一的对象插入到当前的文件。新对象的基点是一个不可见的矩形的左上顶点。这个矩形足

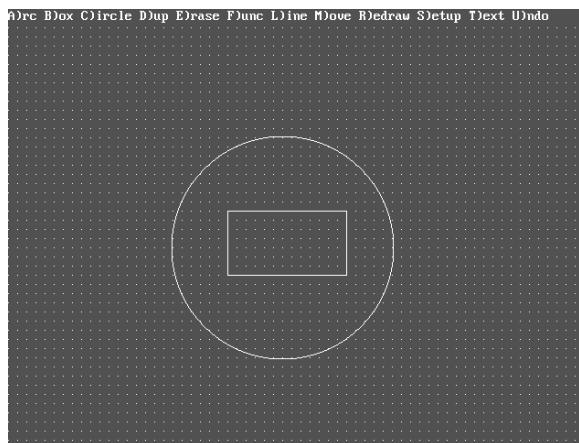


图 187-1 小型 CAD 系统使用界面

够大到可以容纳组中对象的所有基点。当选择对象位置的时候这个点将被显示。

如果用户想要将同一幅图多次插入到当前的文件中，建议使用复制命令，这样会节省做图文件的空间。

(3) L)oad——提示输入一个文件名，系统将会载入这个文件，当前没有被保存的信息将丢失。

(4) Q)uit——结束本系统。

(5) S)ave——提示输入文件名，把当前图形保存到这个文件中。

(8) L)ine——提示选择一个起点和一系列的终点。每次用户选择一个新的点，系统将经过上一个点和这个点绘制一条直线。按鼠标的右键可以取消画线操作。

(9) M)ove——提示选择一个对象和一个新的基点。这个对象将被移动到新的位置。可参看“E)rase”命令了解如何选择一个对象。

(10) R)edraw——重新绘制图形。这个命令在图形对象被文本覆盖、修正线段的空隙或者因为擦除而丢失了网格时非常有用。

(11) S)etup——触发 setup 子命令菜单，包含以下子菜单。

① B)ase-markers——触发是否显示基点。参看“E)rase”命令了解如何获得基点。当开关被打开的时候，每个对象的基点都被一个小的矩形包围。

② C)urser-base——提示输入光标的基点。当用户选择了一个以后，原来以左上角点为原点的坐标系将改成以新选择的点为原点。

③ G)rid——提示网格的大小（以像素为单位）。如果什么都不输入或者键入 0，将不显示网格。网格是间隔相等的点，可以通过这些点来对齐对象。

④ S)nap——提示输入“snap stops”的大小。当 snap 有效的时候，每次鼠标移动的距离为用户设置的最小单位。尽管不是必须的，“snap”距离最好被设置为“G)rid”的整数倍。当什么都不输入或者键入 0，表示光标可以随意移动。

⑤ T)ext-scale——这个参数定义了文本的大小，改变这个参数只会影响之后的文本。这个一幅图里面就可以有不同大小的文本。

被本系统使用的字符集建立在 16 像素宽，24 像素高的矩阵上。比例值 100 说明使用这个大小。比例值 50 将把宽和高都缩小一半，比例值 200 将把宽和高都扩大一倍。

注意：上面所有的设置都保存在做图文件中。当用户后来编辑这个文件的时候，这些参数会被重建。

(12) T)ext——把一个文本串插入做图文件中，系统提示用户输入字符串，之后要求用户选择一个插入位置。

(13) U)ndo——擦除最近一次绘制的对象。U)ndo 命令可以被多次重复，每次使用它，最上一次的对象会被删除。值得注意的是，像 M)ove 或者 E)rase 等操作是不能被取消的。

## 实例解析

### (1) 运行 CAD 系统

用如下的语法运行本 CAD 系统：

本系统 [drawing file] [F=font file] [/Display]

[drawing file]是用户想要编辑的文件的文件名。默认的扩展名是“.DWG”。如果不指定名字，系统假定默认的名字是“本系统.DWG”。这个默认的文件通常保存一幅空白的图，并且仅用于为“S)etup”属性建立默认值。

[font file]是保存字符集的字体文件，可以用它在做图文件中显示文本。如果用户不指定字体文件，那么默认的“本系统.FNT”作为缺省的字体文件。

[/Display]使系统变成一个浏览器。鼠标在这种模式下不能使用，背景网格将被擦除，图形在窗口中显示。按任意的键将结束浏览。

## (2) 鼠标

系统只支持使用鼠标的左键和右键。但鼠标本身可以是二键或者三键的。

当处于命令处理框架的最顶层的时候(在屏幕顶端没有命令文本提示)，鼠标按钮的功能如下。

左键：触发在屏幕顶端的命令菜单。按下对象的按键即可激发相应的命令。

右键：触发上一次执行的命令。这是一个执行多次相同操作的快捷办法。比如可以快速的绘制多个矩形。

如果不特意注明，那么鼠标的按钮在命令被激活的时候功能如下。

左键：执行屏幕顶端提示的操作。有一些命令需要选择好几个位置。每定位一个位置，都可以在屏幕顶端看到提示。用户只要简单的把鼠标指针移动到想要的位置，按下左键即可选择到相应的点。

右键：取消操作。在大多数情况下按下鼠标右键将直接返回到命令框架的最顶层（没有活动命令）。在有些情况下，将会取消一个子功能，用户还可以继续执行命令。

## (3) MCPRINT 工具集

本系统包含 MCPRINT.COM 程序，通过这个打印程序可以把做出的图形通过 HP LaserJet 打印机或者 EPSON 点阵式打印机打印出来。MCPRINT 从图形文件中建立一个光栅位图，之后把它传送到打印机打印。用户可以很容易的修改 MCPRINT 程序让它工作在其他打印机上。

MCPRINT 命令的语法：

```
MCPRINT <drawing file> <printer type> [F=font file] [D=device]
```

<drawing file>是用户要打印的包含图形的文件。“.DWG”是默认的扩展名。

<printer type>指明用户拥有何种类型的打印机。如果用户没有修改打印程序，那么应该是 LASERJET 和 EPSON 其中之一。

[font file] 是保存字符集的字体文件，用它在做图文件中显示文本。如果不指定字体文件，那么默认的“MICROCAD.FNT”作为缺省的字体文件

[device]指定打印机的输出端口。如果没有指定，“LPT1”是默认的。用户也可以把这个端口指定为一个文件，这样就把图形重定向到文件中了。

## (4) 做图文件的格式

MICROCAD 产生的文件由 8 位的字节和 16 位的字组成。字是高位在前存储的。文件的最前面的几个字节是用来存储“S)etup”的值，如下所示。

Word = Grid 大小(0 表示禁止)

Word = Snap 大小(0 表示禁止)

Word = 文本比例 (100 = 1:1)

Word = 光标原点的 X 值

Word = 光标原点的 Y 值

Byte = 对象的基点显示与否，0 = ON, 1 = OFF

设置参数的后面是 0 个或者多个对象，代表直线、圆等，每种类型的使用格式如下。

① 结束符——Byte = 0x00

这个不是 MICROCAD 必须的，而且程序通常也不产生这个结束符。但是程序遇到一个包含单一 0x00 字节的对象时就会停止绘制对象。它有两个作用：第一个作用是如果想要用类似 XMODEM 的协议传输文件，传输过程中，文件的尾部有可能被添加一些垃圾信息。用户可以首先自己添加一个结束符来避免文件出错；第二个作用是一旦碰到结束符，后来的内容虽然可以添加到文件，但是图形重绘或者重新载入时，结束符后面的对象将不会被绘出。

② 直线——Byte = 0x01

Word = 第一个点绝对 X 值

Word = 第一个点绝对 Y 值

Word = 第二个点相对 X 值

Word = 第二个点相对 Y 值

③ 矩形——Byte = 0x02

Word = 第一个点绝对 X 值

Word = 第一个点绝对 Y 值

Word = 第二个点相对 X 值

Word = 第二个点相对 Y 值

④ 圆——Byte = 0x03

Word = 中心点绝对 X 值

Word = 中心点绝对 Y 值

Word = 圆的半径

⑤ 文本——Byte = 0x04

Word = 文本的绝对 X 值

Word = 文本的绝对 Y 值

Word = 文本比例 (100 = 1:1 = 16×24)

Bytes= 文本串

Byte = 0 (结束符)

⑥ 圆弧——Byte = 0x05

Word = 中心点绝对 X 值

Word = 中心点绝对 Y 值

Word = 圆弧的半径

Word = 起点的角度(0-255)

Word = 终点的角度(0-255)

⑦ 组——Byte = 0x06

Word = 组位置的绝对 X 值

Word = 组位置的绝对 Y 值

Word = 下面包含对象的字节数 (in bytes)

... 下面是 0 个或者多个对象...

