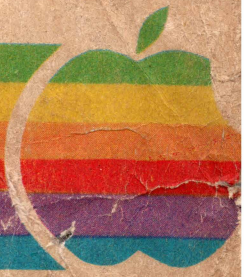
张世英 编

苹果-Ⅱ BASIC 程序设计



北京师范大学出版社

封面设计：田　苗

科　　目　91—70
统一书号：13243·77
定　　价：3.70 元

苹果-II BASIC程序设计

张世英 编

北京师范大学出版社

内 容 简 介

本书以苹果-Ⅱ微型计算机为背景，详细地介绍了BASIC语言的各种语法规则、程序设计方法及上机操作的步骤。这些内容对各种微型机都是适用的。本书后六章以较大篇幅介绍了苹果-Ⅱ微型机图形绘制与游戏程序设计的方法；磁盘操作系统与文件、6502指令系统及监控命令等。这对苹果-Ⅱ的用户充分开拓苹果-Ⅱ上所配的丰富软件会有很大帮助。

本书按教材形式写出，内容由浅入深，由易到难，凡具有中等以上文化程度的读者即可阅读。各章后均附有习题，并在书末给出了参考答案。书末附录把书中有关内容作了归纳整理，以供参考。

苹果-Ⅱ BASIC程序设计

张世英 编

北京师范大学出版社出版
新华书店北京发行所发行
湖南省新华印刷二厂印刷

开本：850×1168 1/32 印张：16.75 字数：414千
1985年4月第一版 1985年4月第一次印刷
印数：1—97,000
统一书号：13243·77 定价：3.70

编者的话

本书是以苹果-Ⅱ微型计算机为背景来介绍BASIC语言的。苹果-Ⅱ微型机软件非常丰富，其功能很强，目前这种机器在国内的用户已占首位。苹果-Ⅱ上所配的BASIC语言除个别语句和命令外，其基本语句在别的微型机上也是适用的。

书中以介绍算法语言为主，也提供了一些常用算法。但这些算法不需要高深的数学知识，有一般中等文化水平也就可以了。书中后一部分介绍的苹果-Ⅱ图形绘制、游戏程序的设计方法、磁盘操作系统与文件、6502指令系统及监控命令、苹果-Ⅱ整数BASIC等仅适用于苹果机，内容比较全面而深入，其中相当一部分从现有资料上是查找不到的。

书中力求给读者以准确的概念，并对一些常见错误给出了必要的分析与说明。同时，书中还以相当的篇幅介绍了调试手段与纠正错误的措施。这些内容不仅对一般读者有用，对教授这门课的教师以及指导上机的教师，也是很有用处的。

本书内容，有一定的厚度。本书以"计算机盲"作为对象开始叙述，除了介绍语言以外，逐步引入计算机的一些更深入的概念，这不仅能引起读者进一步的求知欲，而且对进一步学习计算机的其它课程，有一定的帮助。

本书内容在叙述方法上力求由浅入深，通俗易懂；内容安排上前后连贯，由易到难。各章均配有一定量的习题并在书末给出了答案。这些习题和答案有利于读者对各章内容的复习和巩固。

在本书编写过程中，北京师范大学现代化教育技术研究所荣树熙副教授、北京师范大学出版社编辑部的同志提出过不少宝贵

意见，在此表示感谢。由于编者水平有限，加上编写时间仓促，错误缺点在所难免，敬请广大读者批评指正。

作　者

目　录

第一章　计算机的一般知识……（1）
§1.引言……（1）
§2.电子计算机的基本结构……（4）
§3.计算机中数的表示……（7）
§4.计算机语言……（11）
第二章　BASIC语言的基本概念……（18）
§1.BASIC语言的特点……（18）
§2.BASIC程序的构成……（20）
§3.常量与变量……（22）
§4.标准函数……（26）
§5.运算符和表达式……（29）
§6.上机操作提要……（32）
第三章　数据的输入与输出……（42）
§1.必要的说明……（42）
§2.打印语句(PRINT)……（46）
§3.赋值语句(LET)……（52）
§4.键盘输入语句(INPUT)……（56）
§5.读数语句和置数语句(READ和DATA)……（61）
§6.恢复数据区语句(RESTORE)……（62）
§7.三种取得数据的语句的比较……（65）
§8.算术函数的使用……（67）
§9.PR#与IN#……（72）

第四章　程序流向的转移……（75）
§1.程序框图……（75）
§2.无条件转向语句(GOTO) ……（78）
§3.条件转移语句(IF…THEN)的概念 ……（81）
§4.条件式……（88）
§5.应用程序举例……（94）
§6.暂停语句(STOP)及调试程序的一般方法 ……(101)
§7.注释语句(REM) ……(102)
第五章　程序的循环……(107)
§1.循环语句的基本概念……(107)
§2.循环语句应用举例……(117)
§3.多重循环……(122)
§4.循环语句的深入讨论……(128)
第六章　程序模块……(146)
§1.子程序……(146)
§2.开关语句与出错处置……(163)
§3.用户自定义函数……(167)
§4.应用程序举例……(170)
第七章　数组与字符串函数……(184)
§1.下标变量与数组的概念……(184)
§2.数组说明语句(DIM)……(189)
§3.STORE与RECALL ……(191)
§4.应用程序举例……(192)
§5.与字符串有关的几个函数……(214)
第八章　图形绘制……(223)
§1.文本方式和文本方式下的图形绘制……(223)
§2.低清晰度图形……(229)
§3.高清晰度图形……(241)

§4.用图形表方法绘制高清晰度图形……(252)
第九章　唱歌与游戏程序的编写……(267)
§1.唱歌程序的编写……(267)
§2.游戏程序的编写……(275)
§3.趣味教学程序一例……(296)
第十章　磁盘操作系统命令与文件……(308)
§1.软磁盘……(308)
§2.DOS命令简介……(309)
§3.顺序文本文件……(326)
§4.自动APPLE:EXEC 命令的使用 ……(342)
§5.随机文本文件……(347)
第十一章　6502指令系统及监控命令简介……(355)
§1.6502的寄存器……(356)
§2.6502指令分类……(357)
§3.6502的寻址方式……(362)
§4.补充说明……(372)
§5.几个简单的程序例……(380)
§6.监控命令简介……(383)
第十二章　深入的讨论……(397)
§1.DOS自举与内存分配……(397)
§2.系统的建立……(401)
§3.程序(或命令)的输入……(405)
§4.程序(或命令)的运行……(415)
§5.特殊技巧举例……(424)
§6.同系统有关的几个命令和函数……(438)
第十三章　整数BASIC简介……(445)
§1.整数BASIC的特点……(445)
§2.整数BASIC的语句和命令……(449)

§3.整数BASIC程序例……(454)
§4.整数BASIC解释执行过程……(453)
习题参考答案……(476)
附录……(516)
附录1.ASCII码表 ……(516)
附录2.保留字表……(519)
附录3.出错信息表……(524)
附录4.专用单元表……(527)

第一章　计算机的一般知识

§1. 引　　言

人类在同大自然的斗争中，创造并逐步发展了计算工具。

早期的计算工具，主要是人们自身的附属物(如手指、脚指)或周围可数的有形物体(如石子、绳结、小棒等)。这些计算工具都只是天然的"计算器"，计算也多属计数。人类经过加工制造出来的第一种计算工具，则是我国唐末出现的算盘。随着生产的发展，计算日趋复杂，先后又出现了许多较先进的计算工具。比如机械计算机(1642年)、计算尺(1654年)、手摇计算机(1887年)乃至电动计算机等。然而，这些计算工具的致命弱点是不能自动地连续地进行计算，也不能存放大量的中间结果。把程序控制引入计算机的第一个人是巴贝奇 (Charles Babbage)。他花费了近四十年的时间构思了一台分析机，具备了现代计算机应具备的一切：用许多刻有数字的轮子来存贮数据，通过齿轮和轮子的旋转来进行计算，用一组齿轮和扛杆构成的装置来传送数据，用穿孔卡片输入程序和数据，用穿孔卡片和打印机输出计算结果。第一台实用的卡片程序控制计算机是霍勒力斯 (Herman Hollerith) 的杰作，他设计了一台用卡片存贮大量数据并用卡片进行控制的数据处理机，1890年曾用于人口普查工作，对六千二百万人的数据进行记录、编辑和制表处理，共花了两年时间。

计算工具的现代化和飞速发展是从本世纪开始的。1925年左右，布什 (Vannever Bush) 领导制造了第一台模拟式计算机。

1937年艾肯(Howard Aiken)写了《自动计算机建议》，并从1939年起经过5年努力制成了第一台数字式自动计算机Mark I。1943年宾夕法尼亚大学开始研制的数字积分机和计算机ENIAC(Electronic Numerical Integrator and Caleulator)则是有史以来第一台数字式电子计算机。该机于1945年12月投入运行，1946年2月正式交付使用。ENIAC共用了一万八千多只电子管，一千五百多个继电器，耗电一百五十瓩，占地一百七十平方米，重一百三十吨，运算速度每秒五千次加法运算。领导研制这部机器的有该校的工程师埃克特（J.Presper Eckert）和物理学家毛希利（John Mauchly)。这台机器的出现，被誉为新的工业革命的开始。

从"ENIAC"问世至今还不到四十年，电子计算机以越来越迅猛的势头发展着。据国外报导，电子计算机每五至八年运算速度提高十倍，体积缩小十倍，而成本却降低了十倍。就计算机采用的物理器件来说，已经历了电子管、晶体管、集成电路和大规模集成电路四代。由于大规模集成电路技术的发展，可以把一个小型计算机的运算器和控制器制作在一块很小的芯片上，成为一个微处理器，并以它为主体构成微型计算机。微型计算机的崛起，被称为计算机的"第二次革命"。它继承和发展了小型计算机的先进技术，又以其结构灵活紧凑、品种繁多、应用范围广阔、价格低廉、生产周期短、维修方便等优势，形成了对小型机的威胁。就连一些专门生产巨型机的厂家，也相继推出了自己的微型机，以争夺这个前景无量的市场。

我国计算机的研制工作，正式起步于1956年。1958年试制成功了第一代电子管数字计算机 DJS-1。1965年又研制成功第一台大型通用晶体管计算机。1971年第一台集成电路计算机 TQ-16问世。至今，我国计算机已换了三代，目前处于三代半状况。由于党和政府的重视，我国计算机事业正在形成一个新的发展时期。

计算机之所以能得到如此迅猛的发展并成为新的技术革命的

重要标志，主要地是由于计算机有着强有力的功能和极为广阔的应用领域。

电子计算机有以下几个特点：

1.运算速度快。比如气象“日预报”，用手摇计算机或电动计算机来计算，约需一、二个星期；而用一般的中型计算机则只需几分钟。

2.精度高。一般电子计算机可以有十几位有效数字，如果降低运算速度，有效位数还可以增加。

3.具有“记忆”和判断功能。可以记录程序、原始数据和中间结果，还可以对某些信息进行判断并作出相应的后续处理。

4.能自动地进行控制，不必人工干预。

5.电子计算机是一个不怕麻烦、不闹情绪的计算工具，是人类最忠实、最可靠的朋友。

电子计算机是在人类生活中大量的繁琐的数值计算的需要下应运而生的，所以早期的电子计算机主要用于数值计算。也就是说，输入和处理的对象是数值，处理的算法是数值计算方法，输出的结果也是数值。但是，电子计算机诞生不久，应用的范围很快就突破了这个框框。人们发现，计算机除能进行数值处理之外，还可以处理诸如字母、符号、单据、表格、资料、图形、图象、信号乃至文字、语言、声音等，随之也发展了各种非数值的算法和相应的数据结构，比如排序、查找、插入、删除、字符串匹配、树和图的处理等。计算机的应用从数值计算发展到非数值应用是计算机发展史上的一个跃进，大大地拓宽了计算机应用的领域，使计算机不再是少数人的珍品，而进入人类社会的各种行业以至于家庭之中。

随着计算机应用范围的扩大，计算机软件也在突飞猛进地发展着。系统软件的功能越来越强；程序语言越来越丰富；各种应用软件也越来越多。目前，软件理论日臻成熟，软件研制工作的

工程化，软件产品的标准化、商品化，已成为一股新的潮流。

电子计算机的发展与普及，将对人类社会产生越来越深远的影响。

§2. 电子计算机的基本结构

电子计算机，从原理上讲分为两大类：电子模拟计算机（用来处理连续量的）和电子数字计算机(用来处理非连续量的)。从用途上讲，又可分为专用计算机和通用计算机。我们这里只介绍通用数字微型计算机。

通用数字计算机的基本结构，大体上有以下几个部分：控制器、运算器、内存贮器、输入设备和输出设备。前三者又合称主机，后两者又称作外部设备。现就这五个部分的功能，作一些简要说明。