X/Y 的坐标都是相对于组的起点的。

⑧ 绝对拷贝——Byte = 0x07

Word = 复制对象的的绝对 X 值

Word = 复制对象的的绝对 Y 值

Word = 源对象在文件中距文件头的偏移值

⑨ 相对拷贝——Byte = 0x08

Word = 复制对象的的绝对 X 值

Word = 复制对象的的绝对 Y 值

Word = 复制对象和源对象的偏移值 (负值)

当对象通过命令菜单的“D)up”复制时候，使用绝对拷贝。当处理离散对象的时候，这个形式非常有用。

当然，对象被分组后（插入一幅图），任何绝对拷贝的对象将被改成相对拷贝。但是用户不用

担心使用“E)rase”等命令后，对象的位置会改变。MICROCAD 把一个组当成一个对象来处理。因为绝对拷贝一定要参照相同组的对象并且一个组的内容不会被改变，这样做是非常安全的。

#### (5) 字符字体

下面的字符字体文件包含在系统中。

MICROCAD.FNT——这是默认的字体文件。

HIGHRES.FNT——这个字体的质量相当好，但是当它缩小的时候不如 MICROCAD.FNT 的效果。

MATRIX.FNT——一种(8×8)低分辨率的字体。效果看起来像点阵打印机的输出。

THINLINE.FNT——一种细长的字体。

系统同样包含一个字体编辑器(FE.COM)，通过这个程序用户可以创造和编辑自己需要的字符字体文件。用户只要以要编辑的字体文件名为参数即可运行这个程序。

#### (6) 其他注意事项

系统所有内部的计算都是16位的。所以圆和圆弧如果半径太大，很可能显示出错。因为这些对象需要平方运算，结果可能超出16位。

网格不是作为对象的一部分被记录的。如果用户执行了M)oove、E)rase、U)ndo等在网格上的操作，有的时候网格就会消失。每次重绘屏幕的时候，这些网格又会被重建。

## 程序代码

### 【程序187】综合CAD系统

//本实例源码参见光盘

由于本系统比较大，涉及到的文件比较多，为了节省篇幅，实例的源代码将在光盘中给出。下面只介绍部分核心代码的情况。

#### (1) 程序头

程序头主要定义了有关显示器、鼠标等设备的常量和全局变量。定义了与图形文件相关的缓冲区。对图形的编辑，首先保存在缓冲区中，只有用户选择了保存文件命令，在缓冲区的数据才会被写入磁盘的文件。程序头还定义了一些必须初始化的做图参数。

#### (2) 分发文件操作命令

function 函数用来显示S)etup 命令项的子功能，并且接收用户输入的命令。分发子命令到相应的处理函数。

#### (3) 插入图形函数

执行这个函数可以把一个做图文件的图形插入到当前图形中。一个图形是以一个组的形式插入到另外一个文件中的，并且组中的每个对象基点表示相对组基点的偏移值，在插入前的绝对拷贝对象都要转换为相对拷贝。

#### (4) 图形绘制函数

draw\_line()函数在屏幕上绘制出一条或者多条直线。函数提示用户选择一个起点和一系列的终点。每次用户选择一个新的点，MICROCAD 将经过上一个点和这个点绘制一条直线。按鼠标的右键可以取消画线操作。函数调用了line()函数，可以通过这个函数经过两个指定端点绘制一条直线。函数将提示用户选择两个顶点，当第2个顶点被选择后，MICROCAD 将以这两个顶点为对角线绘制一个矩形。用户可以通过点击鼠标右键重新选择顶点。

提示用户选择一个中心点和圆弧上的任意一点。MICROCAD 以中心点为圆心，以中心点到另一点的距离为半径，绘制一个圆。

提示用户选择一个中心点、一个起点、一个终点。这个圆弧是一个圆的一部分。程序通过find\_vector()函数找到离起点和终点最近的矢量。过起点矢量顺时针到终点矢量绘制一段圆弧。

把一个文本串插入做图文件中，系统提示输入字符串，之后要求用户选择一个插入位置。显示的文本的字体是在 `font` 文件中定义的。

程序通过从头搜索做图缓存，找到最后一次绘制的图形在缓存的位置。这样的效率比较低，有兴趣的读者可以在程序头中加入一个全局变量，来记录最后绘制的图形元素。加入这个变量可以避免每次查找最后的图形元素时，要重新搜索一次缓存。

#### (5) 删除、移动、拷贝对象函数

上面是操作对象的几个函数，包括删除、移动、拷贝。这些操作都是建立在前面定义的数据结构上，通过仔细研读本例，可以发现好的算法都是以精巧的数据结构定义为前提的。

#### (6) 绘制对象函数

绘制对象函数把 `dpos` 变量指示的对象偏移 (`xoffset, yoffset`) 绘制出来。程序首先得到源对象的绝对位置，把绝对位置加上偏移量后作为绘制位置。

#### (7) 绘制对象函数

通过上面的函数，文件指针可以毫无偏差的在对象间移动，大大简化了用户调用接口，上层程序不用理会后面的对象的类型，只需调用函数就可以跳过一个对象。

#### (8) 对象选择函数

当使用 `C`opy、`E`rase、`M`ove 命令的时候，都要先选择要操作的对象。MICROCAD 实时监控鼠标的光标的位置，并且选择任何基点处于鼠标光标下的对象。一旦一个对象被选择，MICROCAD 通过暂时隐藏它来表示选中状态。用户可以按下鼠标的左键来确定自己的选择，这时候就把对象从做图文件中删除了。

#### (9) `find_vector()` 函数

把一个圆在每个象限分成 64 等份，这样每个圆在每个象限都有 64 条矢量。用户给出的任意一点有可能不过其中的任意一条矢量。这时候就用一条距离所求点最近的矢量代表端点的角度。

#### (10) 底层图像绘制函数

通过在屏幕上写像素的方式绘制各种图元。

#### (11) 缓存和文件操作函数

这些函数嵌入了一些汇编语言，大大提高了访问硬件的速度。

#### (12) 设备相关函数

设备相关函数主要用于图形界面的初始化、鼠标的初始化和对鼠标的调用。它们中间也嵌入了汇编代码，提高了对硬件访问的效率。



## 归纳注释

本书使用了汇编指令，而内嵌汇编指令的 C 程序只能采用 TCC 命令行的编译连接方法，用 TCC 命令行实现编译连接方法是：

`TCC -B-L: \LIB 文件名 库文件名`

其中 “-L” 选项指定了连接所需的库文件路径，文件名指有内嵌汇编指令的 C 程序名，库文件指程序中要用的库函数所在的库文件（Turbo C 标准库可以省略）。

含有内嵌汇编指令的 C 程序进行编译时，必须有 “-B” 选择项，否则编译时，一旦遇到汇编代码，便立刻给出警告信息，并以 “-B” 选择重新编译，若在 C 程序中加上 “#program inline” 语句，则可以省略 “-B” 选择项。

由于汇编时 TCC 要调用 TASM.EXE 程序，若无此程序，可以将 MASM (3.0 以上) 改名为 TASM.EXE 以代替之。

# 实例 188 功能强大的文本编辑器

## 实例说明

在前面的章节中，介绍了一个简单的文本编辑器，本章将介绍一个功能强大的文本编辑器——EDITOR。EDITOR 是一个多文档/多窗口的文本/二进制编辑器，适合编辑批处理文件、二进制文件、文本文件和多种编程语言。惟一的限制就是编辑的文件数目和大小受制于内存的大小。同样，最大的窗口数目也受制于内存的大小。程序运行后的界面如图 188-1 所示。

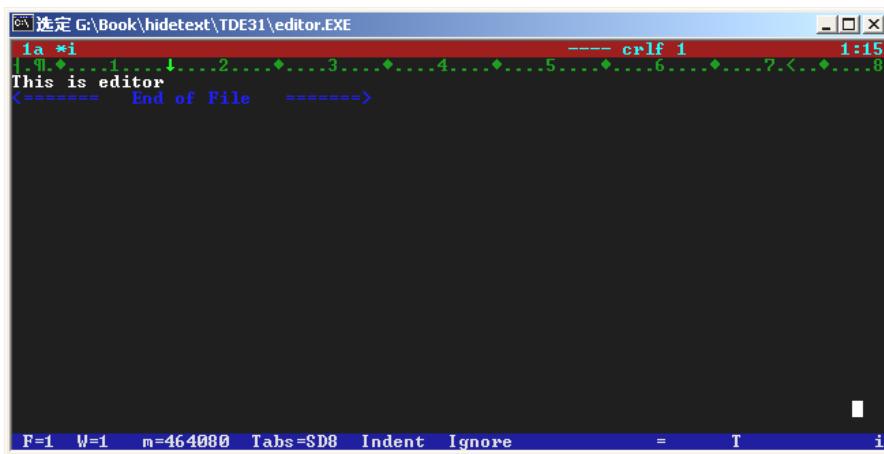


图 188-1 编辑器界面

EDITOR 可用于处理日常的数据文件、计算机代码和各式各样的文本文件。EDITOR 提供的窗口界面和光标操作命令使得对文件的编辑非常直观和方便。特别是块操作命令大大方便了对源代码的编辑和文本文件的格式化。此外，EDITOR 还有一些简洁的命令格式化文本和段落。

## 实例解析

### (1) 运行 EDITOR

```
editor [ [-f search_pattern] file name(s)]  
editor [ [-g regular_expression] file name(s)]  
editor [ [-b] file name]
```

### (2) 使用 editor

```
editor  
editor foo.bar  
editor c:\c70\TDE\main.c  
editor *.c \qc25\TDE\*.h  
editor foo.bar foobar f* foo.* f??b??
```

### (3) 匹配查找字符串或者正则表达式

```
editor -f find_me_load_me foo.*  
editor -f find_me_load_me foo.* *.bar  
  
editor -g "this|that" foo.* <== 找 "this"或者"that"  
editor -g [a-zA-Z0-9_]+(\ foo.bar <== 查找 C 语言函数
```

如果符号“|”需要在正则表达式中出现，搜索字符串必须用引号括起来。否则 C 的命令行解释器会把“|”理解成管道操作符（参见 DOS 的命令帮助），而不是正则表达式中的“或”运算符。

#### (4) 编辑二进制文件

```
editor -b tde.exe  
editor -b80 tde.exe
```

如果用户在命令行中不指定文件名, editor 将提示用户输入一个文件名。如果指定的文件不存在, 那么系统将建立一个以这个名字为文件名的新文件。点击两次“Enter”键将出现一个目录列表, 如图 188-2 所示。

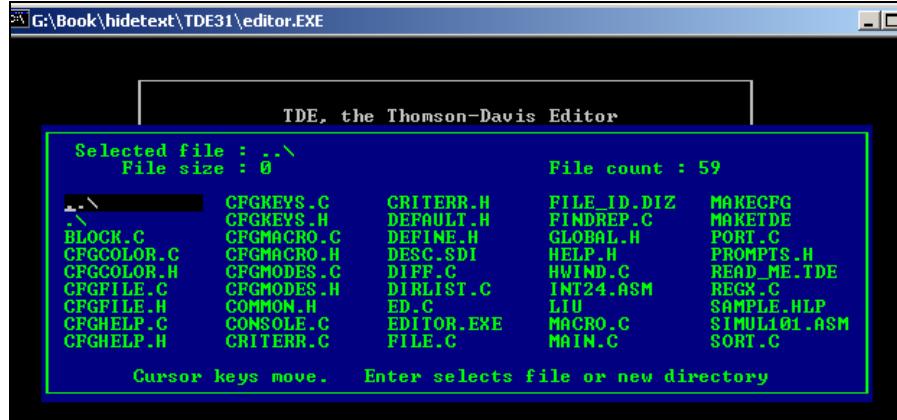


图 188-2 目录列表界面

#### (5) 文件操作命令

文件操作有以下快捷命令。

F2 表示保存文件。当前的文件将被修改后的文件覆盖。系统默认覆盖原文件, 所以不会提示输入新的文件名。

F3 表示关闭当前文件。如果用户正在编辑多个文件或者打开了多个窗口, editor 将搜索其中一个不可见的文件在当前窗口显示出来。如果没有不可见的窗口, editor 就会重新组合窗口。如果被关闭的文件是当前惟一被编辑的文件, F3 命令将关闭 editor 程序。如果关闭前对文件做了修改, 系统会提示用户是否保存修改, 如图 188-3 所示。如果修改当前文件的时候又加载了一个新的文件, 系统会创建一个不可见的窗口。



图 188-3 关闭当前文件界面

F4 表示保存并且关闭当前文件。这个命令的效果相当于 F2 和 F3 组合。

#F2 表示另存为。如果不覆盖原文件但又想保存当前的文件内容, 可以选择另外一个名字保存文件。这个命令提示用户输入一个文件名以创建一个新的文件来保存当前对文件的修改(可以按 escape 键放弃这个命令)。

#F4 表示编辑新文件。这个命令把新文件加载为当前窗口, 使现在的窗口变得不可见。用户

甚至可以加载同一个文件的几份副本。如果在修改过程中，需要参考未改变前的源文件，这个命令就非常有用。

@F2 表示设置文件属性。属性包括 A=存档文件，S=系统文件，H=隐藏文件，R=只读文件。如果想要编辑一个只读文件，那么就要修改文件属性：编辑文件；把文件属性设置为存档；保存对文件的修改；把文件属性设置回只读。

@F4 表示编辑下一个文件。如果在命令行中输入了多个文件名或者通配符，通过这个命令可以把下一个匹配的文件加载到编辑器中来。

```
-f pattern file(s)      (DefineGrep 的命令行形式)
-g pattern file(s)      (DefineRegXGrep 的命令行形式)
^F12 = DefineRegXGrep
#F12 = DefineGrep
```

Grep 将搜索文件来查找一个模式，并且仅仅加载那些匹配到的文件。命令行形式的 Define Grep 和功能键定义的 Grep 有一些细微的区别。在命令行下，空格键被 DOS 命令行解释器理解成分隔符。因此在命令行方式下，模式串不能嵌入空格。但是在功能键的方式下，模式中可以含有任何字符。定义 Grep 同时定义了搜索模式，当一个文件已经被加载，按 RepeatFind 可以跳转到模式出现的下一个位置。这里实际上有两个查找结构，一个为标准搜索函数定义，一个为 Grep 定义。虽然定义 Grep 同时定义了搜索模式，但是重新定义标准的搜索结构对 Grep 模式却没有影响。除非重新定义一个 Grep，Grep 模式是不会发生变化。

F12 表示 RepeatGrep，在 Grep 被定义后，通过按此键来搜索下一个文件。

#### (6) 查找、替换命令

EDITOR 使用了 Boyer-Moore 搜索算法来查找字符串。通常，要查找的模式越长，搜索速度越快。所以，如果想加快搜索速度，应该尽量搜索更长的字符串，比如词组。

^F5 表示大小写敏感与否。用户可以在任何时候定义搜索是否大小写敏感。即使在用户定义了搜索模式后，这个命令依然起作用。

#F5 表示向前查找。用户输入定义模式后，EDITOR 就会向前搜索文件来匹配。

#F6 表示 find backward。用户输入定义模式后，EDITOR 就会向后搜索文件来匹配。

F5 表示重复向前查找。一旦用户通过 Shift+F5 定义好向前查找的模式，可以通过 F5 来查找下一个匹配的位置。如果到了文件的尾部，就会翻转到头部继续向前查找。

F6 表示重复向后查找。一旦用户通过 Shift+F6 定义好向后查找的模式，可以通过 F6 来查找下一个匹配的位置。如果到了文件的头部，就会翻转到尾部继续向后查找。

#F8 表示替换。这个命令要求用户指定查找和要替换为的内容。它同样会提示用户指定替换的方向。如果想要设定大小写敏感，一定要翻转搜索标志。可以选择提示替换或者非提示替换。不管提示还是非提示，这个函数实际上把光标移动到匹配到的位置，如图 188-4 所示。

#F7 表示 FindRegX，提示用户输入一个正则表达式作为查找条件，可以通过 F1 获得帮助。

F7 表示 RepeatFindRegX，一旦定义了一个正则表达式，通过这个键来查找下一个。

@F7 表示 RepeatFindRegXBackward，一旦定义了一个正则表达式，通过这个键来查找上一个。

#### (7) 比较命令

#F11 表示 DefineDiff，这个命令初始化 diff，它提示用户指定要比较的窗口号和字母。命令将忽略领头的空格、空行，忽略大小写，忽略行尾。任何两个窗口都可以被比较，但是要求两个都是可见的。

```

G:\Book\hidetext\TDE31\editor.EXE
1a CFGKEYS.C      A--- crlf 431
int ch;
int i;

w_ptr = NULL;
hw.dply_col = 2;
hw.dply_row = 3;
hw.line_length = 69;
hw.avail_lines = 12;
hw.v_row = 0;
hw.select = 0;
hw.num_entries = AVAIL_KEYS;
hw.ulft_col = 3;
hw.ulft_row = 2;
hw.total_col = 73;
hw.total_row = 21;

initialize_keys();
master_help(&hw, key_defs, key_head, t, &ch);
if (ch == F10) {
    for (i=0; i<hw.num_entries; i++)
        key_func.key[key_defs[i].key_index] = key_defs[i].func_index;
}

```

F=1 W=1 m=439488 Tabs=SD8 Indent Ignore i=69x T

图 188-4 替换操作界面

Diff 命令将会提示用户输入如下内容：

```

DIFF: Enter first window number and letter (e.g. 1a) :
DIFF: Enter next window number and letter (e.g. 2a) :
DIFF: Start diff at (B)eginning of file or (C)urrent position? (b/c)
DIFF: Ignore leading spaces (y/n)?
DIFF: Ignore all space (y/n)?
DIFF: Ignore blank lines (y/n)?
DIFF: Ignore end of line (useful with reformatted paragraphs) (y/n)?