1.输入设备。输入设备是用来向主机输入原始数据和处理这些数据所使用的计算程序的设备。输入设备的种类很多，但在微型机上不外乎下列几种：

(1) 终端键盘。利用手指击键方法向计算机输入信息。用户自己写的程序和准备处理的数据，都可由键盘上敲入。

(2) 盒式磁带和磁盘。盒式磁带或磁盘，实际上也是存贮信息的，因为它们都是主机之外的设备，所以也称为外存贮器。外存贮器上的信息，也可以输入到机器中去。

(3) 模—数(A/D)转换器。它可以将连续变化的模拟量（如电流、电压、长度、角度等)转换为数字量，送入计算机内。

此外，图形输入板、声音输入装置等，实际上是专用的模—数转换器，同样可以为计算机输入信息。

2.输出设备。输出设备是用来输出计算结果或其它信息的。常用的输出设备有：

(1) 显示终端。用以显示计算机的有关信息，如用户从键盘上敲入并为机器接收的字符、程序清单、机器向用户的提示、程序运行时的输出(包括数字、文字或图形)等。

(2) 打印机。显示终端上可显示的东西几乎都可以由打印机打印到纸上。

(3) 磁带和磁盘。可以用来存贮程序和数据(包括数字、文字、图形和声音信息等)。

(4) 数—模(D/A)转换器。将机器输出的数字量转换为模拟量。

绘图仪、声音输出装置等也是专用的数—模输出设备。

3.主存贮器，即内存贮器(简称内存)。它用来存放原始数据、处理这些数据的程序以及计算结果（包括中间结果，也包括图形和声音信息等)。系统程序也放在内存中。

内存贮器分为一个个单元，好似一间间房子，并按顺序编了号码(从0号开始)，通常又称为一个个地址。机器中的所有信息都以一定的规则存放在内存的一个个单元中。

对任何一个单元来说，它很象从左到右顺序安放的一排灯泡，每个灯泡代表一位数字：灯泡点亮代表1，灯泡熄灭代表0。于是这一排灯泡就可表示由0和1构成的一个数。这排灯泡的个数，就称为位数或字长。目前，一般微型计算机的内存贮器是用半导体器件组成的电路制成的，称为半导体存贮器。字长，一般取决于微处理器的字长。我们介绍的苹果机，微处理器为6502，其字长为8位。所以苹果机是八位机。

一般微型机的内存贮器又分为两部分。一部分是随机存贮器(RAM)，每个单元的数据是可以改变的，而且关电以后所有信息都会自动消失。这类存贮器是用户可以使用的空间。另外，还有一部分是只读存贮器(ROM)，每个单元中的信息是固化了的，用户只可读出使用，但无法使其改变。任何时侯，只要接通电源，

这些信息就建立好了。

4.运算器。运算器是计算机进行信息加工的场所，所有算术运算、逻辑运算等都在这里进行。就象用算盘算题时一样，它只能放当前被操作的和操作完的一个数据，中间结果一般要送内存中保存起来，以备以后使用。所以，没有内存，单靠运算器是无所作为的。

5.控制器。它是用来实现计算机各部分协调动作使计算过程自动进行的装置。也就是说，它是计算机内的指挥部。

控制器可以向计算机的其它部件发出信号，控制数据的传输与加工；同时，控制器也接收其它部件送来的信号，以便调整其控制功能。

所以，在计算机工作时，有两种信息流：控制流与数据流，由控制流控制数据流的传输与加工，完成所有的计算动作。

图1—1是典型的计算机框图，其中实线代表数据流，虚线代表控制流，箭头代表信息流动的方向。

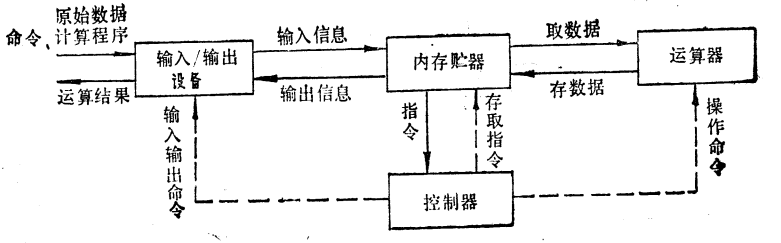


图1—1　计算机结构示意图

图1—2是苹果机基本系统配置的示意图。

苹果机的外部设备，比这个示意图上画的要多得多。那些设备将会大大扩充苹果机的用途。

附带说一下，这里介绍的运算器、控制器、内存贮器、输入

输出设备等，都是一些看得见、摸得着的“硬”东西，所以又称它

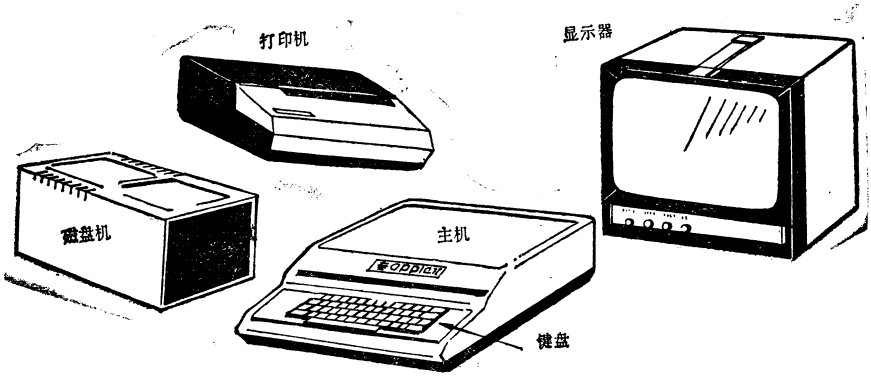


图1—2　苹果-Ⅱ微型机基本系统配置图

们为“硬件”或“硬设备”。相对地，那些在计算机工作过程中必不可少的数据以及对这些数据进行处理的控制命令等信息流都是摸不着的“软”东西，通常称之为“软件”或“软设备”。

BASIC语言本身，属于软件的范畴。

§3．计算机中数的表示

日常生活中，我们非常习惯使用十进制计数法，可是在日常生活里我们也还会遇到一些别的进制，如二进制(两只为一双)、十二进制(十二只为一打，十二英寸为一英尺，十二个月为一年)、十六进制(中国老秤十六两为一斤)、二十四进制（二十四小时为一天)和六十进制(六十分为一小时，六十秒为一分)等等。在计算机内部，则采用二进制计数法。

为什么计算机要使用二进制计数法呢？这是因为电气元件中两种状态最容易实现(如电路的通断、电位的高低等)，也最为稳定，并且最容易实现对电路本身的控制。

在计算机里，一般以高电位代表1，低电位代表0。而二进制

的两个数基，就用0和1来表示。凡够 2 时，就向左进一位。比如十进制的2，用10表示；十进制的4，用100表示等。

把十进制的0到10用二进制表示出来，如下：

十进制数	二进制数
0	0
1	1
2	10
3	11
4	100
5	101
6	110
7	111
8	1000
9	1001
10	1010

二进制的算术运算规则是很简单的：

$0+0=0$	$0\times0=0$
$0+1=1$	$0\times1=0$
$1+0=1$	$1\times0=0$
$1+1=10$	$1\times1=1$

个位加法4条，个位乘法4条。而十进制个位加法规则有100条，乘法规则有81条。

用逻辑电路实现二进制数的运算，是极为方便的。

一、十进制数和二进制数的转换

把一个十进制数转换为二进制数，方法如下：把这个十进数反复地除以2，直到商为零，所得的余数(从末一位读起）就是这个数的二进制表示。

如十进制的11，反复用2除：

$$
\begin{array}{l|l}
2 & 11 \quad (1 \\
2 & 5 \quad (1 \\
2 & 2 \quad (0 \\
2 & 1 \quad (1 \\
 & 0
\end{array}
$$

用二进制表示，是1011

通常用　　$(11)_{10}=(1011)_2$表示。

其中括号外的小10和小2分别表示是十进制数或二进制数。

换句话说，把一个十进制数化成以2为底的指数形式，则它的系数(由高次到低次)就是其二进制表示的数。

像上面提到的十进制数11，换成以2为底的指数形式为：

$$11=1\times2^3+0\times2^2+1\times2^1+1\times2^0$$

把它的系数顺序排列起来，就是**1011**，这就是它的二进制表示。

一般地，如果一个十进制数可以表示为：

$$a_0\cdot2^n+a_1\cdot2^{n-1}+\cdots+a_{n-1}\cdot2^1+a_n\cdot2^0$$

则　$a_0a_1\cdots a_{n-1}a_n$

就是这个十进制数的二进制表示。

反之，一个二进制数如有以下形式：

$$a_0a_1\cdots a_{n-1}a_n$$

那么它的十进制表示，就是下式的计算结果：

$$a_0\cdot2^n+a_1\cdot2^{n-1}+\cdots+a_{n-1}\cdot2^1+a_n\cdot2^0$$

例如：

$$(1011)_2=1\times2^3+0\times2^2+1\times2^1+1\times2^0=(11)_{10}$$

这就表明，二进制数**1011**是十进制数的11。

换言之，将一个二进制数转换成十进制数的方法是：将这个二进制数的最末一位乘以2^0，倒数第2位乘以 2^1，……最后将各项相加即可。

二、十进制数与十六进制数的转换

在计算机内，所有的数都是用二进制表示的（电位的高低表示1或0）。然而，如果让人们对任何二进制数能象对十进制数那样一目了然，实在不是一件容易的事。这一方面由于人们用二进制数终归没有用十进制数来得那么普遍与习惯；另一方面，用二进制表示一个数时，其位数较长（比如大于十进制63的数，用二进制表示至少要6位），且每位数只有0和1可资区别。

为此，人们引进了八进制和十六进制的表示方法。

必须指出，八进制或十六进制并不是机器中的某一位有八种状态或十六种状态，在机器内仍是以二进制为基础的（每一位只可有两种状态），只是为着输入或输出一个二进制数的方便，对机器内的二进制数从右向左每三位分作一组，每组便可表示0—7的某个数，显然最右一组满8时要向左邻一组进1，于是这三位一组构成了八进制数。类似地，把二进制数从右到左每四位分作一组，每组表示0～15中的某个数，这一组一组就构成了十六进制数。目前用十六进制表示较为普遍。可以想见，把两个四位连在一起可以用两个十六进制数表示，其数值范围可为十进制数的0到255，共256个数，选用它们来代表所有字符的内部编码（以便用二进制数代表字符）是足够使用了。因此又把两个十六进制数作为一个基本单位，称作“字节”。在八位机里，一个字节正好是一个机器字。

十六进制数的表示法，0～9仍沿用十进制中的0～9，接下去的六个数依次用A、B、C、D、E、F表示。

因此，十进制的 0～16表示成二进制数和十六进制数，如表1—1所示。

十进制数和十六进制数之间的转换，类似于十进制数和二进制数之间的转换。

把一个十进制数反复地除以16，直到商为0，将所有余数（从末一个起）顺序排列起来，就是这个数的十六进制表示。

表1—1

十进制表示	二进制表示	十六进制表示	十进制表示	二进制表示	十六进制表示
0	0	0	9	1 0 0 1	9
1	1	1	10	1 0 1 0	A
2	1 0	2	11	1 0 1 1	B
3	1 1	3	12	1 1 0 0	C
4	1 0 0	4	13	1 1 0 1	D
5	1 0 1	5	14	1 1 1 0	E
6	1 1 0	6	15	1 1 1 1	F
7	1 1 1	7	16	1 0 0 0 0	10
8	1 0 0 0	8			

如，十进制的269

$$16\overline{|269}\quad (D \longleftarrow (13)_{10}$$

$$16\overline{|16}\quad (0$$

$$16\overline{|1}\quad (1$$

$$0$$

转换为十六进制数为10D，也即

$$(269)_{10} = (10D)_{16}$$

反之，把一个十六进制数的末位乘以16^0，倒数第二位乘以16^1…再将各项累加，所得的数就是那个十六进制数的十进制表示。

如：

$$(10D)_{16} = 1\times16^2 + 0\times16^1 + 13\times16^0 = (269)_{10}$$

关于八进制数及其与十进制数之间的转换，不再叙述，请读者自己考虑。

§4．计算机语言

人们进行思维活动或人与人之间交流思想，要通过语言，计

算机进行计算或人与计算机交流信息，也需要通过语言。这就是计算机语言。计算机语言是用来指明让计算机依次做些什么事情的，所以又称为程序设计语言，这种语言有其自身的特点和发展过程。

一、机器语言

前文提到，计算机的指挥中心是控制器，且计算机中的数都是用二进制数表示的。实际上，控制器也是用二进制数的0或1（即低电平、高电平）来实现其控制功能的。

原来，任何一种电子计算机，都有一套指令系统，由若干条指令组成，每条指令都可指挥计算机实现某些功能。

一条指令，通常由操作码和操作数两部分构成。操作码由若干位二进制数组成，由于这若干位0和1（即低电位、高电位）的不同组合，因而能使计算机产生不同的动作。操作数部分也是由若干位二进制数构成，它指明被操作的对象。一般地，它可以是被操作数的本身，也可是被操作数在内存中存放的地址，甚至不指明操作数部分（有时是隐含的，这在后面专门介绍）。如果操作数部分指明的是地址，不同类型的机器里，又可能只指明一个地址（被操作数地址）、或指明两个地址（两个操作数的地址，或一个被操作数地址，一个操作结果存放地址），或指明三个地址（被操作的两个数的地址和操作结果存放地址）等等，它们分别被称为一地址、二地址或三地址指令。6502微处理器是一地址指令，但在指明地址时有时只用一个字节（比如A8，实际上指的是00A8），有时又用两个字节（比如3A 8D，它实际上指的是地址8D3A，它总是先指明低八位，后指明高八位）。关于6502指令系统，我们在第十一章里还会进一步介绍。

现在假定在苹果机内存的某一些连续单元中有如下一些数：

10101001 00010000 01100101 00100000

10001101 00001010 00110000

改用十六进制表示，为：

A9　10　65　20　8D　0A　30

这组数实际是6502微处理器的三条指令，其中A9，65，8D分别是三个操作码。

第一条指令：A9　10

把十六进制数的10送入运算器中；

第二条指令：65　20

把运算器中的数(即刚刚送入的十六进制数10)加上内存中用十六进制数表示的第20号单元中的数，结果仍留在运算器中；

第三条指令：8D　0A　30

把运算器中的数(即刚加完的结果)存放到内存中用十六进制数表示的300A号单元中。

类似这种用连续指令写成的算题步骤，称为机器语言程序。

控制器每遇到一个操作码，就会产生不同高低电位的组合，控制机器作相应的动作。地址部分的不同高低电位组合，又会指向相应的被操作对象。从而完成一条指令的功能。一条条指令执行完结，运算就告结束。

由于指令的操作码会直接使机器产生相应动作，操作数部分会直接指向被操作对象，所以用一条条指令写成的机器语言程序，能为机器直接认识。换言之，机器语言是计算机能懂得的唯一语言。

用机器语言写的程序，在机器上运行时，速度是最快的。

值得说明的是，上述例子中，由七个机器字构成的三条指令，必须事先由用户送入机器内存的七个连续单元中；十六进制数的20号单元中也得事先送入一个数；还有，结果存放的地址（十六进制数的300A)也是用户指定的。这就是说，内存中什么地方放程序，什么地方放被操作的数，什么地方放结果，得由用户自行

分配。短小的程序比较好办，长而复杂的程序，分配内存时得格外小心，以防止搞乱了。

显然，用机器语言编制计算程序，对用户来说有着很大的缺陷：

1.难写、难读、难修改；

2.由于每种机器的指令系统不一样，这种程序没有通用性；

3.需要人工分配内存。

二、汇编语言

为着克服机器语言固有的缺陷，人们经过研究探讨，引进了一种汇编语言。

汇编语言实际上是一种符号语言，它把人们难以记忆和辨认的操作码改用有意义的英文单词(或其缩写)作为助记符来代替，而对操作数部分稍加改动，更易于理解。

比如前面那三条机器语言程序，改写成汇编语言如下：

```
LDA     #$10
ADC     $20
STA     $300A
```

这就比机器语言好写、好读，也好修改了。

上述汇编语言中，数字前面的$，指明该数为十六进制数，#表示其后是操作数本身(而不是操作数的地址)。

然而，从机器语言到汇编语言是花了一定代价的。因为计算机只认识机器语言，为了让计算机能认识汇编语言，最简单的办法是在机器内部编制一个对照表，像一本字典一样，对任何汇编符号都能从这个对照表中查到相应的机器语言的指令操作码。进行这种对照查找，当然也得有一个用机器语言写的（以便机器能认识的)程序,即汇编程序。机器在执行这个汇编程序时会把一条条汇编语言翻译成机器语言，从而又为机器所能认识。

可见，执行汇编程序得额外花费机器运行时间，汇编程序和

对照表也要占用内存空间。因而由机器语言改用汇编语言，是以降低机器运行速度和减小用户可用内存空间为代价的。

汇编语言可以克服机器语言的第一个缺陷，但其它缺陷却依然存在。

三、算法语言

算法语言又称“高级语言”，它比较完满地解决了机器语言的所有缺陷，是程序设计语言的一大突破。

常用的算法语言有BASIC、ALGOL、FORTRAN、PASCAL、COBOL等。BASIC语言是其中最简单、最容易学习的一种。

高级语言的最大特点是，规定了一些与英文原意基本上一致的词汇(称为保留字)，从而象自然语言一样来对算法进行描述，而且一些数学运算符号甚至一些代数式子几乎都可以拿来直接使用。

高级语言是一种最为人们容易理解的程序设计语言，而且它不依赖于机器，也就是说有通用性，也不需人工分配内存。因而用高级语言编写程序，给用户提供了很大方便，从而也大大推动了计算机应用的发展。

用高级语言写出的程序，称为源程序，它更是计算机不能直接认识的东西。让计算机认识并能执行高级语言源程序，需要一个翻译和执行的过程，它比汇编程序功能要强得多。

这种翻译和执行过程，通常有两种方式：解释方式和编译方式。解释方式是将源程序逐句地翻译解释，并立即予以执行，然后对后续语句一次次重复这种解释和执行工作，直到源程序结束。BASIC语言大多使用这种方式。完成这种解释执行工作的是一个解释程序，它是用机器语言编制的，所以机器能直接执行它。图1—3是解释方式的示意图。编译方式则是将高级语言源程序全部予以翻译，并转编成相应的机器语言程序(被称作目标程序或目的程序)，然后让机器直接执行这个目标程序，从而实现源程序的所

有任务。ALGOL、FORTRAN、PASCAL、COBOL等高级语言都采用编译方式。完成编译任务的有一套编译程序，通常它也由机器语言编写成，所以机器能直接执行它。图1—4是这种方式的示意图。

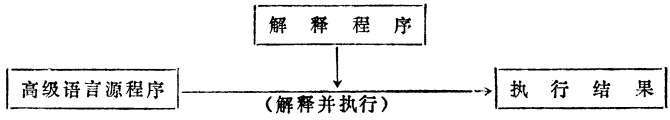


图1—3 解释方式示意图

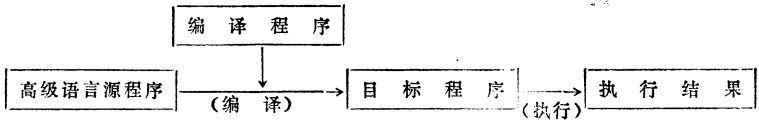


图1—4 编译方式示意图

由此可见，我们通常用高级语言写好一个程序，让机器运行它，这种说法并不十分确切。拿BASIC语言写的程序来说，真正运行的并不是BASIC源程序，而是运行BASIC解释程序。通过解释程序的运行，源程序一句句地被解释执行，从而完成了源程序所要做的一切。然而它还是顺着BASIC程序一句句解释执行的。不同程序语句有不同解释方法和效果，所以我们以后还是说成运行(或执行)BASIC源程序。

不言而喻，高级语言的优点是以降低运行速度和减小用户可用内存空间为代价的。运行速度大约只及机器语言的数百分之一。一个BASIC解释程序大约要占用10k左右的内存空间。随着大规模集成电路技术的飞速发展，指令执行周期越来越短，存贮器越来越便宜，这些代价也就越显得不那么重要了。当然，对某些程序来说，降低运行速度可能是致命性的，所以近年来不少应用程序采用高级语言与机器语言相结合的办法，不是没有道理的。

上面提到的汇编程序、解释程序、编译程序以及操作系统、

服务性程序、库程序等，一般称为系统程序；为解决特定问题而编写的程序，一般称为应用程序。这些都是属于软件的范畴。

习　题

（一）将下列二进制数转换成十进制数：

(1) 101　(2) 1100　(3) 11011001

(4) 101101100110

（二）将下列十进制数转换成二进制数：

(1) 26　(2) 54　(3) 129　(4) 25560

（三）把下列十进制数转换为十六进制数：

(1) 15　(2) 287　(3) 6438　(4) 39684

（四）把下列十六进制数转换成十进制数：

(1) B　(2) 1C　(3) A93　(4) FFFF

（五）把下列二进制数转换为十六进制数：

(1) 10100110　(2) 11000110　(3) 1011001110

(4) 1010010111111010

（六）把下列十六进制数转换为二进制数：

(1) 8F　(2) 3A4　(3) 6D35　(4) FFFF

第二章　BASIC语言的基本概念

本章将向读者介绍什么是BASIC语言，构成BASIC程序有那些最基本的东西。正象盖房子一样，这里仅仅是准备钢筋、水泥、砖、瓦、沙、石等材料。同时在本章中我们还要介绍一些BASIC语言常用的命令和常规的上机操作方法。除NEW、HOME、DEL、LIST、LOAD、SAVE、RUN和END以外，本章中提到的其它命令或语句将在以后的章节中作更为详细的说明。此外，算术函数也放在本章中介绍。

§1. BASIC语言的特点

BASIC是Beginner's All-purpose Symbolic Instruction Code（初学者通用符号指令代码）一词的缩写，是国际上较为普及的一种通用的算法语言。绝大部分小型机和微型机都配置有这种语言。它是一种容易学习而又实用的计算机高级语言。学会了BASIC，再学习其它高级语言就比较地好学了。

BASIC语言中，有17个语句，称为基本BASIC，在任何机器上的任何版本的BASIC中都应该有它们（至少有其中的绝大部分），否则它将欠完备性。这些语句的功能、使用方法及语句格式，在任何机器上也都基本一样。这就使得BASIC程序能保证其通用性。但在不同的机器上，或不同版本的BASIC中，在这17个语句之外还可能增加一些别的语句，这就构成了扩展BASIC。每种机器的扩展都可能有所不同，因而扩展BASIC的通用性受到了一定限制。

本书主要介绍苹果-Ⅱ (APPLE-Ⅱ)微型机上的基本BASIC及其扩展。

苹果-Ⅱ的BASIC又分浮点BASIC (APPLE SOFT BASIC)和整数BASIC(INTEGER BASIC)两种。本书以介绍浮点BASIC为主。关于整数BASIC，将在第十三章中作简要介绍。

BASIC语言有以下一些特点：

1．BASIC语言简单易学。基本BASIC只有17个语句，而且语句和命令所用的词同英语中的单词意义一样，BASIC程序中的运算符号和计算式子同数学上的用法也很类似，因此很容易理解和记忆。另外，BASIC语言程序结构简单，规则也不复杂，初学者易于掌握。

2．BASIC语言是一种会话式语言，人机对话十分方便。源程序输入机器后，计算机会自动检查有无语法错误，一旦发现有错，会输出一些信息通知用户有什么错误，用户可以随时予以改正。一个程序在执行过程中发现错误时，也会给出必要的提示信息，并等待用户核查改正。用户在程序中适当的地方写上一些特定的语句，计算机在执行这些语句时，可以给出一些必要的信息，或等待用户输入一些数据。计算机甚至可以根据用户的不同输入产生不同的动作。

3．BASIC提供一种立即执行的方式，用这种方式，可以在终端上直接进行某些简单的计算或执行一些语句。并且可以用某些命令或语句使程序在执行当中暂停下来。还能用立即执行方式检查某些量的变化情况，或改变某些量的当前值，甚至可以改变程序本身。这对于检查、调试程序是非常方便的。

4．BASIC是一种通用的高级语言，它既可以进行数值运算，也可以进行非数值运算，适用的范围还是比较广阔的。

5．苹果机的BASIC，还可以在屏幕上绘制出各种可变化的彩色图形。并且有游戏控制器，能够进行各种游戏。利用机内的喇

叭，还可以奏出各种曲子来。因而它具有图文并茂、有声有色的特点，对于初学者具有较强的吸引力。用于教学示范，也有着很大的潜力。

§2. BASIC程序的构成

让我们先从一个简单的例子说起。

假定某甲有人民币15元，某乙有人民币20.5元，求甲乙二人共有人民币多少？

把这个问题写成BASIC程序，可以是下面的样子：

```
10  LET  A = 15
20  LET  B = 20.5
30  LET  C = A + B
40  PRINT  C
50  END
```

计算机在执行这个程序时，先让A取得值15，再让B取得值20.5，然后让C取得A与B之和，即35.5（以上都是LET的作用）；再把C的值打印出来(PRINT的作用)；最后停止程序的运行(END的作用)。

BASIC程序的构成规则是：

1. 一个BASIC程序（program）是由若干个程序行（Line）组成的。上例程序就是由五个程序行组成的。

2. 一个程序行又由行号和语句两部分组成：

(1) 行号（Line number），或称标号。如上例中的10，20，30，40，50。它必须是一个正的十进制整数，其作用是用来规定计算机在执行程序时的先后顺序。通常计算机总是先从最小的行号执行起，依序执行下去，直到最大的行号。行号的范围因机器的不同而异，苹果机允许从0到63999。从上面程序例中可以看出，

两个相邻行号并不是相邻的正整数，中间留出了一些间隔，这是为了以后需要时能在相邻程序行之间再插入一些新的程序行。比如上述程序写好以后，又希望求乙比甲多多少钱，并想把这个差打印出来，这时只需增加下面一个程序行即可：

```
45 PRINT B-A
```

尽管行号为45的程序行是最后输入的（即在行号为50的程序行输入之后），计算机会自动按行号大小顺序插入相应的位置。

为了叙述方便，以后我们把以某某为行号的程序行说成是某某行程序。象上面增加的那个程序行，我们说成第45行程序。

（2）语句（statement）。上面程序例中，每个行号右边的部分，都分别地称为一个语句。语句又可以分为语句定义符和语句体两部分：

①语句定义符。如上例中的LET、PRINT、END皆是。语句定义符是用来规定该语句执行何种功能的。如LET表示“赋值”，PRINT 表示打印，END 表示结束。语句定义符是语言本身规定的，解释程序能够识别它们，通常称为保留字，用户不得修改或自造。在本书以后的大量章节中，将分别介绍这些保留字的功能、格式，以及如何使用它们来写出一些有用的BASIC程序。

②语句体。如上例中的A = 15，B = 20.5，C = A + B，B - A，C 等皆是。语句体表明所需执行的具体内容。应该指明的是，有些语句不允许带有语句体。比如50行程序中的END。

由于一个语句可以让计算机完成一种特定的功能，因此，一个语句也可以看作是一条命令。

3．一般来讲，一个程序行只可以写一个语句。但在较多的BASIC 版本中，也可以写多个语句，这时必须在两个语句之间加一个语句分隔符冒号（:）。比如，把上例程序的第10行取消，而把第10行的语句并入20行，这时20行程序有两个语句，应写成下面的样子：

```
20  LET  A = 15 : LET  B = 20.5
```

4．一个语句必须在一个程序行内写完，不允许将一个语句拆开写在两个程序行中。比如上例第30行程序，拆成如下两个程序行是不允许的：

```
30  C =
35  A + B
```

苹果机允许一个程序行多至239个字符，是足够一个语句使用的。万一一个语句特别长时，可以改写成两个语句，这时便可以放在两个程序行里了。比如

```
10  LET  A = 12345 + 6789 + 654 - 768 - 567 + (X - Y) - 23456
```