```

F11 表示重复比较命令。如果找到了一个差别，按 F11 可以查找下个不一样的位置。

#### (8) 窗口命令

F8 表示垂直拆分窗口。当前的文件将在多个窗口被显示出来。见图 188-5，对于文件的修改会在相同文件拷贝的每个窗口显示出来。在一个窗口对文件做的块标记，也会在另外的窗口显示出来。每个窗口最多可以有 15 行，所有可以显示的窗口数目被显示能力所限制。

```

G:\Book\hidetext\TDE31\editor.EXE
1a *CFGKEYS.C      65:32
int i;

w_ptr = NULL;
hw.dply_col = 2;
hw.dply_row = 3;
hw.line_length = 69;
hw.avail_lines = 12;
hw.v_row = 0;
hw.select = 0;
hw.num_entries = AVAIL_KEYS;
hw.ulft_col = 3;
hw.ulft_row = 2;
hw.total_col = 73;
hw.total_row = 21;

initialize_keys();
master_help(&hw, key_defs, key_head, t, &ch);
if (ch == F10) {
    for (i=0; i<hw.num_entries
        key_func.key[key_defs[i].key_index]
        fseek(tde_exe, keys_offset, SEEK_SET);
        fwrite(<void *>&key_func, sizeof(KEY_FUN
F=1 W=2 m=439488 Tabs=SD8 Indent Ignore

```

图 188-5 垂直拆分窗口界面

F9 表示水平拆分窗口。当前的文件在多个窗口被显示出来。见图 188-6，对于文件的修改会在相同文件拷贝的每个窗口显示出来。在一个窗口对文件做的块标记，也会在另外的窗口显示出来。

```

G:\Book\hidetext\TDE31\editor.EXE
1a CFGKEYS.C
1b CFGKEYS.C
1c CFGKEYS.C
F=1 W=3 m=439488 Tabs=SD8 Indent Ignore

```

图 188-6 水平拆分窗口界面

#F9 表示调整窗口的大小。使用向上和向下的光标键可以把窗口调整到合适的大小。但是不允许调整顶端窗口的大小。

^F9 表示最大化当前窗口。使当前的窗口充满屏幕，除了当前的窗口，其他的窗口都变得不可见。

F10 表示下一个窗口。如果显示了不止一个窗口，可以按 F10 使光标移到下一个窗口。

#F10 表示前一个窗口和 F10 命令相反。

^F10 表示下一个隐藏窗口，在当前窗口的位置显示下一个隐藏或者不可见的窗口。当前的窗口变得不可见。

#### (9) 块命令

块操作可以在文件内部也可以在文件间执行。行块的移动操作对象是光标所在行文本。矩形块的和 stream 块移动操作从光标所在的列开始。按 Control+Break 可以阻止绝大多数的矩形块操作。其他的块操作因为执行速度很快，几乎没有机会停止它们。

@B 表示标记矩形块——用来标记矩形的左上角和右下角，如图 188-7 所示。

```

G:\Book\hidetext\TDE31\editor.EXE
1a CFGKEYS.C
* Name: tdekeys
* Date: October 1, 1991
* Notes: Set up most of the window global variables.
*/
void tdekeys( void )
{
HELP_WINDOW hw;
char t[80];
int ch;
int i;

w_ptr = NULL;
hw.dply_col = 2;
hw.dply_row = 3;
hw.line_length = 69;
hw.avail_lines = 12;
hw.v_row = 0;
hw.select = 0;
hw.num_entries = AVAIL_KEYS;
hw.ulft_col = 3;
hw.ulft_row = 2;
hw.total_col = 73;
F=1 W=1 m=439488 Tabs=SD8 Indent Ignore

```

图 188-7 标记矩形块界面

@L 表示标记行块——和上面的类似，但是整个行都被标记了。

@X 表示标记 steam 块——在开始行和结束行之间的文本被标记。

@U 表示取消对块的标记——如果用户不小心错误的标记了一个块，可以用这个命令取消。注意，对标记的块进行几次操作后，块可能还是被标记的。

@G 表示归并要删除的块——有点像自我扩展，这个函数将删除在块中的文本。

@M 表示移动块——把光标移动合适的位置，按@M 把选中的块引导到这个位置。在行块模式下，文本被移动到当前光标所在的行的下面。在矩形块和 stream 块模式下，块被移动到光标所在行。

@C 表示拷贝块——在行块模式下，文本被拷贝到当前光标所在的行的下面。在矩形块和 stream 块模式下，块被拷贝到光标所在行。

@K 表示拷贝块——和上个命令相同，除了块还是被标记。

@O 表示覆盖块——只能覆盖矩形块。原来的块保持标记状态。

@F 表示填充矩形块——用字符填充被标记的矩形块。

@N 表示给矩形块标号——EDITOR 提示用户输入开始数字，增量。EDITOR 只处理整数。

@P 表示打印块——把选中的块输出到打印设备并打印出来。在运行此命令之前，首先要选定一个块，可以按 Control+Break 中止打印。

@S 表示矩形块排列——矩形块中的行按照内容做升序或者降序排列。用户可以按 Control+Break 中止排序。

#@S 表示交换块——交换块只对矩形块有效。被交换的区域假定和矩形块的宽度相同。要保证光标在要交换的区域的左上角。

@W 表示把块写入文件中——EDITOR 提示用户输入文件名。

@E 表示用 Tab 键移动当前块——必须是行块，移动的距离是 Tab 键设置的空格数。

@T 表示裁减块尾的空格

@<表示转化为大写字母——把一个块中所有小写字母转换为大写字母。

@>表示转化为小写字母——把一个块中所有大写字母转换为小写字母。

#@>表示 BlockFixUUE——解决 Email 传输中 EBCDIC 码到 ASCII 码的转换问题。

#### (10) 字处理命令

@V 表示翻转换行模式——在关闭换行、固定换行和动态换行间切换。在固定换行模式下，左面的空白在页面的设置被明确的指定。在动态换行模式下，左面的空白被当前的行的排版所决定。

在 EDITOR 中，除了空白，字换行和格式化段落是没有任何关联的。字换行主要是用来当光标或者字符接近右边空白的时候回车换行的。格式化的段落和文本在任何时候都可以使用，他们不依赖于换行模式。

^F6 表示设置左面的空白——可以设置为大于等于 1 列，小于右空白的任意列数。

^F7 表示设置右面的空白——可以设置为大于左空白，小于 1040 的任意列数。

^F8 表示设置段落空白——段落的空白可以设置为任何小于右空白的列数。

格式化段落——当前没有给任何键赋予此功能

文本从段落的开始就根据当前设置的左右空白和段落空白被格式化。如果光标没有在段落的开始，EDITOR 会搜索到段落头，并且从头开始格式化段落。

^B 表示格式化文本——文本从光标的位置开始被格式化。这个命令并不搜索段落的头。

@F8 表示左对齐——左对齐当前行。

@F9 表示右对齐——右对齐当前行。

@F10 表示居中——把当前行居中。

#### (11) TAB 键

Tab——在插入模式下，按下 Tab 键会插入几个空格，并且向右移动光标。如果在改写模式下，仅仅向右移动光标。

#Tab——把光标移回先前的位置。

^Tab——设置逻辑和物理 Tab 间隔。

@Tab——打开 Smart Tab 模式（这个命令在 Windows 自带的模拟 DOS 环境中可能无法正常工作）。通过寻找在光标上面的第一个非空格行来确定 Smart Tab 的位置。

#@T——打开 TabInflate 模式（这个命令在 Windows 自带的模拟 DOS 环境中可能无法正常工作），在这个模式下，Tab 不会被物理扩展。

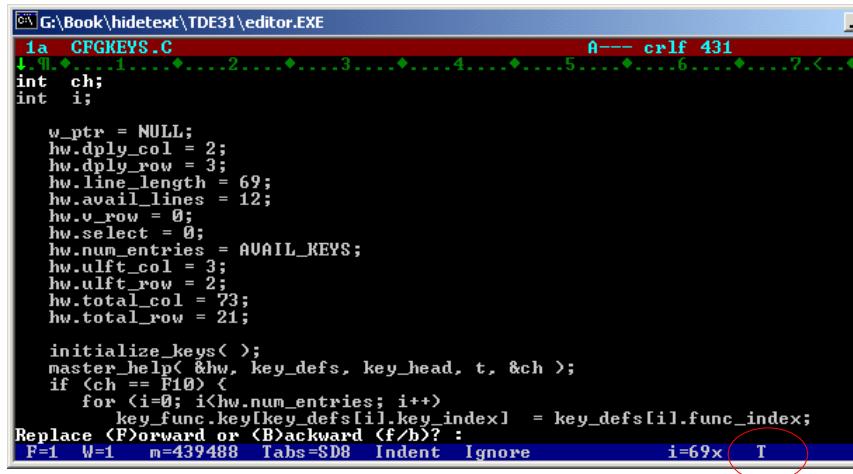
#### (12) 开关命令

^F1——开关光标同步标志。只有光标移动命令才被同步。

^F2——开关 Eol 显示。Eol 是一个特殊的标识行结束的符号。

^F3——决定在行尾写什么字符：<CR><LF>、<LF> 或者什么都不写。EDITOR 可以读任何一种格式的文件。但是用户可以通过这个命令指定写文件的时候采用哪种方法。