可以改写成：

```
10  LET  B = 12345 + 6789 + 654 - 768
20  LET  A = B - 567 + (X - Y) - 23456
```

效果是一样的。

5．一个程序，一般应以语句END结束。END没有别的功能，程序执行时遇到它就停止了。在某些机器上，如果一个程序没有语句END，执行时会给出错误信息。在APPLE SOFT BASIC中，没有END，程序一样会正常运行直到结束。

§3．常量与变量

上节提到的那个程序例中，我们除看到了行号和保留字之外，还看到了几个数字，如15、20.5，也还看到了几个字母，如A、B、C。这些数字和字母，分别叫做常量和变量。

一、常量

在程序执行过程中，具体数值本身始终保持不变的量，称为常量。比如数字15，不管程序如何变化，它永远是数字15。

常量又分下面三种数据类型：

1．整常量，也就是整数。

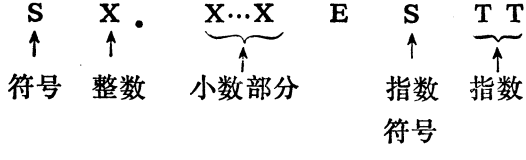
整常量的范围是－32767～＋32767，在这个范围以外的值无法使用。

整常量的数值范围较小，但计算机在处理这类数时速度较快。

2.实常量，即实数，也称浮点数。

实常量的数值范围，在苹果机上为 1×10^{-38}到1×10^{38}。比1×10^{-38}小的数，当作0处理；大于1×10^{38}的数，机器无法表示，给出“溢出”信息。实数的有效位数为九位，当一个实数超过规定的有效位数时，机器会自动转换成科学计数法来表示。

科学计数法的形式如下：



其中S表示正负号，一般最左部的正号予以略去；

X表示0～9的数字；

T也表示0～9的数字，只允许两位，标明指数的数值；

E表示1×10的方次。

如 1000000000

可以用1E＋9表示。

把一个超过规定有效位数的数从键盘上输入计算机时，一般没有必要人工转换成科学计数法，因为这一转换工作计算机会自动地完成。对用户来说，有必要熟悉的倒是：对一个用科学计数法表示的数，如何把它转换成常规表示的数。

把一个科学计数法表示的数转换成常规表示的数，其关键是移动小数点的位置。移动的方向取决于指数部分的正负号：正号时右移，负号时左移；小数点移动的位数取决于指数的值，指数为

多少就移动多少位。移动前要把指数部分（从E开始的末4位）去掉，移动中要补入必要的0值位。如，1.2345678E－06

指数的符号为负，指数值为6，于是将E－06这4 位去掉，然后将小数点向左移动6位，变成：

.0000012345678

这就是1.2345678E－06的常规表示法。

再如，－6.23456E＋06

指数为正6，去掉 E＋06，将小数点向右移6位，变成：

－6234560

3．字符串常量

用双引号“ ”对括起来的字符序列，称为字符串常量。一个字符串常量的字符个数最多不得超过255。下列字符（§6中会进一步介绍它们）因有特殊功能，不得（也不会）出现在字符串中：CTRL－X，CTRL－M，RETURN 键，←键，CTRL－H。由于”号是字符串定界符，也不应出现在字符串中。

例如，“GOOD”，“123”，“1A2B” 等都是合法的字符串常量。

引号对中没有任何字符的串“ ”，也是字符串常量，称为空串。

二、变量

在程序运行中可以改变的量，称为变量。

变量实际上是常量的名字，只不过在程序运行的不同时刻可能代表不同的常量罢了。一个变量名好象旅馆的一个房间标记，今天可以住张三，明天可以住李四。

苹果机浮点BASIC的变量名的构成，必须严格遵守下面三条规则：

1．变量名可以多至238个字符，但机器只承认最左边的两个字符有效。比如变量名 ABCD和ABT，机器将认为它们是同一个

变量名AB。

2．变量名的第一个字符必须是英文大写字母 A～Z 中的一个，其后的字符可以是英文大写字母，也可以是数字。

3．保留字不得用作变量名或变量名的一部分。比如，END、LET 是保留字，那么，END、ABEND，CILET，ALET 等均为非法的变量名。

为避免用户违反第3条规则，某些机器的BASIC中没有第3条规定，而对第2条作了更严格的限制：变量名只允许1到2个字符，且第二个字符必须是数字。但这样一来变量名的个数就大大减少了。按照这种规则，变量名最多只有 286 个，而苹果机的浮点BASIC，变量名可多达955个（若在磁盘操作系统下为954个）。

变量又因所代表的数据的类型不同，有整型变量、实型变量和字符串型变量之分。

整型变量名的尾符是%。换言之，尾符为%的变量名，称为整型变量名，它只能代表合法的整常数（或者说，它只能取得－32767到＋32767之间的整数值）。如AB%，A2%，A%等都是合法的整型变量名。

实型变量名没有尾符，如 AB，A2，A 等都是。实型变量名可以代表任何一个合法的实常量。

字符串变量名的尾符是＄。如AB＄，A2＄，A＄等都是。字符串型变量只能代表合法的字符串型常量。

在苹果机浮点BASIC中，当两个变量名的最左边两个字符相同而尾符不同时，作为不同变量名处理。

任何一个变量参与运算时，总是取它所代表的具体数据（即常量）来进行。整型量（常量或变量）与实型量（常量或变量）虽不属于同一数据类型，但都属于数型量，也称算术型量，相互之间可以自行转换，它们是相容的数据类型，可互相取代。

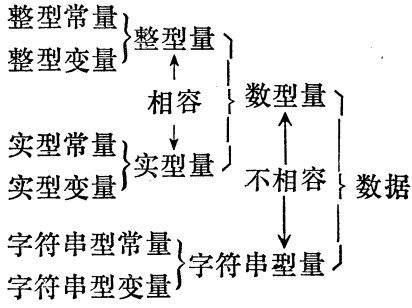
一个整型常数本身就是一个实型常数，实际上不必作什么转

换；而一个实型常数，用常规表示法表示出来，并舍去小数点以后的部分，就是整型常数。当然，将一个实型量转化为整型量时，要求这个实型量的值必须在整型常数允许的范围内才行。

根据上面的说法，实数23.99转化为整型数是23，而不是24。

数型量和字符串型之间不能直接转换，它们是不相容的数据类型。

数据类型之间的关系，可归纳如下：



请读者不要误认为这里提到的整型量属于整数BASIC。这里的整型数只是浮点BASIC下的整型数，使用它的目的在于提高数据处理的速度。

最后说明一下，我们这里所介绍的变量，有一个共同的特点，这就是：任何一个变量，在程序运行的任一时刻，总是只代表一个具体的数据（即常量），而不能代表一组数据。这类变量我们称作简单变量。在第七章介绍数组时，我们将会看到另一类变量——下标变量。

§4. 标准函数

在BASIC程序中，除了允许使用常量、变量之外，还允许使用若干事先定义了的函数，用户只要给出函数名与自变量（或称参

数）的值，就可以求出相应函数的值来。如求5的平方根，可调用函数SQR(5)。

函数名也是BASIC保留字，BASIC解释程序遇到这类保留字时，会自动转入相应的机器语言子程序中，很快算出结果来。

函数又因功能与处理对象不同而分为几种，这里介绍一组算术函数，它处理的对象为数型量，结果也是数型量。或者说它处理的原始数据和结果都是算术值，处理方法也是算术运算。这组算术函数又称标准函数或内部函数。

苹果机的标准函数共11个，现列表2—1说明如下：

表2—1

标准函数	功　　能	备　　注
SIN(x)	给出x的正弦值	自变量x取弧度值
COS(x)	给出x的余弦值	自变量x取弧度值
TAN(x)	给出x的正切值	自变量x取弧度值。当x接近于$\pi/2$（或$\pi/2$的奇数倍）时，这个函数的结果可能超过机器允许值的范围而溢出。
ATN(x)	给出x的反正切值	函数值的单位为弧度，在$-\pi/2$和$\pi/2$之间。
LOG(x)	求x的自然对数	x应为正
EXP(x)	给出e^x的值	$e=2.71828\cdots$
ABS(x)	给出x的绝对值	
SQR(x)	给出x的平方根值	x不得为负
INT(x)	给出不大于x的最大整数	如INT(-8.6)=-9，INT(8.6)=8
SGN(x)	符号函数	$SGN(x)=\begin{cases}1 & (当x>0时)\\0 & (当x=0时)\\-1 & (当x<0时)\end{cases}$
RND(x)	给出大于、等于0而小于1的一个随机函数	*

＊：RND(X)称为随机函数，在有些 BASIC中，X只具形式上的作用，随便取什么结果都一样。在苹果机浮点 BASIC中，X的取值不同，可能会产生不同的随机数，称作哑元。当X大于0时，每使用一次产生一个新的随机函数；当X为0时，使用随机函数会得到刚刚产生过的那个随机数；当X 取负数时，随着负数值的不同，产生不同的随机数；当多次使用随机函数而X 都取一个固定的负数，则每次都产生同一个随机数。好象随机数是由一大批数（大于等于0而小于1)构成的一个表，它们虽不是按大小顺序排列的（是杂乱的)，但却有一定的先后顺序。随机函数的自变量取负数时，随着这个负数值的不同，确定一个指向表中不同位置的指针，并将指针指向的那个数作为随机函数的值，而后指针下滑，指向下一个数。自变量负数不同时，指针的位置也不同；自变量是负数而且相同时，指针指向表中同一数。当自变量大于0时，取出当前指针指向的数作为函数值，并使指针下滑一个位置。当指针等于0时，指针先退后一个位置，取出所指向的数作为随机函数的值，再将指针下滑一个位置。

随机函数主要用于数据的模拟。

另外，算术函数虽只提供了十一种，但不少函数可由它们衍生出来。

比如对数函数 LOG，这里是以e为底的。如果需要求X 的以10为底的对数，则可以利用换底公式：

$$LOG_{10}X = \frac{LOG_eX}{LOG_e10}$$

三角函数可以衍生出来的函数更多。现略举几例：

求X的正割：

SEC(X) = 1/COS(X)

求X的余割：

CSC(X) = 1/SIN(X)

求X的余切：

COT(X) = 1/TAN(X)

求X的反正弦：

ARCSIN(X) = ATN(X/SQR(－X＊X＋1))等等。

§5．运算符和表达式

将数据类型相同的常量、变量和函数用规定的运算符连接起来，就构成了BASIC表达式。表达式本身有一个值。象本章第二节提到的B－A就是一个算术表达式，它的值为5.5。

一、算术运算符和算术表达式

算术运算符有下面五个：

＋	加号	
－	减号	
＊	乘号	
/	除号	
↑	乘方	在键盘上为∧符

单独一个数型量或用算术运算符把两个数型量连接起来，构成算术表达式。算术表达式具有算术常量值，仍是一个数型量。

如5，5＊A，10/5 都是算术表达式。其中 A 是浮点型变量。

SIN(3.1416)/20＋15

也是算术表达式。因为 SIN(3.1416) 是一个算术函数（即180° 的正弦值），具有算术值，属于数型量；将它除20(数型量)仍是数型量；再把这个数型加上15（数型量），结果还是数型量。

在这里我们已看出：算术表达式中算术运算符号的优先顺序，和在数学上是完全一样的：

(1) 如果有括号，必须先做括号内的。在BASIC中，没有大、

中、小括号之分，一律使用小括号（ ），括号内可以套用括号，但不得超过36层。这就同数学上一样，先做最内层括号中的计算，层层向外脱，最后处理最外层括号中的计算。

（2）无论括号内或括号外，函数计算优先于其它算术运算符。

（3）乘方运算优先于加、减、乘、除运算。

（4）乘、除运算优先于加、减运算。乘或除同时出现在一个表达式中时，先完成左边的后完成右边的（即以先后出现顺序为序）。

（5）最后做加、减运算。加、减号同时出现在一个表达式中时，也以先后出现顺序为序。

对于初学者来说，下列几点要格外注意：

1．在数学上，代表两个数的字母相乘时，可以连写。如A代表一个数，B代表一个数，则可用AB代表两个数相乘。在BASIC表达式中，这是绝对不允许的（这时它会把AB当作变量名）。必须写成A＊B。用A·B也是不允许的。此外，＊号不能写成×。

2．↑号代表自乘，A↑B代表A的B次方（即数学上的A^B）。在BASIC表达式中不能写成A^B，不能省略↑符。

3．/号代表除号，必须且仅可占用一个字符位置。2/3不能写成$\frac{2}{3}$，或2÷3（没有÷号键）。

4．左右括号必须成对出现。且不得用方括号〔 〕或花括号{ }代替。

二、字符串运算符与字符串表达式

字符串运算符只有一个：

＋　　连接运算符

其功能是将两个字符串连接起来。

如　“ABCD”＋“123”

将得到一个新的字符串，其值为“ABCD123”。

可见，一个字符串型量，或用字符串运算符连接两个字符串型量，构成字符串表达式。字符串表达式具有字符串常量值。

三、其它运算符与表达式

除上面介绍的两类运算符与表达式之外，还有关系运算符 =、<、<=、>=、>、< >和逻辑运算符AND、OR、NOT 以及关系表达式、逻辑表达式。这些将在第四章中介绍。

四、运算符的优先顺序

在一个表达式中如果出现多种运算符号，它们的优先顺序规定如下（排在同一行上的若干运算符，视为同一优先级的，以在表达式中出现的顺序为序）：

( ) —括号

函数

+、-、NOT —单边运算符，+：正号，-：负号。

↑

*、/

+、-

<、<=、=、>、>=、< >

AND

OR

例如：

```
2+3*(4-(5+6*7))
  ↑ ↑  ↑  ↑ ↑
  ⑤ ④  ③  ② ①
```

再如：

```
NOT3AND(3+3-1)↑2-3*8-1OR3
 ↑   ↑    ↑ ↑  ↑ ↑ ↑ ↑  ↑
 ③   ⑧    ① ②  ④ ⑥ ⑤ ⑦  ⑨
```

圆圈内的数字，表明优先顺序。

§6．上机操作提要

这一节介绍一般的上机操作过程和常用命令。

学习计算机的任何一种语言，都要力求上机实践。“无机教学”尽管也可以学到一些东西，但效果要差得多，特别是对于初学者来说。

上机操作，因机型不同而可能有某些区别。这里，仅介绍苹果机的使用方法。其中大部分命令在别的机器上用法差不多。

一、系统建立

在计算机上建立BASIC系统，是使用BASIC的必要准备。它会把解释程序建立好，以便解释执行你的BASIC程序。

苹果机浮点BASIC系统的建立非常简单。用下面三种方法之一均可：

1．当你所使用的苹果机系统没有配置磁盘机时，打开机器电源后立即进入苹果浮点BASIC系统。

2．当你所使用的苹果机系统配置有磁盘机，而你又不打算使用磁盘机时，打开机器电源后立即按一下键盘上的RESET键（有的机器要按CTRL－RESET键），机器随即进入浮点BASIC系统。

3．当你准备使用磁盘机时，打开电源后，1号磁盘机将被启动，磁盘机上的小红灯被点亮。这时你必须得将载有磁盘操作系统(DOS)的磁盘轻轻插入磁盘机的扁平口内，并将小门盖关上。等到磁盘机停止转动，小红灯也就熄灭，这时浮点BASIC系统就建立好了。

用第3种方法建立BASIC系统时，磁盘操作系统也被装入内存。并且在建立过程中会在屏幕上出现一些提示，这是磁盘上第一个BASIC程序（通常以HELLO命名）运行时产生的输出。

浮点BASIC系统建立好的唯一标志是在屏幕上适当行的左部出现一个向左开口的方括号]，被称为浮点 BASIC 提示符。在这个提示符之后还会出现一个方块形的闪烁光标，它表明等待用户从键盘上输入程序或命令。

二、键盘简介

既然程序和命令可由键盘输入，那么键盘上有那些键、又如何排列的呢？

计算机的键盘大同小异。苹果机的标准键盘如图2—1所示：

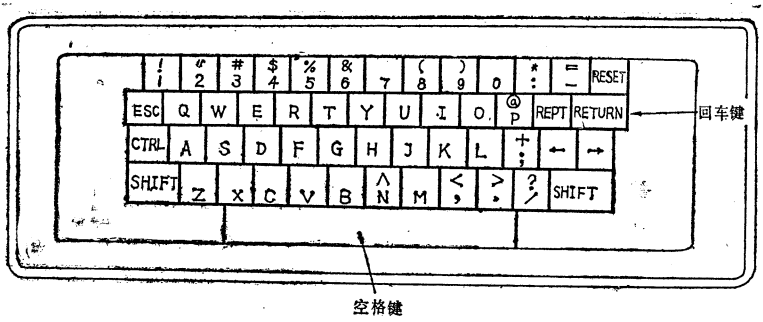


图2—1　苹果机键盘示意图

键盘上大部分键的位置与英文打印机的排列位置差不多。会用英文打字机的人，能很快地学会计算机键盘的使用方法。

键盘上有些键上只标有一个字，无疑敲击该键时可以输入所标的字母（或数字）。有一部分键上则标有两个符号，如最上面一排，左起第一键上标有！和1，左起第二键上标有〝和 2，……这种键共17个，它们被称为双档键或双功能键。直接敲击这种键时，输入的是该键上靠近击键者的那个符号，如数字的1～9等，这是一个档级（功能）。当击键者先按下SHIFT键（在最靠近击键者的标有字符的那排键的最左端和最右端，随便按下其中一个。并且勿将手指提起！）然后再按下某一双功能键（注意，此时SHIFT和相应的双功能键都在按着），当你将所有按下的手指提起后，刚刚按过

的那个双功能键的标在上部（离击键者较远）的那个符号被输入计算机中。在键盘上还有一个最长的键，叫空格键，安置在最靠近击键者的部位上，没有标任何字符。每敲击它一次，输入计算机一个空格。

在输入上述各类字符时，每次输入的字符将被送往屏幕在闪烁光标处显示出来，同时将闪烁光标向右推一个字符位置。当闪烁光标移至再不能向右移动时（即该行显示满 40 个字符时），被自动移至下一行的最左端。可见，闪烁光标表明等待输入字符的显示位置。

RESET键（左上角）的功能，我们已略为提到过。它是一个复位键。任何时候按下它，都可以恢复到BASIC建立时的状态，即在适当行（一般在击RESET键前光标所在行的下一行）的左端出现一个BASIC提示符，随后提示一个闪烁光标。RESET键并不破坏当前内存的状态，包括BASIC解释程序、DOS以及用户输入的程序和敲此键前运行的结果。

←键、→键，即左移键和右移键，它可使闪烁光标左移或右移一个位置。连续按下多次可作连续移动。当光标向左移时，被移过的字符虽不会从屏幕上消失，但它不再被当作已输入字符对待。当光标向右移动时，屏幕上被移动过的位置上的字符（如果没有字符则当作空格符）也被当作输入字符处理。

REPT 键称为重复键，在你按下这个键几秒钟之后，如果你还按着这个键，那么，在按此键前最后敲入的字符会重复地被输入，直到你停止对此键的按压。

←键、→键和 REPT 键的使用，能为你提供很好的编辑手段。而ESC键可以提供更强的屏幕编辑功能。有兴趣的读者，不妨读一下第十一章监控命令简介一节中的编辑字符一段。

用上述方法输入、编辑好的字符序列，仅仅被存放在键盘输入缓冲区中，还没有被当作命令或程序提交处理。因为机器还不知

道你输入的这个字符序列是否结束。因此，当你结束输入时，应通知计算机。这就是RETURN键的主要作用。只有击了这一个键，你所敲入的字符序列才为计算机实际接收。RETURN 敲入之后，计算机会自动检查在 RETURN 键敲入以前你敲入机器中什么内容。如果是带行号的一个程序行，那么计算机会把这个程序行放入程序区，然后在下一行上出现BASIC提示符及闪烁光标，等待用户新的输入；如果计算机发现你刚刚敲入的是一立即执行个命令，计算机会自动执行它，执行结束后，在下一行上出现BASIC提示符和光标，等待用户输入。可见 RETURN 键有回车换行的功能，故又称作回车键。

CTRL 键是一个特殊键。它一般和字母键配合使用，产生特殊的字符编码。有相当一些字母键和它配合使用时会产生特定的控制功能，故又称作控制字符。如CTRL-X可以使刚刚敲入的字符序列（在没按RETURN键时）失效，CTRL-M等同于 RETURN 键，CTRL-H 等同于←键，CTRL-U 等同于→键，CTRL-C 可以暂停程序的运行，CTRL-S 可以暂停输出等等。敲入控制字符的方法类似于敲入双档键的上档字符：先按下 CTRL 键（手指不提起）再按相应的字母键。值得提醒的是，控制字符是一类不可显示字符。因此对一些没有定义过的控制字符，可用来作程序保密用。

提高键盘输入效率，是节省机时的重要手段。一个熟练的操作人员一分钟可以输入完的程序，让一个对键盘很生疏的人来输入，可能要用到一、两个小时。熟悉键盘、熟悉手法，对于青少年计算机使用者来说是必须强调的，要使他（她）们能象钢琴家弹钢琴那样熟练地使用键盘，要十个指头有分工，熟悉到不看键盘也能准确无误地输入程序。

三、输入一个程序的方法

苹果机的BASIC系统建立好后，最好先使用以下两个命令：

1.NEW。NEW命令的功能是清除内存中当前的BASIC 程序和变量，以便接收一个新的程序。

如果内存中已有一个 BASIC 程序（这个程序可能是建立BASIC系统时从磁盘上调入的，如前面提到过的HELLO；也可能是别人敲入的），当你输入一个新程序时，事先没有使用NEW命令，那么内存中的旧程序就会和你新输入的程序参杂在一起，那就乱套了。

2. HOME。HOME 命令的功能是将显示屏清除干净，并将光标移至屏幕的左上角。

HOME命令也可用命令CALL－936 或命令 ESC@（先按ESC键，随后按下CTRL－键@）代替。

注意：各种命令敲入后，要击RETURN键方为机器接收。

在用过上面两个命令之后，就可以输入程序了。

输入程序时，以程序行为单位，顺序输入其中所有字符（包括行号、语句），行号与语句之间、语句定义符与语句体之间，最好输入一个空格，如果你不输入这些空格，在重新列示出来时，计算机时也会自动加上。当发现输入有错时，可用←键将光标移至出错处重新输入正确的字符。当一行程序输入完，要敲RETURN键告诉计算机。然后再输入下一行程序。

用同一个行号输入两个程序行时，前一次输入的程序行被后一次输入的程序行取代。因此，想要修改一个已被机器接收的程序行，可以用重新输入这行程序的方法。

当需要删除一个已输入的程序行时，只要敲入该程序行的行号并立即敲回车键即可。这相当于敲入一行空语句。

如果想删除若干相连的程序行，可用DEL命令。该命令的格式是：

DEL 行号1，行号2

它可以删除从行号1到行号2的所有程序行。

四、程序的显示与打印

当机器的内存中已有一个BASIC程序时，用LIST命令，可以将程序清单一行行显示在屏幕上。

LIST命令的格式是：

LIST　行号　——将指定行号的程序行显示出来。

LIST 行号1，行号2 ——显示从行号1到行号2的所有程序行。

LIST 行号，——显示从指定程序行到程序尾的所有程序行。

LIST，行号 ——显示从程序头到指定程序行的所有程序行。

LIST ——显示全部程序行。

如果想把程序清单打印出来，这首先要打开打印机的电源，其次要用 PR#s 命令（命令中的 *s* 表示打印机接口板所在的槽口号。如果打印机接口板插在1号槽口上，就用PR#1命令）来启动打印机。然后用上述LIST命令，不仅程序行能显示于屏幕上，而且能通过打印机打印到纸上。关闭打印机时，要用 PR#0命令。

由于屏幕尺寸是有限的，它从上到下只能显示24行字符。因此，当需要列出的程序多于24行时，先行列出的程序行被依次从屏幕顶端移出，这称作滚动输出。在这种情况下，被移出去的部分就看不到了。为此，定义了一个CTRL－S命令，任何列示过程中按下此键可暂停列出。等到看清屏幕上的内容，需继续列示以后的部分时，可按任何一个键（甚至 CTRL－S 键，但不得按下CTRL－C，RESET等键）。

将程序清单列示于屏幕上（或打印出来）并不是必要的步骤。它的用途在于阅读或检查某些程序段，以便读懂它，或修改它。

三、程序的执行

程序的执行方式有两种：延迟（程序）执行方式和立即（命

令）执行方式。

象本章开始引入的那种程序，每个程序行之首为行号。这类程序输入机器后，程序并没有被执行，只有敲入RUN 命令之后，程序才能被执行。这种执行方式称为延迟执行方式或程序执行方式。用这种方式执行一个程序后，源程序仍留存在内存中。

如果在RUN之后跟上一个行号，则机器会从给定的行号开始运行，直到结束。

使用 RUN 命令执行一个程序时，内存中原有的变量将被清除，然后根据程序的需要使程序中的所有变量取得相应的数据。

如果从键盘上输入不带行号的语句（如果是几句语句，要用语句分隔符冒号分隔开）或命令，当敲入 RETURN 键（前面已提到过，这是输入一个程序行或命令后所必要的）后，则语句或命令会立即执行。这种执行方式称为立即执行方式或命令执行方式。用这种方式执行过的语句或命令，不会再在内存中找到（事实上它们并没有存入程序区）。若是立即执行的语句会使得某些变量取得数据，则这些变量和它们所代表的数据，将保留在内存中。

我们前面用过NEW、HOME、CALL－936、RUN、LIST等命令，实际上都是在立即执行方式下实现的。

敲入不带行号的语句：

```
PRINT  15+20.5
```

一旦敲入回车键，立即会输出35.5。

敲入不带行号的语句：

```
C=15+20.5
```

当敲入回车键后，虽不会输出什么结果，但变量C却得到了实数35.5。

这时再敲入命令：

```
PRINT  C
```

会在屏幕上显示35.5。

这些都是立即执行方式的例子。

四、程序的存贮、调入与删除

一个程序输入机器后，被存放在随机存贮器中，当机器断电后，这些程序也就消失了。如果程序有保留价值，可以把它暂存在外存贮器上。下次开机后只要从外存贮器上调入内存中，便可继续使用或修改。

SAVE命令，可以把内存中当前的BASIC 程序存入磁带（这当然要打开录音机的电源，并设置成录音状态）。

LOAD 命令则可以把磁带上的当前位置开始存放的程序装入到内存中（这当然也必须打开录音机的电源，并设置成放音状态）。

用磁带读写程序，速度较慢，且音量大小要调整好。还有，磁带必须事先放在适当的位置才行。这就使得操作稍微麻烦点。如果用磁盘机来读写程序，则要方便得多。

下面几个命令是在磁盘操作系统 DOS 装入内存后才可使用的，用起来极为方便。

SAVE　程序名——可以把当前内存中的 BASIC 程序以给定的名字（即命令中的程序名）存入磁盘。程序名必须以字母打头，其后的字符不限（但不得为CTRL－M，CTRL－X等控制字符；且字符个数不要多于30个）。如果磁盘上已有同名的BASIC程序，则旧的程序为新存入的程序取代。