^F4——决定是否裁减行尾的空格。如果这个功能打开，那么在屏幕下面的状态栏的右方会显示一个 T，如图 188-8 所示。



```
G:\Book\hidetext\TDE31\editor.EXE
1a CFGKEYS.C
int ch;
int i;

v_ptr = NULL;
hw.dply_col = 2;
hw.dply_row = 3;
hw.line_length = 69;
hw.avail_lines = 12;
hw.v_row = 0;
hw.select = 0;
hw.num_entries = AVAIL_KEYS;
hw.ulft_col = 3;
hw.ulft_row = 2;
hw.total_col = 23;
hw.total_row = 21;

initialize_keys();
master_help(&hw, key_defs, key_head, t, &ch);
if (ch == F10) {
    for (i=0; i<hw.num_entries; i++)
        key_func.key[key_defs[i].key_index] = key_defs[i].func_index;
}
Replace <F>forward or <B>backward {f/b}?
F=1 W=1 n=439488 Tabs=SD8 Indent Ignore
i=69x T
```

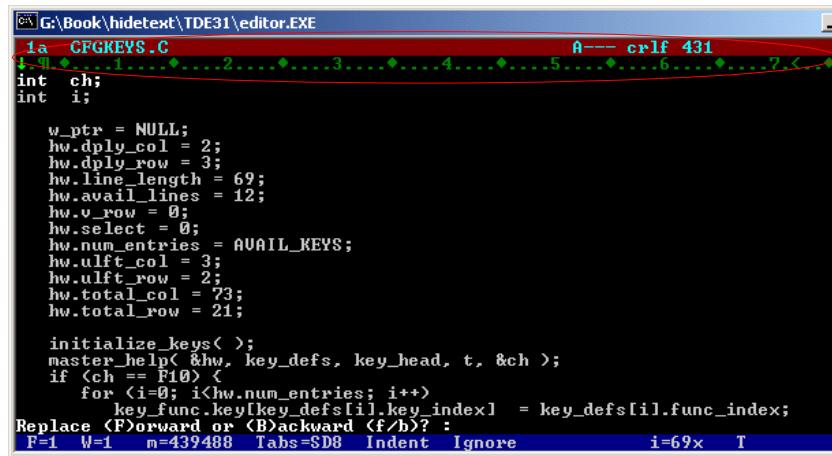
图 188-8 裁减行尾的空格界面

^F5 表示决定搜索或者排序的时候是否大小写敏感——可以在任何时候翻转这个标志，即使在定义过搜索模式后。

Ins——在插入和改写模式下切换。

@I——决定是否采用缩进模式。在缩进模式下，EDITOR 在前面的行中搜索第一个非空格的字符，当按下回车的时候，把光标移到这列。

@R 表示标尺开关——决定是否在可见的窗口显示标尺，如图 188-9 所示。



```
G:\Book\hidetext\TDE31\editor.EXE
1a CFGKEYS.C
int ch;
int i;

v_ptr = NULL;
hw.dply_col = 2;
hw.dply_row = 3;
hw.line_length = 69;
hw.avail_lines = 12;
hw.v_row = 0;
hw.select = 0;
hw.num_entries = AVAIL_KEYS;
hw.ulft_col = 3;
hw.ulft_row = 2;
hw.total_col = 23;
hw.total_row = 21;

initialize_keys();
master_help(&hw, key_defs, key_head, t, &ch);
if (ch == F10) {
    for (i=0; i<hw.num_entries; i++)
        key_func.key[key_defs[i].key_index] = key_defs[i].func_index;
}
Replace <F>forward or <B>backward {f/b}?
F=1 W=1 n=439488 Tabs=SD8 Indent Ignore
i=69x T
```

图 188-9 标尺开关界面

@V 表示换行方式。

@Z——决定是否在文件的尾部写入“Control+Z”。对于某些程序，如果文件的尾部没有“Control+Z”，它们将不会正常工作。

### (13) 其他命令

Enter——在光标的位置插入一行，如果光标在一行的中间，从光标的位置把行一分为二。如果是缩进模式，还会匹配缩进。

#Enter——把光标向下移一行。光标被放置到下一行中第一个非空格字符的位置。这个按键不会在文件中添加行。

^Enter——把光标向下移一行。光标被放置到下一行中第一列的位置。这个按键不会在文件中添加行。

Up(箭头)——把光标向上移。

Down(箭头)——把光标向下移。

Left(箭头)——把光标向左移。

Right(箭头)——把光标向右移。

Home——使光标在一行中的第一个分空格字符和第一列中间跳转。

End——把光标移到一行行尾。

Page Up——把光标向上移一页。

Page Down——把光标向下移一页。

^Right——把光标移到一个单词的右端。

^Left——把光标移到一个单词的左端。

^#Right(Control+Shift+Right)——把屏幕向右移动一个字符。

^#Left(Control+Shift+Left)——把屏幕向左移动一个字符。

^Home——把光标移动到屏幕上的第一行。

^End——把光标移动到屏幕上的最后一行。

^Page Up——把光标移到文件的第一页。

^Page Down——把光标移到文件的最后一页。

ESC=撤销对一行的修改——使行的状态回到未修改前的状态。当光标离开此行后，所有对这行的修改将生效，且没有快捷的途径恢复到原来的内容。

Del——删除光标上面的字符。

^Del——删除一个文件中的所有字符，就好像他们在—个流中，文件中的所有字符最终都会被删除。

Backspace——删除光标前面的字符，并且左移光标。如果光标在行的第一列，那么当前的行就会和上面一行合并。

@=——复制当前行

@-——删除从当前光标位置到行尾的文本

^Y——删除当前行，光标不动。

@D——删除当前行，光标不动（和^Y一样）。

@Y或者^U——取消最近一次的删除行操作。重做缓存最大可以容纳200个被删除的行，如果删除的行大于200个，那么最早删除的行就被移出缓存为新删除的行空出空间。用户可以重复按^U来恢复在缓存中的所有行。

上面介绍的删除命令，删除的行都会保存在缓存中。

@A——在光标的下面加入一个空白行。

^\_或者 Control underline——在当前的光标位置拆分一行。

@J——把当前行和下一行合并。

^]——匹配括号。

@1~@3——设置文件标号，每个文件最多可以设置 3 个文件标号。

#@1~#@3 (Shift+Alt+1~Shift+Alt+3)——跳转到定义的文件标号位置中。

^@(Control+At 符号)——在光标所在位置写入当前系统的日期和时间。用户可以通过 editorcfg 实用程序来配置日期和时间的格式。

Control+Break——在 EDITOR 中，按下 Control+Break 将停止诸如打印、排序、查找/替换和递归宏的执行。

## 程序代码

### 【程序 188】 文本编辑器

#### (1) 结构性函数

结构性函数主要在 main.c 文件中定义。整个主函数的流程非常简单，如图 188-10 所示。

这些结构性函数用于初始化图形界面和各个内存区域，定义程序的行为，并且接收用户的输入、格式化命令、操作命令等，最后在程序结束的时候释放创建的内存区域和结构。

#### (2) 文件操作函数

文件操作函数在 FILE.C 文件中定义。文件操作包括了关于文件的 I/O 的函数。在 EDITOR 中，文件通过一个双向链表管理。每个链表中节点都有指向前一个和后一个节点的指针。每个节点还有一个指针指向一行文本，一个行宽度变量，还有一个脏位指示器。用一个精确的计数器统计每行文本的字符数。

#### (3) 查找替换函数

查找替换函数在 FINDREP.C 文件中定义。

在程序中使用的是 Boyer-Moore 文本查找算法。具体的算法可以参见论文 Robert S. Boyer and J Strother Moore, "A fast string searching algorithm.", Communications of the ACM\_ 20 (No. 10): 762-772, 1977。