LOAD　程序名——可以将指定的程序从磁盘上装入内存中。执行任何一个LOAD命令（包括对录音机的）都会把内存中原有的程序和变量清除掉。

用LOAD命令装入内存的程序，可以被修改和运行。

RUN　程序名——可以将指定的程序从磁盘上装入内存并运行它。执行这个命令，实际上相当执行下列两个命令：

LOAD程序名

RUN

CATALOG——可以列出磁盘上所有程序（及其它文件）的目录。在使用SAVE程序名命令前，最好先用一下 CATALOG 命令，以防止冲掉有用的程序。

DELETE 程序名——可以从磁盘上把指定的程序删除掉。

最后这四个命令都不是BASIC命令，而是磁盘操作命令。由于经常要用，故仅在此简要介绍(第十章要专门讲)。

习　题

（一）指明下列各数据，哪些是合法的常量。

(1) 1E+39

(2) 359,672001

(3) "ABC　34"

(4) 29·"AB"

(5) 32.001

（二）指明下列各项，哪些是合法的变量名。

(1) AA

(2) A.5%

(3) 2A$

(4) AB$%

(5) BBC%

（三）将下列各数转换成常规计数法表示的数。

(1) 8.2345E+05

(2) 2.3456E-06

(3) 1.259E+09

（四）将下列各代数式写成BASIC表达式。

(有条件的读者，用立即执行方式输出各算式的结果)

(1) $\frac{88-52\times63}{18+47\times3}$

(2) $\dfrac{(2\sin 45°)^4}{e^2 \cdot \ln 5}$

(3) $6 \cdot \dfrac{2 \times 3^2}{5^3}$

(4) $\left| \sqrt{\sin^2 65° + \cos^2 25°} \right|$

(五) 写出下列各表达式中运算符号的执行顺序

(1) 3 * 5 + 2 * (68/(2 + 3))

(2) A + B * 6/4 + 8OR 65 + 2 * 8 ↑ 2

(3) A + B * SIN(25 + 4) - 85 AND NOT 3

第三章　数据的输入与输出

任何一个有意义的BASIC 程序，总要有一些初始数据，否则这个程序就没有处理对象。同样，程序运行的结果（包括某些中间结果）也有必要显示或打印出来，否则人们将不知道这个程序干了些什么事情。

本章将介绍语句LET、INPUT、READ、DATA、RESTORE、PRINT以及命令 PR#、IN#。在介绍这些语句之前，还对变量表和一些描述符号作一些说明，它们在以后的章节中会经常用到。

§1．必要的说明

一、变量表简介

算法语言中引入变量这一概念是非常重要的，它可以使一些数学上的代数式子直接被使用。然而，变量名是不能直接参与数值运算的。在算法语言中，如果设定A 是一个变量名，要想求得A＊A的结果，必须把A所代表的值拿来进行自乘才可以求得。即使在数学中可以用a^2表示$a\times a$，但它实际上也是一个代号。要想得到a^2的数值，也必须把a的值拿来自乘。

为了记录每个变量所代表的数据，就需要在内存中建立一个变量表，这个表在程序运行过程中建立和使用。换言之，机器在执行一个BASIC程序时，会依据变量出现的先后顺序，把它们的名字和所代表的数据登记下来。当然一个变量只登记一个名字，当它所代表的数据改变以后，用改变后的数据代替原来的数据。

例如第二章我们提到的那个程序：

```
10  LET  A = 15
20  LET  B = 20.5
30  LET  C = A + B
40  PRINT  C
50  END
```

在执行程序的前两行时，先后让变量A取得数15，让变量B取得数20.5，这样变量表中先后登记了两个变量名和它们所代表的数：

A	15
B	20.5

执行第三行程序时，先登记变量名C，根据该语句的功能，C应取得A＋B之和。而变量A和变量B的值从变量表可查得分别为15和20.5，于是它们之和为35.5。把这个35.5填入变量名C下，这个语句就执行完了。这时变量表如下：

A	15
B	20.5
C	35.5

第四行程序是打印C的值，这从变量表中可直接查得，输出即可。

如果我们将上述程序的30行、40行修改成：

```
30  A = A + B
40  PRINT  A
```

其它程序行照旧，输出的结果是一样的。但是在执行第30行时，要让变量A取得一个新值（旧值为15），由于变量名A已登记，则无需登记新的变量名。它同前面介绍的方法一样，先从变量表

中取出变量A和变量B的值相加，再将其和填入变量名A之下。这时变量A的旧值15就被冲掉了。变量表变成下面的样子：

A	35.5
B	20.5

可见，程序在执行过程中逐步地建立了一张变量表。程序运行时会不断地查询这张表，实现按名置数或按名取数的功效。

字符串型变量名也被构造在这张变量表中。但由于每个字符串的长度（即字符的个数）并不固定（整常数和实常数在机器内部表示时长度是固定的），把字符串常量放置在这张表中会破坏这个表每个元素的定长结构，使得以后查找时发生困难。因此另辟了一个字符串数据区，专用于存放字符串型数据。而在变量表中相应的字符串变量名下给出一些索引信息（如该字符串的长度，在字符串数据区中的首地址）就行了，这同样可以按着名字找到数据。

二、三个记号

在以下一些章节中，为了描述方便，我们引入符号对〔 〕、符号对{ }和符号|。它们的意义分别如下：

1.〔 〕——表示其中的内容为可选项。也就是说，其中的内容可以用，也可以不用。

比如：

变量〔，变量〕

等价于：

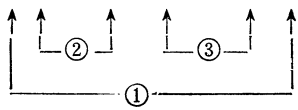
变量

或：

变量，变量

我们前面介绍的LIST命令，如果加用〔 〕表示，可写成以下(1)、(2)的格式：

LIST〔〔行号1〕〔，行号2〕〕 ……(1)



LIST 行号， ……(2)

对格式(1)，我们把各〔 〕对标了号码。可以看出：

当不取①内的内容时，相当于LIST命令；

当不取②内的内容时，相当于LIST，行号命令；

当不取③内的内容时，相当于LIST 行号命令；

当全取时，相当于LIST 行号1，行号 2 命令。

可见格式(1) 用了〔 〕对，短短的一行命令格式相当于四条命令格式。这就简化了描述。

格式(2) 是为了补齐LIST的全部格式。

2.{ } ——表示其中的内容可以重复。也就是说，其中的内容可以用一次，二次，三次…。

比如：

变量 {，变量}

等价于：

变量1，变量2

或：

变量1，变量2，变量3

…

3.| ——表示取其左边的一项或取其右边的一项。

比如：

，|；

表明取逗号(，)，或取分号(；)。

如此，数字的定义如下：

0|1|2|3|…|8|9

英文大写字母的定义是：

A|B|C|…|X|Y|Z

字符串常量显然可定义成：

“〔{字符}〕”

必须说明，这种符号描述法仅仅为了书中叙述的方便，在写BASIC程序时千万不要使用这三个符号。

另外，上面所说的记号中的内容，比如变量，行号等等，这里仅具形式上的意义。使用时要给出真正的实体（如变量名A，或行号20等），且多变量的重复，一般是指若干个不同变量。切记不要弄错。

§2. 打印语句（PRINT）

打印语句是BASIC语言中输出数据的主要手段。

最简单的打印语句格式是：

〔行号〕 PRINT 表达式

其功能是把保留字后面的表达式的值打印出来（注意，想把结果打印在纸上时，必须打开打印机的电源并接通打印机。否则所谓打印应作为输出在屏幕上来理解。因此，有时我们直接说成显示或输出。这些在以后的叙述中不再重复说明）。

比如：

```
PRINT  5
```

将在屏幕上显示5。

```
PRINT  2+3
```

也在屏幕上显示5。

上面两个例子，我们使用了立即执行的打印语句。

如果把第二章开始引入的那个程序输入机器，按RUN命令后，会在屏幕上显示35.5。那里的PRINT是延迟执行的。

对打印语句，需要注意以下几个细节问题：

(1) 打印语句输出的是跟在PRINT之后的表达式的值，而不是表达式那个式子。

所以语句：

```
PRINT 2+3
```

显示的是数字5，而不是式子2+3。

字符串表达式的值是字符串常量。我们知道字符串常量是用引号对“ ”括起来的，但输出时，这个引号对将被略去。

比如：

```
PRINT "GOOD BYE"
```

输出 GOOD BYE，而不是“GOOD BYE”。

同样，语句

```
PRINT "123.4+2"
```

输出为 123.4+2，而不是 125.4。

因此，PRINT之后跟的如是字符串型常量，则会将引号对内的字符照样输出。

这样，若是要输出一个计算式子，用引号对把这个式子括起来放在PRINT保留字之后，就可以实现。

PRINT语句这种输出字符串常量的功能，常常被用来在程序运行的某些时刻输出一段文字，以便给用户一些必要的提示。譬如，把一篇文稿的内容作为字符串常量，用PRINT将其输出，可以打印或显示出这篇文稿来。

如果PRINT之后跟的是变量名（或带有变量名的算式），机器会从变量表中查得这个变量的值（或参与算式运算后得到算式的值）打印出来。

(2) PRINT语句之尾部如果没有逗号或分号时，在执行完数据的输出后，会使下次输出位置改变到下一行的最左端（通常沿用打字机叫法，称为回车换行）。

如程序：

```
10  PRINT  15
20  PRINT  30
30  END
```

敲入命令RUN之后，显示

```
15
30
```

如果当一个PRINT语句没有语句体时（即PRINT之后什么也没有），仅产生回车换行动作。

如程序：

```
10  PRINT  15
20  PRINT
30  PRINT  30
40  END
RUN
15

30
```

这种不带输出项的PRINT语句，常被用来使屏幕上或打印纸上留出若干空白行。

（3）PRINT语句的输出项之后，如果带一个分号时，会使下个PRINT语句输出项紧跟在前次输出位置之后输出（称为紧密输出）。

如程序：

```
10  PRINT  3+5;
20  PRINT "A"
30  END
RUN
8A
```

(4) 如果PRINT语句的输出项之后紧跟一个逗号，会使下一个输出项从下一个输出区的区头开始输出（称为分区输出）。

在苹果机的显示屏上，从左到右每行可显示40个字符，分为一、二、三三个输出区。最左边16个字符位置（即1～16)为一区，后续的16个字符位置（即17～32）为二区，最右边 8 个字符位置(即33～40）为三区。当一个数据在一个输出区内放不下时，会自动顺沿到后续输出区中。第三个输出区的后续输出区为下一行的第一输出区。此外，由于第三区只有8个字符位置，当一个数据等于或多于8个字符而又该在第三区输出时，机器会跳到下一行的第一区输出，以保持一个数据的完整性。

因此，程序：

```
10  PRINT  3,
20  PRINT  3 * 15,
30  PRINT  42 + 5,
40  PRINT  7 ↑ 2/2
50  END
RUN
3                    45                   47
24.5
```

而程序：

```
10  PRINT  3,
20  PRINT "GOOD  BYE" ,
30  PRINT  304.28157,
40  PRINT  65
50  END
RUN
3                    GOOD  BYE
304.28157            65
```

特别地，当一个PRINT保留字之后仅有一个逗号时，也会使

下一个PRINT输出项推后一个区。

如：

```
10  PRINT  3,
20  PRINT  ,
30  PRINT  6
40  END
RUN
3                                   6
```

(5) 一个PRINT语句可以输出若干个数据项，但两个输出项之间必须用逗号或分号分隔开。分号和逗号的用法分别按照上面介绍的规则。

如：

```
10  PRINT  32/8, 45; "G", "CD12", 54
20  END
RUN
4                45G            CD12
54
```

为了用紧密输出而又希望两个数据能分离开时，可在两个输出项之间加入一个空格项。

如：

```
10  PRINT  3+5; " "; 2*6; " "; 300
20  END
RUN
8  12  300
```

当然，上述插入的空格输出项也可改用逗号输出项。如：

```
10  PRINT  3+5; ", "; 2*6; ", "; 300
20  END
RUN
8, 12, 300
```

显然，想在屏幕上输出一个算式，再跟着输出这个算式的运算结果，是很容易的：

程序：

```
10  PRINT "(3*7-9)/4=" ; (3*7-9)/4
20  END
```

和程序

```
10  A=(3*7-9)/4
20  PRINT "(3*7-9)/4=" ; A
```

运行结果输出相同：

```
(3*7-9)/4=3
```

(6) 在APPLE SOFT BASIC中，从键盘上敲入 PRINT 时，可以用问号（？）代替。机器会自动地把字符“？”理解成保留字PRINT存入程序区（对延迟执行方式）或立即执行（对立即执行方式）。这一点很有意思：“PRINT A”的意思是打印A的值，而“？A”象是问A为多少。意义很吻合。

最后，综合上面介绍的打印语句的各种情况，不难写出如下打印语句的完整的格式：

〔行号〕 PRINT 〔〔表达式〕〔{, |; 〔表达式〕}〕〕

对这个格式的理解，我们不全部列举。稍许指出几个供参考。

当最左端的〔行号〕被选用时，此打印语句为延迟执行方式；否则为立即执行方式。

保留字PRINT之后的描述中，最外层方括号中的内容不选用时，相当于保留字之后什么也没有，产生回车换行动作。

保留字PRINT之后的内容全部使用时，相当于PRINT之后有连续若干个输出项，每两个输出项之间可以是逗号或者是分号予以分隔，并且不得省去这种分隔。

其它情况，读者可自行体察。

上述介绍的打印语句，一般说来还是很好使用的。但有时对

一些特殊的输出，用起来稍嫌麻烦。第八章的第一节，我们将介绍几个语句和函数，把它们和PRINT语句结合起来使用，会使得输出控制更加方便。

§3. 赋值语句(LET)

赋值语句是使变量取得数据的常用方法之一。

赋值语句的格式是：

〔行号〕〔LET〕 变量名＝表达式

赋值语句的功能是：把“＝”号右端的表达式的值赋予“＝”号左端的变量。或者说，让“＝”号左端的变量取“＝”号右端表达式的值。

比如：

```
LET A=10
```

就是让变量A取得10这个数。

正象我们在这一章开始时所叙述的那样，机器在执行上面这个赋值语句时，会在变量表中变量A（如果变量表中尚无此变量名，则把它登记下来）的名下填入10这个数。

对于赋值语句，需要明确下列各点：

(1) 保留字LET本身是可选项。也就是说，保留字LET可以省去不写。

这样，上面那行程序可直接写成：

```
A=10
```

保留字LET可以省略，在多数计算机上是允许的。

(2)“＝”号，称为赋值号，不同于数学上的等于号。

在数学上，

```
A=A+1
```

是矛盾方程，不存在这样的A。

而在BASIC中，这类语句是屡见不鲜的。它执行的结果是在旧A取值的基础让 A 取得一个新值。也就是改变变量A所代表的数值。上面那个语句(A = A + 1)执行的结果就是让变量A增大1。

(3) 赋值号左部的变量名必须和赋值号右部的表达式属于同一的或相容的数据类型。

比如：

```
A = BC + D
C = A% + A
A$ = "GOOD" + C$
```

等，都是合法的赋值语句。

而语句：

```
A$ = A
A = "GOOD"
A = "123"
A$ = B%
A$ = 123
```

等，都不是合法的赋值语句。

关于整型量和实型量之间的赋值，机器以遵从赋值号左端变量名的数据类型为原则。也就是说，当赋值号左端为整型变量名时，赋值号右端的表达式若为实型数，则将其小数部分舍去（不是四舍五入），赋给赋值号左端的变量；当赋值号左端的变量名为实型量时，赋值右端表达式的值视同实数。

因此，程序：

```
10  A = 10 + 20.5
20  B% = A + 4.9
30  PRINT  A; ", "; B%
40  END
RUN
```

```
30.9, 35
```

(4) 一个变量未被赋值时，将取零值(对数型量) 或空串(对字符串型量)。

如程序：

```
10  PRINT  A : PRINT  B$
20  A = A + 5
30  B$ = B$ + "A"
40  PRINT  A, B$
50  END
RUN
0

5                A
```

(5) 不能够连续赋值，但可以辗转赋值。

如下列程序行原想使变量A和变量B均取值30：

```
10  A = B = 30
```

由于不能连续赋值，这行程序运行后，结果全然异样：变量A将取值1或0，而变量B的值根本没有变化(为什么会这样？学完下章自明)。

而程序：

```
10  A = 32
20  B = A
 ⋮
```

是合理的。运行后变量A和变量B的取值均为32。

初学算法语言的人，有时不大注意规定的语句格式而习惯于数学上的写法。这点要格外注意。

比如下面的程序：

```
10  A = 5
20  B = 10
```

```
30  A+B=5+10
40  END
```

或者把30行写成

```
30  A+B=A+B
```

这样写出的两个30行，都是错误的。因为赋值语句的格式中，要求赋值号的左端为一变量名。而这里的A+B并不是变量名(它不符合变量名的构成规则)。

下面就赋值语句的使用，给出两个程序例。

例1. 某学生5门功课的考试成绩分别为75，82，90，63，84。求该生的考试总分和平均分。程序如下：

```
10  A1=75
20  A2=82
30  A3=90
40  A4=63
50  A5=84
60  A=A1+A2+A3+A4+A5
70  C=A/5
80  PRINT "TOTAL=" ; A, "MEAN=" ; C
90  END
RUN
TOTAL=394          MEAN=78.8
```

当然这个程序还可以写得简练些：

```
10  A=75+82+90+63+84
20  C=A/5
30  PRINT  A, C
RUN
394                    78.8
```

这说明，同一问题的程序，并不是唯一的。

例2. 求解一元二次方程式$AX^2+BX+C=0$的两个实根。

当A≠0，$D=B^2-4AC>0$时，有两个实根：

$$X_{1,2}=\frac{-B\pm\sqrt{B^2-4AC}}{2A}$$

设：A=2，B=18，C=23

程序如下：

```
10  LET  A=2
20  LET  B=18
30  LET  C=23
40  LET  D=B*B-4*A*C
50  LET  X1=(-B+SQR(D))/(2*A)
60  LET  X2=(-B-SQR(D))/(2*A)
70  PRINT "X1=" ; X1,  "X2=" ; X2
RUN
X1= -1.54196011        X2= -7.458043989
```

§4. 键盘输入语句(INPUT)

键盘输入语句是使变量取得数据的另一种方法。

键盘输入语句的格式是：

行号　INPNT〔“提示字符串”；〕变量名〔{，变量名}〕

（请读者注意：这个格式说明中的行号是不可省缺的，因而表明它仅可用于延迟执行方式中。可以想见：如果一个命令仅可用于立即执行方式时，格式说明中将不带行号部分）

这个语句的功能是使保留字INPUT之后的各个变量名依次从键盘上取得数据。

计算机在执行一个BASIC程序时，凡遇到INPUT语句，通常显示一个问号“？”，等待用户从键盘上输入数据。假定INPUT保留字之后只有一个变量名，那么用户从键盘上敲入相应的数据后，应敲击回车键表示输入结束，这时机器将把刚刚经由键盘接

收的数据，填入变量表中对应的变量名下（即INPUT后跟的那个变量名。如果变量表中尚无此变量名时，在变量表的尾部登记上这个变量名）。然后执行INPUT语句的后续语句。

当INPUT保留字之后有多个变量名时，可以一次输入一个数据（经过多次输入，把所有数据输完），也可以一次输入几个数据，还可以一次把所有数据输入（注意，作为一次输入，总是以敲回车键为标志的）。当机器发现输入数据的个数少于INPUT保留字之后变量名的个数时，总会出现两个连续的问号"？？"提示，继续等待用户的键盘输入。在一次输入多个数据时，必须在输入的两个数据之间敲入一个逗号予以分隔。

例如，在程序运行中，希望从键盘上为变量A、B、C分别输入数据，并假定要输入的三个数依次为12，34，567.8。

相应的程序段如下：

```
⋮
100  INPUT  A, B, C
110  PRINT  A, B, C
⋮
```

程序在执行到100行时，提示一个问号，然后等待用户的键盘输入。

用户可以用以下任一种方法敲入数据（为了不引起误会，在本节下面的表述中，我们把用户从键盘上敲入的内容之下划了横线，并把回车符用↙来表示）：

①第一种方法：

```
RUN↙
? 12↙
? ? 34↙
? ? 567.8↙
12                    34                    567.8
```

②第二种方法：

```
RUN↙
? 12, 34↙
? ? 567.8↙
12              34              567.8
```

③第三种方法：

```
RUN↙
? 12, 34, 567.8↙
12              34              567.8
```

使用INPUT语句时，要注意下列问题：

(1) INPUT保留字之后的可选项“提示字符串”被选用时，执行INPUT语句将不会显示单问号提示符，而显示提示字符串本身（引号对内的字符）。

比如程序：

```
⋮
100  INPUT "X=? ", X
110  PRINT  X
⋮
RUN↙
X=? 100↙
100
```

它等价于下列程序：

```
⋮
100  PRINT "X=" ;
105  INPUT  X
110  PRINT  X
⋮
```

INPUT语句带上“提示字符串”可选项，可以提示输入些什么内容，比单纯一个? 要明确（加用一个PRINT语句虽可达到相同目的，但毕竟多了一个语句）。

提示字符串不得用变量名来代替（即使它已经取得应提示的字符串值）。因为解释程序弄不清你的目的，它是用于提示呢？还是希望从键盘取得数据呢？解释程序只好给出错误信息。

(2) 用户对INPUT语句的回答，必须是常量，不得是变量名，也不得是带运算符号的表达式或函数。

比如：

```
100  INPUT  A, B
RUN↙
? 35, 28.3↙
```

是允许的。而

```
RUN↙
?  B, 3+A↙
```

或

```
RUN↙
? 6, 6*7↙
```

都是错误的。机器会提示“? REENTER”，要你重新输入。在这种情况下，总是重新执行INPUT语句，即使你回答的第一个数据是正确的也必须重新输入。

要求给字符串型变量输入数据时，同样应输入字符串型常量。但在APPLE BASIC中规定：输入字符串型常量时，字符串常量前后的引号对，一般可以略去。又由于数字、小数点、运算符号及$号均可以出现在字符串中，所以当INPUT语句要求给一个字符串型变量输入数据时，下列形式的输入都是合法的：

```
ABCD↙
123↙
12+34*56↙
A*B$↙
A$+B$↙
```

注意，这里的A$，B$都不被当作变量名，而是字符串常量

中的一部分字符。

不过，由于逗号通常被当作数据分隔符来处理，所以如果字符串常量中有逗号时，这个字符串常量必须用引号对把这个字符串常量括起来，否则，会把一个字符串拆成两个字符串。

例如，希望A$取得“AB，CD”，B$取得“EF”。对程序：

```
100  INPUT  A$, B$
110  PRINT  A$, B$
```

如敲：

```
RUN↙
? AB, CD, EF↙
AB                CD
```

把“AB，CD”拆开了，并且丢失了EF。而敲入：

```
RUN↙
? "AB, CD", EF↙
AB, CD                EF
```

这就正确了。

另外，当希望从键盘上输入的字符串常量中头尾有空格时，也要用引号对把这个字符串常量括起来。否则这些空格会被删除掉。

(3) 回答INPUT时输入的各个常量，必须和INPUT之后的各个变量依序属于同一的或相容的数据类型。

如程序行

```
100  INPUT  A, A$
```

那么用户从键盘上敲入的第一个数据必须是数型常量，第二个数据必须是字符串型常量。

例3. 把例1的程序予以改写，学生的5个考分用INPUT输入。程序如下：

```
10  INPUT  A1, A2, A3, A4, A5
20  A=A1+A2+A3+A4+A5
```

```
30  C=A/5
40  PRINT "TOTAL=" ; A,  "MEAN=" ; C
50  END
RUN↙
?   75, 82, 90, 63, 84↙
TOTAL = 394                    MEAN = 78.8
```

§5. 读数语句和置数语句(READ和DATA)

使用读数语句和置数语句，也可以使变量取得数据。

这两个语句的格式分别是：

〔行号〕 READ 变量名〔{，变量名}〕

行号 DATA〔常量〔{，常量}〕〕

在运行一个程序时，如果遇到READ语句，保留字READ之后的各个变量，会依次地从DATA语句中取得数据。

象前面所举的那个例子，一个学生的5个考分，可通过READ/DATA语句取得：

```
10  DATA  75, 82, 90, 63, 84
20  READ  A1, A2, A3, A4, A5
 ⋮
```

这两行程序执行之后，A1，A2，A3，A4，A5这五个变量分别得到数据75，82，90，63，84。

对READ和DATA语句，要明确以下几点：

(1) DATA语句仅仅提供数据，并不执行，它是一个非执行语句，因此可以写在任何一个行号中。而且只要保持数据原来的顺序，一个DATA语句可以拆写成几个DATA语句。

如上面第10行的DATA语句，可以改写成：

```
10  DATA  75, 82, 90
25  DATA  63, 84
```

类似地，一个READ语句，只要不改变所跟变量的顺序，也可拆成几个READ语句。如前面第20行程序，可改写成：

```
20  READ  A1, A2
21  READ  A3, A4, A5
```

对初学者来说，要特别小心：在DATA语句中最后一个常量之后，不要再跟一个逗号“，”。那样会无意中增加一个数据，从而造成混乱。此外，DATA语句中的数据，不能是变量名或带运算符号的表达式，也不能是函数。

(2) DATA语句中常量的个数，一般地不应少于READ语句中变量的个数。以免某些变量得不到数据。进一步讲，一个程序允许有DATA语句无READ语句；但不得有READ语句而无DATA语句。

(3) DATA语句中各常量的数据类型应与READ中对应的变量的数据类型一致或相容。

(4) DATA中若有字符串型常量，通常这个字符串的前后引号对可以略去。但若是希望在串值中嵌入逗号或带有领头的空格、拖尾的空格时，这个字符串常量必须用引号对括起来。

§6. 恢复数据区语句(RESTORE)

用READ从DATA中读出一批数据后，如果同样一批数据又需要读给某些变量（这些变量名很可能和刚刚读得数据的那些变量名不同，因而需另写READ语句），是否需要再把那些数据用DATA语句重写一遍呢？完全不必。RESTORE语句可以使其后的READ语句从程序中的第一个DATA语句重新读入数据。这称为恢复数据区。

恢复数据区语句的格式是：

〔行号〕 RESTORE

在执行任何一个BASIC程序时，如果第一次遇到READ语句，机器会设置一个指针，使其指向程序中的第一个DATA语句中的第一个数据处。并将该数据读给READ中的第一个变量名，随即将这个数据指针后移一个数据位置（指向DATA语句中的下个数据。一个DATA语句中没有下一个数据时，则指向下一个 DATA 语句的第一个数据）。当需要再给一个变量读数据时，将数据指针当前指向的数据读给相应变量，数据指针再移后一个数据位置，如此类推。这样，所有DATA语句中的数据，构成了一个数据区。其中的数据和READ所跟的变量名有着一一对应的关系。当程序执行中遇到RESTORE语句时，数据指针将会回退到数据区之首，即重新指向程序中第一个 DATA 语句中的第一个数 据。于 是，RESTORE之后的READ将重新从数据区的头上开始读取数据。