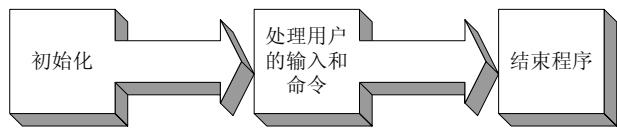


图 188-10 程序流程图

```
/*
 * 作用：建立并且执行查找操作
 * 参数：window 为当前窗口的指针
 */
int find_string( WINDOW *window )
{
    int direction;
    int new_string;
    char pattern[MAX_COLS]; /* 想要查找的文本 */
    long found_line;
    long bin_offset;
    line_list_ptr ll;
    register WINDOW *win; /* 把文件指针放到一个寄存器中 */
    int rcc;
    int old_rcol;
    int rcol;

    switch (g_status.command) {
        case FindForward :
            direction = FORWARD;
```

```

    new_string = TRUE;
    break;
case FindBackward :
    direction = BACKWARD;
    new_string = TRUE;
    break;
case RepeatFindForward1 :
case RepeatFindForward2 :
    direction = FORWARD;
    new_string = bm.search_defined != OK ? TRUE : FALSE;
    break;
case RepeatFindBackward1 :
case RepeatFindBackward2 :
    direction = BACKWARD;
    new_string = bm.search_defined != OK ? TRUE : FALSE;
    break;
default :
    direction = 0;
    new_string = 0;
    assert( FALSE );
    break;
}
win = window;
entab_linebuff( );
if (un_copy_line( win->ll, win, TRUE ) == ERROR)
    return( ERROR );
/* 得到搜索文本，上次的搜索文本做为缺省值*/
if (new_string == TRUE) {
    *pattern = '\0';
    if (bm.search_defined == OK) {
        assert( strlen( (char *)bm.pattern ) < MAX_COLS );
        strcpy( pattern, (char *)bm.pattern );
    }

    /*
    * 查找
    */
    if (get_name( find4, win->bottom_line, pattern,
        g_display.message_color ) != OK || *pattern == '\0')
        return( ERROR );
    bm.search_defined = OK;

    assert( strlen( pattern ) < MAX_COLS );

    strcpy( (char *)bm.pattern, pattern );
    build_boyer_array( );
}

rc = OK;
if (bm.search_defined == OK) {
    old_rcol = win->rcol;
    if (mode.inflate_tabs)
        win->rcol = entab_adjust_rcol( win->ll->line, win->ll->len,
win->rcol );
}

```

```

update_line( win );
show_search_message( SEARCHING, g_display.diag_color );
bin_offset = win->bin_offset;
if (direction == FORWARD) {
    if ((ll = forward_boyer_moore_search( win, &found_line, &rcol )) != NULL) {
        if (g_status.wrapped && g_status.macro_executing)
            rc = ask_wrap_replace( win, &new_string );
        if (rc == OK)
            find_adjust( win, ll, found_line, rcol );
        else
            win->bin_offset = bin_offset;
    }
} else {
    if ((ll = backward_boyer_moore_search( win, &found_line, &rcol )) != NULL) {
        if (g_status.wrapped && g_status.macro_executing)
            rc = ask_wrap_replace( win, &new_string );
        if (rc == OK)
            find_adjust( win, ll, found_line, rcol );
        else
            win->bin_offset = bin_offset;
    }
}
if (g_status.wrapped)
    show_search_message( WRAPPED, g_display.diag_color );
else
    show_search_message( CLR_SEARCH, g_display.mode_color );
if (ll == NULL) {
/*
* 没有找到
*/
    if (mode.inflate_tabs)
        win->rcol = old_rcol;
    combine_strings( pattern, find5a, (char *)bm.pattern, find5b );
    error( WARNING, win->bottom_line, pattern );
    rc = ERROR;
}
show_curl_line( win );
make_ruler( win );
show_ruler( win );
} else {
/*
* 没有定义查找模式
*/
    error( WARNING, win->bottom_line, find6 );
    rc = ERROR;
}
return( rc );
}

```

#### (4) 窗口函数

窗口函数在 WINDOW.C 文件中定义。在窗口函数定义了窗口的行为，比如对窗口的拆分、合并，对窗口的大小进行调整，在不同窗口中切换等。

## (5) 块函数

块函数在 BLOCK.C 文件中定义。

```
/*
 * 作用：记录块的起始位置
 * 参数：window 为指向当前窗口的指针
 * 注意：假定用户将会定义块的开始和结束。若采用混和方式，那么块的类型被认为是当前的块类型
 */
int mark_block( WINDOW *window )
{
    int type;
    int num;
    long lnum;
    register file_infos *file;           /* 临时文件 */
    register WINDOW *win;                /* 把当前窗口指针放到一个临时寄存器里面 */
    int rc;

    win = window;
    file = win->file_info;
    if (win->rline > file->length || win->ll->len == EOF)
        return( ERROR );
    if (g_status.marked == FALSE) {
        g_status.marked = TRUE;
        g_status.marked_file = file;
    }
    if (g_status.command == MarkBox)
        type = BOX;
    else if (g_status.command == MarkLine)
        type = LINE;
    else if (g_status.command == MarkStream)
        type = STREAM;
    else
        return( ERROR );

    rc = OK;
    /*
     * 仅对于一个文件定义块，用户可以在这个文件的任何窗口进行操作
     */
    if (file == g_status.marked_file) {

        /*
         * 不管块的模式，标识块的起始和中止位置
         */
        if (file->block_type == NOTMARKED) {
            file->block_ec = file->block_bc = win->rcol;
            file->block_er = file->block_br = win->rline;
        } else {
            if (file->block_br > win->rline) {
                file->block_br = win->rline;
                if (file->block_bc < win->rcol && type != STREAM)
                    file->block_ec = win->rcol;
                else
                    file->block_bc = win->rcol;
            } else {

```

```

        if (type != STREAM) {
            file->block_ec = win->rcol;
            file->block_er = win->rline;
        } else {
            if (win->rline == file->block_br &&
                win->rline == file->block_er) {
                if (win->rcol < file->block_bc)
                    file->block_bc = win->rcol;
                else
                    file->block_ec = win->rcol;
            } else if (win->rline == file->block_br)
                file->block_bc = win->rcol;
            else {
                file->block_ec = win->rcol;
                file->block_er = win->rline;
            }
        }
    }

/*
 * 如果用户标识的块的终止位置在起始位置前，那么交换两个位置
 */
if (file->block_er < file->block_br) {
    lnum = file->block_er;
    file->block_er = file->block_br;
    file->block_br = lnum;
}

/*
 * 如果用户标识的块的终止列在起始列前，那么交换两个位置
 */
if ((file->block_ec < file->block_bc) && (type != STREAM ||
    (type == STREAM && file->block_br == file->block_er))) {
    num = file->block_ec;
    file->block_ec = file->block_bc;
    file->block_bc = num;
}
}

/*
 * 如果块类型已经被定义，但是如果用户使用混和模式，那么块的类型被置为当前块的类型
 */
if (file->block_type != NOTMARKED) {
/*
 * 如果块的类型是矩形块，那么要保证左上角小于右下脚
 * 如果块的类型是 stream 块，那么保证起始列小于中止列
 */
    if (type == BOX) {
        if (file->block_ec < file->block_bc) {
            num = file->block_ec;
            file->block_ec = file->block_bc;
            file->block_bc = num;
        }
    }
}

```

```

}

assert( file->block_er >= file->block_br );

file->block_type = type;
file->dirty = GLOBAL;
} else {
/*
* 已经定义好块
*/
    error( WARNING, win->bottom_line, block1 );
    rc = ERROR;
}
return( rc );
}

```

块操作可以在文件内部也可以在文件间执行。行块的移动操作对象是光标所在行文本。矩形块和 stream 块的移动操作从光标所在的列开始。按 Control+Break 可以阻止绝大多数的矩形块操作。其他的块操作因为执行速度很快，几乎没有机会停止它们。



## 归纳注释

EDITOR 编辑器是一个功能非常强大的文本编辑器，它支持目前流行的文本编辑器的绝大部分功能。读者可以在该程序的基础上加以改进，做出功能更加完善的程序。

前面介绍的几个函数都是 EDITOR 编辑器中几个比较重要的函数，系统是由近一百个函数组成，有兴趣的读者可以仔细阅读光盘里的源程序。

# 实例 189 图书管理系统



## 实例说明

本实例将介绍如何通过 C 语言开发一个图书管理系统。程序运行界面如图 189-1 所示。

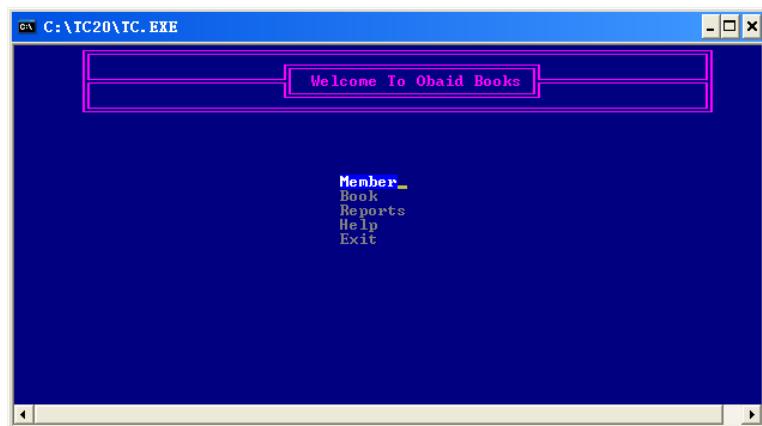


图 189-1 图书管理系统主界面

用户通过上下方向键可以选择具体的功能菜单。系统提供了 5 个功能菜单，分别是“Member”、“Book”、“Reports”、“Help” 和 “Exit”。

“Member” 表示会员管理。选中该选项，单击回车，可以进入到其子菜单中，如图 189-2 所示。



图 189-2 会员管理功能菜单

“Member” 选项提供了 4 个子选项，分别是 “Add New Member”（添加新会员）、“Renew Existing Member”（继续使用户账号有效）、“Issue Duplicate I-card”（签发借书卡副本）以及 “Back”（返回）。

选中 “Add New Member” 进入添加新会员界面，如图 189-3 所示。

在主界面中选中 “Book” 选项，可以进行图书管理，如图 188-4 所示。

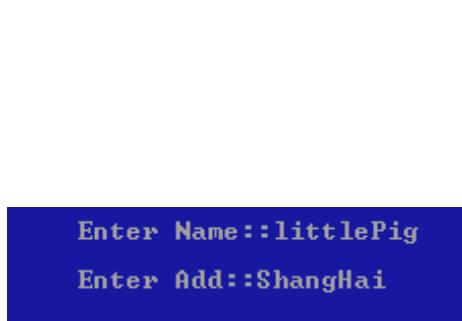


图 189-3 添加新会员信息

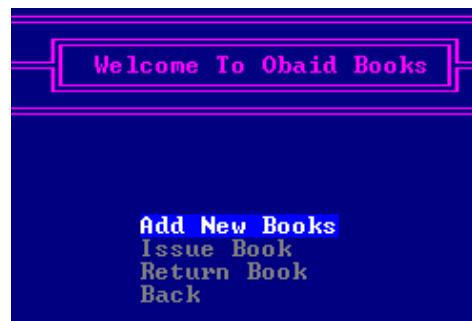


图 189-4 图书管理功能菜单

在图书管理功能中，提供了 4 个功能选项，分别是 “Add New Books”（添加新图书）、“Issue Book”（借出图书）、“Return Book”（还书）以及 “Back”（返回）。

选中 “Add New book”，进入添加新图书界面，如图 189-5 所示。

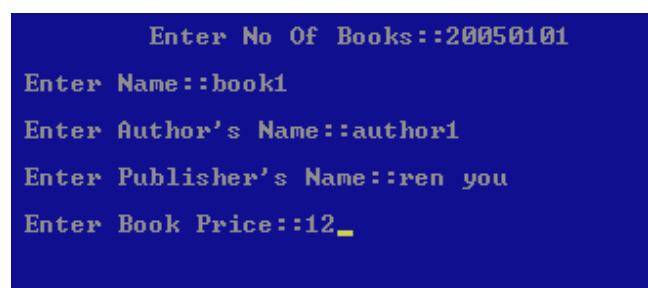


图 189-5 添加新图书界面

选中 “Issue book” 选项后按回车，出现如图 189-6 所示的借书界面。

注意：“Book id” 和 “Membership id” 要到 “Reports”（报表）中查询，这些编号由系统自动实现。在输入 “Book id” 时，首先需要输入 “2005”，回车后输入 “1”，然后再次回车输入 “1”。

选中 “Return book” 选项后按回车，出现如图 189-7 所示的界面。

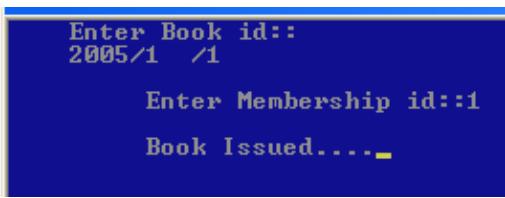


图 189-6 借书界面

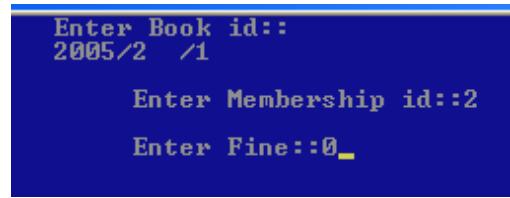


图 189-7 还书界面

回到主界面，选择“Reports”选项，可以查看各种报表，如会员报表、图书报表以及借书还书报表等，报表界面如图 189-8 所示。



图 189-8 报表管理界面

在报表管理中，提供了 3 种报表查看，“Member Details”（查看会员报表），“Books Details”（图书报表）以及“Transactions Details”（业务报表）。

会员报表如图 189-9 所示。

***** List Of All Members *****					
Id	Name	Address	Member Since	Expiration	
1	1 forway	1 beijing	3 /2 /2005	3 /8 /2005	
2			3 /2 /2005	3 /8 /2005	

图 189-9 会员报表

图书报表如图 189-10 所示。

***** List Of All Books *****					
BookId	Book Name	Author Name	Publisher's Name	Price	
200511	book1	author1	ren you	12	
200521	littlepig	shanghai	jixie	23	

图 189-10 图书报表

业务报表如图 189-11 所示。

***** List Of All Transactions *****						
Trans ID	Mem Id	Book Id	Issue Date	Return Date	Fine	Reason
1	1	000	3 /2 /2005	0 /0 /0	500.00	A
2	2	000	3 /2 /2005	0 /0 /0	500.00	A
3	1	200511	3 /2 /2005	3 /2 /2005	0.00	R
4	2	200521	3 /2 /2005	3 /2 /2005	0.00	R

图 189-11 业务报表

## 实例解析

由于 C 语言中操作数据库不是很方便，因此大部分的数据存储都是通过文件来进行的。在本实例

中，数据存储采用了3个文件来实现。程序第一次运行的时候便会自动在目录下创建。**①BOOK.DAT**，用于保存图书信息；**②MEMBER.DAT**，用于保存会员信息；**③TRANS.DAT**，用于保存业务信息，即结束还书信息。

在实例中，会员信息、图书信息以及业务信息分别由结构体来表示，如下所示：

```
struct member //会员管理结构体
{
    int mid;
    char mname[20],madd[30];
    struct msince
    {
        int day,mon,year;
    } ms;
    struct mexpir
    {
        int day,mon,year;
    } me;

} M;

struct book //图书管理结构体
{
    struct bkid
    {
        int gno,bno,no;
    } b;
    char bname[20],author[15],pub[20];
    int price;
} B;

struct transaction //业务管理结构体
{
    int mid,tid;
    struct bookid
    {
        int gno,bno,no;
    } b;
    struct issued
    {
        int day,mon,year;
    } i;

    struct returned
    {
        int day,mon,year;
    } r;

    float fine;
    char reason;
} T;
```

而菜单的定义是通过字符串数组定义的：

```
char *mainmenu[ ]={  
    "Member",  
    "Book",  
    "Reports",  
    "Help",  
    "Exit"  
};  
  
char *memmenu[ ]={  
    "Add New Member",  
    "Renew Existing Member",  
    "Issue Duplicate I-Card",  
    "Back"  
};  
  
char *bookmenu[ ]={  
    "Add New Books",  
    "Issue Book",  
    "Return Book",  
    "Back"  
};  
  
char *rptmenu[ ]={  
    "Members Details",  
    "Books Details",  
    "Transactions Details",  
    "Back"  
};
```

## 程序代码

### 【程序 189】 图书馆管理系统

由于本系统比较大，涉及到的文件比较多，读者可参考光盘中的源代码进行学习。

## 归纳注释

本实例采用了多个文件来保存数据结构，用户可以改进程序，采用单个文件或者数据库来实现各项功能。另外在设计数据库时候，可以将会员编号或者图书编号设计为自定义的，这样用户就可以很方便地借书或者还书了。

# 实例 190 进销存管理系统

## 实例说明

本实例将编制一个功能比较完善的进销存管理程序。运行该程序，系统主界面如图 190-1 所示。



图 190-1 进销存管理系统主界面

在系统的主界面中提供了 11 个功能选项，同时在主界面上会显示当前的时间。下面依次演示系统中的重要模块。

选择功能菜单“1”进入查看商品信息界面，如图 190-2 所示。

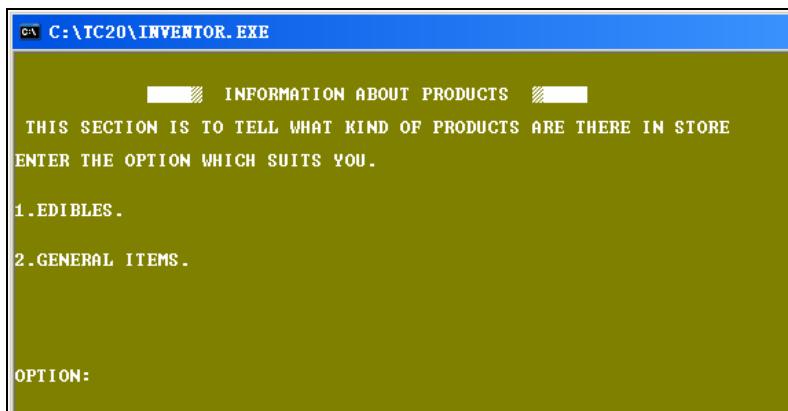


图 190-2 进销存管理系统主界面

在界面中会显示各种商品，可以单击商品编号，查看该类商品的详细信息，如图 190-3 所示。

ID :	ITEM	COMPANY	RACK NO
ID : 133.	MILO	MITUCHELS	24
ID : 134.	KEY BRAN	PAK PURE	25
ID : 135.	TANG	HALEEB	26
ID : 136.	ALWAYS	WONDER	27
ID : 327.	PEPPARS	JUNAID	28
ID : 138.	JHERBAL	SONS	29
ID : 139.	PEPSI	MAJEED	30
ID : 130.	ORAGD	JAFFAR	31
ID : 131.	ICECREAM	POLKA	FREEZER 32
*****			
ID : 143.	SHAMPO	MITUCHELS	24
ID : 144.	AGRI	PAK PURE	25
ID : 145.	TIMO	HALEEB	26
ID : 146.	WASHSN	WONDER	27
ID : 147.	LACE	JUNAID	28
ID : 428.	YARDLY	SONS	29
ID : 149.	MUSK	MAJEED	30
ID : 140.	BUTTER	JAFFAR	31
ID : 141.	IMPERIAL	POLKA	FREEZER 32
*****			
Press enter to return to main menu			

图 190-3 商品详细清单界面

在主界面中选择“2”，进入商品信息录入界面，如图 190-4 所示。

商品信息录入完毕，会提示是否继续录入，选择“n”，回到主界面。

在主界面上选择“3”，进入商品销售界面，如图 190-5 所示。



图 190-4 商品信息录入界面



图 190-5 商品销售界面

在主界面中选择“4”，进入商品查询界面，如图 190-6 所示。

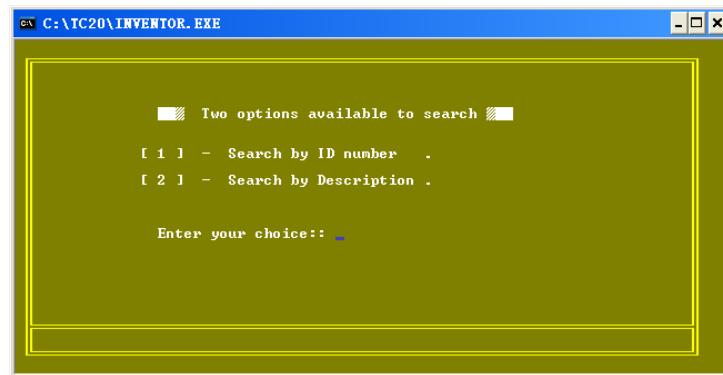


图 190-6 商品查询界面 1

系统提供了两种查询方式，即根据商品编号查询以及根据商品描述查询，如果选择“1”表示根据商品编号查询，在查询界面上输入查询编号，可以显示出查询结果，如图 190-7 所示。



图 190-7 商品查询界面 2

在主界面上选择“5”，可以删除商品信息，同样可以根据商品描述或者根据商品编号删除商品信息，界面类似商品查询界面。

在主界面上选择“6”，可以查看销售、入库以及利润报表，如图 190-8 所示。

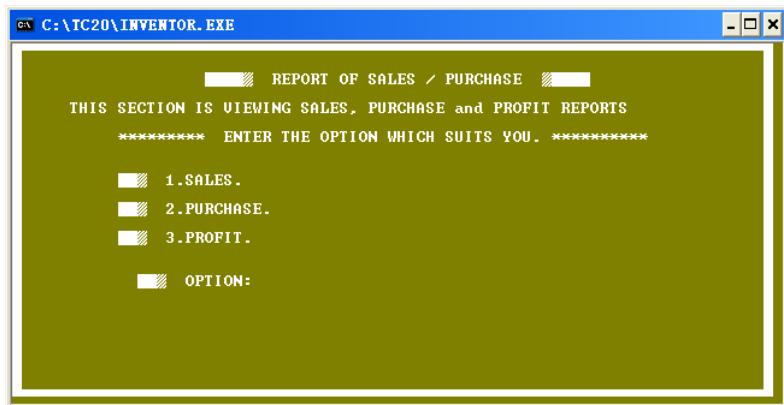


图 190-8 报表显示界面

在图 190-8 所示中，选择“1”可以查看销售报表，选择“2”可以查看入库报表，选择“3”可以查看利润报表。销售报表如图 190-9 所示。

VIEW OF SALES					
PRODUCT ID.	NAME.	SALE PRICE.	QUANTITY.	TOTAL PRICE	
1	Masdf&OE‡	\$34.00	23	\$782.00	
123	good	\$21.00	0	\$0.00	
3	ok	\$4.00	4	\$16.00	

Press Enter to go to MAIN MENU .....

图 190-9 销售报表界面

在主界面中选择“7”可以打印记录。

在主界面中选择“8”可以图形化显示一些数据，如库存数量，销售利润等，如图 190-10 所示。

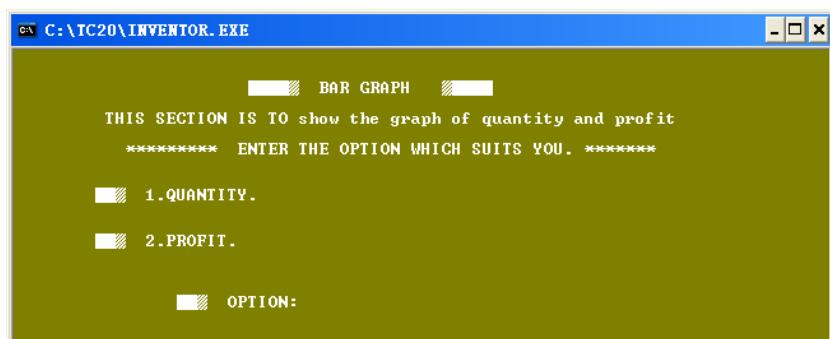


图 190-10 图形显示界面

在图 190-8 中选择“1”可以图形化显示商品库存，如图 190-11 所示。

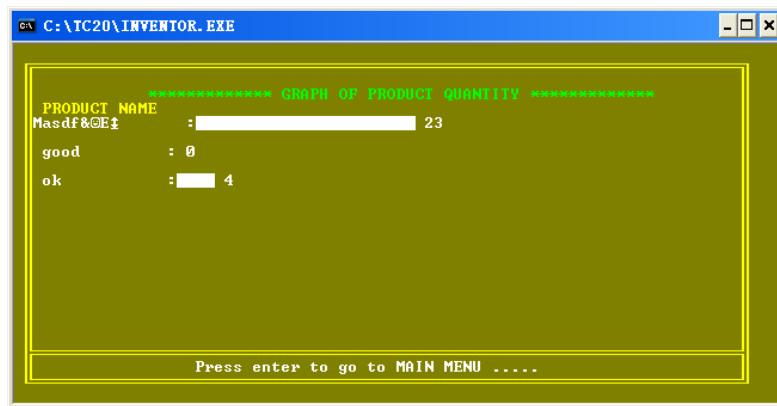


图 190-11 库存的图形显示界面

## 实例解析

设产品信息以一个产品一条记录的形式存储在文件中，每个产品记录包含的信息有：产品名称、描述、数量和价格等。程序将具有以下几项功能。

- (1) 显示产品的信息。
- (2) 增加商品库存，相当于进货。
- (3) 减少商品库存，相当于销售。
- (4) 查找商品库存内容。
- (5) 从数据库中删除商品信息。
- (6) 察看销售，购买，利润报告。
- (7) 打印报告。
- (8) 数量或者利润值的图示。



## 程序代码

### 【程序 190】 进销存管理系统

//本实例源码参见光盘



## 归纳注释

除了主函数以外，将各个模块细化，编写了一个个具有独立功能的函数。从而让主函数显得整洁。通过函数的调用，也方便了对每个功能模块的修改。如果程序需要添加功能，或者删除功能也可以仅对每个小函数进行改进。