读懂下列程序，并弄清它的运行结果是不困难的。

```
10  READ  A, B$
20  READ  C, D
30  RESTORE
40  READ  E, F$
50  PRINT  A, B$, C, D, E, F$
60  DATA  1, ABC, 15, 18
70  END
RUN
1                ABC              15
18               1                ABC
```

有时，数据区的数据在某一遍读入时没有使用价值，但数据指针又不会自行跳过这些数据，这可以用一种“假读”的方法：让一些没有用处的（或马上会重新得到数据的）变量来读得那些没有用处的数据。

假定有下表：

1	2	3	4	5	6	7
A	B		C	D		E
	F	G			H	I

它的第一行表明有七个数，相当于DATA中的七个数据；第二行和第三行是一些变量名，希望这些变量名读得所在列的数据。程序可写成：

```
10  READ  A, B, L, C, D, L, E
20  RESTORE
30  READ  F, F, G, H, H, H,I
40  PRINT  A, B, C, D, E, F, G, H, I
50  DATA  1, 2, 3, 4, 5, 6, 7
60  END
RUN
1             2             4
5             7             2
3             6             7
```

执行第10行时，变量L先后读得3和6。而L变量是不需要的。执行第30行时，变量F先读得1，接着又读得2，原先的值1被冲掉了。同样，变量H虽曾先后读得过4和5，但马上又为读入的6冲掉了。这样就实现了题目的要求。

例4. 分别求$S=\frac{1}{2}at^2$和$X=v_0t-\frac{1}{2}gt^2$，其中a和g的值都等于9.8，$t=5$，$v_0=4$。

程序如下：

```
10  READ  A, T
20  DATA  9.8, 5, 4
30  S=A*T*T/2
```

```
40  PRINT "S=" ; S
50  RESTORE
60  READ  G, T, V0
70  X=V0*T-G*T*T/2
80  PRINT "X=" ; X
90  END
```

§7. 三种取得数据的语句的比较

LET、INPUT和READ/DATA，均可使变量取得数据。它们各有各的特点。

LET语句是使变量从表达式取得数据的唯一方法。所以，任何中间结果只能通过赋值语句将其值记入某一变量名下，这是它的一个最大优点。但一个赋值语句只能使一个变量取得数据，如果程序中有较多的原始数据需记入若干变量中时，用赋值语句一一赋值会使程序变得很长，而且也影响运行速度。如果原始数据需改动时，还要修改程序本身。这是它的缺点。

READ/DATA。一个READ语句可以给多个变量提供数据，但原始数据需改动时，也得修改程序。

一个INPUT语句也可以使多个变量得到数据，而且这些数据是在程序执行过程中由键盘输入的，原始数据需改动时从键盘输入新数据就行了，不必改动BASIC程序。但它的缺点是经常要等待用户输入，运行时间长；而且当一个数据送错时，得重新输入该INPUT语句所需的全部数据，较为麻烦。

三种取得数据的语句各有利弊，这要在编写程序时根据情况择优选用。

例5. 某人买苹果5斤，每斤0.52元；买梨3斤，每斤0.35元；买桔子6斤，每斤0.61元。问共花了多少钱。

下面用三种取得数据的语句来写成四个程序，以资比较：

程序1：

```
10  AQ = 5
20  AP = 0.52
30  PQ = 3
40  PP = 0.35
50  OQ = 6
60  OP = 0.61
70  GT = AQ * AP + PQ * PP + OQ * OP
80  PRINT "GRAND  TOTAL  IS: " , GT
90  END
```

程序2：

```
10  READ  AQ, AP, PQ, PP, OQ, OP
20  GT = AQ * AP + PQ * PP + OQ * OP
30  PRINT "GRAND  TOTAL  IS:", GT
40  DATA  5, .52, 3, .35, 6, .61
50  END
```

程序3：

```
10  INPUT "APPLE  QTY  AND  PRICE:"; AQ, AP
20  INPUT "PEAR  QTY  AND  PRICE:"; PQ, PP
30  INPUT "ORANGE QTY AND PRICE:"; OQ, OP
40  GT = AQ * AP + PQ * PP + OQ * OP
50  PRINT "GRAND TOTAL  IS:", GT
60  END
```

其中10行，20行，30行可简化成一个程序行：

```
10  INPUT AQ, AP, PQ, PP, OQ, OP
```

程序4：

```
10  INPUT "APPLE  QTY", AQ
20  INPUT "PEAR  QTY", PQ
30  INPUT "ORANGE QTY", OQ
```

```
40  READ  AP, PP, OP
50  GT =  AQ * AP + PQ * PP + OQ * OP
60  PRINT "GRAND  TOTAL  IS:"; GT
70  DATA  0.52, 0.35, 0.61
80  END
```

比较上面四个程序，可以看出：第一个程序语句最多，第二个程序（和简化后的第三个程序）语句最少。前两个程序执行时速度快，但若是买主买的数量改变时，要修改源程序。后两个程序用了INPUT语句，可以适应每次买的数量不同的情况。第三个程序还适用于水果单价有可能涨跌的情况，第四个程序则仅适用于固定水果单价情况。后两种程序运行速度慢，但无论那一个程序，作为总金额的变量GT的值，总是通过赋值语句得到的。

§8. 算术函数的使用

在第二章里，我们介绍了苹果机的十一个算术函数。这些函数都具有算术值，它们总是作为算术表达式（或算术表达式的一项）出现的，因此大都用在赋值语句或打印语句中。

这一节，我们简要地介绍一下它们的使用方法。

算术函数的格式都是一样的：

函数名（算术表达式）

此处括号中的算术表达式，正是第二章中描述各算术函数时括号中的X。

指数函数只有一个：EXP。函数EXP(X)是用来求 e^x 的。这里e = 2.71828…

比如计算e的10次方，可编成如下程序：

```
10  A = 10
20  B = EXP (A)
```

```
30  PRINT  B
40  END
```

它的运行结果是输出

```
22026.4658
```

当然这个程序可以写的更简单些：

```
10  PRINT  EXP (10)
20  END
```

指数函数也可以直接用乘方代替，如：

```
10  B = 2.71828 ↑ 10
20  PRINT  B
```

运行结果输出

```
22026.3177
```

但它的精度稍差，运行速度也稍慢。

对数函数，也只有LOG。LOG(X) 是求 X 的以e为底的自然对数的。

比如求100的以e为底的自然对数，可用程序：

```
10  A = 100
20  B = LOG (A)
30  PRINT  B
40  END
```

运行后，输出

```
4.60517019
```

它表明2.71828的4.60517019次方为100。

对于求以10为底的对数，可用换底公式。比如求10000以10为底的对数，可以用下列程序：

```
10  B = LOG (10000) /LOG (10)
20  PRINT  B
30  END
```

这个程序运行后，输出为

4

这就是说，$10^4 = 10000$

读者可以想象，如果不使用函数LOG，而编写一个求对数的程序，非但烦琐，而其运行时间也会加长。

关于三角函数，苹果BASIC提供了SIN、COS、TAN和 ATN四个（当然由它们还可以衍生出更多的三角函数）。其中前三个函数的参数应取弧度值，后一个函数的结果也是弧度值。

为了将角度换为弧度，只要把角度值乘以

3.14159/180

就可以了。

比如有直角三角形ABC，其中∠BAC = 20°，AB = 100，欲求BC和AC。

那么，可写出如下的程序：

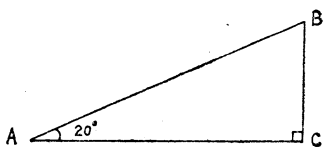


图4—1

```
10  AB = 100
20  BC = AB * SIN (20 * 3.1416/180)
30  AC = AB * COS (20 * 3.1416/180)
40  PRINT "BC = "; BC, "AC = "; AC
50  END
```

程序运行后，输出：

BC = 34.2020911　　AC = 93.9692342

将上述程序略作扩展，再使用SQR函数（对一个数开平方），可以用来验证直角三角形中勾股定理的正确性。

比如将上述程序插入下列两个程序行：

```
45  D=SQR (BC↑2+AC↑2)
46  PRINT  D
```

整个程序运行后，将输出：

```
BC=34.2020911     AC=93.9692342
100
```

这表明：

$$\sqrt{(BC)^2+(AC)^2}=100=AB$$

由于函数的参数可以由算术表达式给出，而函数本身也可视作算术表达式，因而一个函数的参数也可以由函数给出。所以，为要验证勾股定理，也可以使用下面的程序：

```
10  AB=100 : K=20
20  C=3.1416/180
30  PRINT  SQR ((AB*SIN (K*C))↑2
    +(AB*COS (K*C))↑2)
40  END
```

改变第10行的数据，对其它斜边长和其它角度，也可以予以验证。

对于取整函数INT，我们先看下列程序例：

```
10  A=89.6
20  B=45.6754
30  C=1235.87
40  D=5437.98776
50  E=659.56
60  R=A+B+C+D+E
70  S=INT (R)
80  T=INT (R+0.5)
90  U=INT (R*100+0.5)/100
100 PRINT  R, S, T, U
```

程序运行后，输出为：

```
7468.69316        7468              7469
7468.69
```

程序中10～50行，通过赋值语句使变量A、B、C、D和E分别取得相应的数值，60行将它们求和赋给变量R，这个值为7468.69316。程序的70行对它取整，意思是取不大于该数的最大整数，那就是7468。

如果希望取整时，能考虑到四舍五入，应该采取80行的措施：就是将原来的数加上0.5，然后再取整。在上述例子中，这样处理后的结果为7469。

另外，有时要求精确到小数点后规定位数，可以采用90行的办法。在此，可以精确到小数点后两位，并且也考虑了四舍五入，上例的结果经这样处理，输出为7468.69。

取整函数还常常用来求除法的整商及余数。譬如求59÷16的整商与余数，可写出如下程序：

```
10  A = INT (59/16)
20  B = 59 - A * 16
30  PRINT "59/16 = "; A;"…"; B
40  END
```

运行后输出：

```
59/16 = 3…11
```

这个程序可以理解成：十进制的59是十六进制的3B（十进制的11是十六进制的B）。

取整函数对于正的参数而言，可以理解成舍去小数点后的部分，但对负的参数，并不是如此。比如：

```
10  A = - 5.7
20  B = INT (A)
30  C = INT (5 - 6.8)
40  PRINT  B, C
```

运行结果将输出：

-6　　-2

因为，只有－6才是不大于－5.7的最大整数（－5，－4，－3，…都比－5.7大；－6，－7，－8，…都比－5.7小，但其中最大的为－6。同样，－1，0，1都比－1.8大；－2，－3，－4，…都比－1.8小，但其中最大的为－2）。

其它如符号函数、绝对值函数等，较好理解，不再举例。随机函数，在游戏程序中将会看到它们的使用。

§9. PR#与IN#

PR#与IN#是苹果BASIC的两个命令，也是磁盘操作系统的两个命令。但它们不属于基本BASIC的成分。

这两个命令的格式是：

〔行号〕 PR#s

〔行号〕 IN#s

其中s表示一个槽口号码（0～7）。命令的功能是接通（即启动）s号槽口上的设备。IN指的输入设备，PR 指的输出设备。

设磁盘机接在6号槽口上，那么 PR#6或IN#6均可启动磁盘机（因磁盘机既是输出设备，也是输入设备），它会像刚打开电源时那样：把磁盘操作系统装入内存，并调入磁盘上第一个BASIC程序运行之。

再设打印机接在1号槽口上，则PR#1（注意不能用IN#1，因打印机非输入设备）可启动打印机。以后凡能送到屏幕的内容均可打印出来（包括INPUT的提示及键盘输入时的显示）。如果仅希望打印运行时的某些输出，则可在这些输出语句之前加上带行号的PR#1来启动打印机。不需要再打印时用带行号的 PR#0来关闭打印机。

接在0号槽口上的设备规定为显示终端（输出）和键盘（输入）。

通常它们总是被接通的。PR#0仅为了脱开非0号设备。

习　题

(一) 指出下列各程序行中有哪些错误。

(1) 10　X=5, Y=6

(2) 10　X+2=Y+3

(3) 10　"DISTANCE IN ONE TURN IS";
　　3.1416*D

(4)　　INPUT Z,

(5)　　INPUT, Z

(6)　　PRINT DISTANCE IN ONE TURN IS;
　　3.1416*D

(7)　　INPUT 56, 78

(二) 写出下列程序的运行结果。

```
10 LET A$ = "    ×××××"
20 LET G$ = "      ×××"
30 LET F$ = "   ×     ×"
40 LET B$ = "  ×       ×"
50 LET C$ = "(×  O O  ×)"
60 LET E$ = "  ×  <  >  ×"
70 LET D$ = "  ×    V   ×"
80 PRINT A$
90 PRINT B$
100 PRINT C$
110 PRINT D$
120 PRINT E$
130 PRINT F$
140 PRINT G$
150 END
```

（三）写出下列程序的运行结果。

```
10  JOHN = 5
20  GERTRUDE = 10
30  PRINT  JOHN + GERTRUDE; ", ";
40  PRINT  JOE + GENE,
50  PRINT  A + B
60  PRINT  C + JO
70  END
```

（四）写出下列程序的运行结果。

```
10  READ  A, B
20  DATA  1, 2, 3
30  READ  C, D, E
40  RESTORE
50  DATA  4, 5, 6, 7
60  READ  F, G, H, I, J, K, L, M, N
70  RESTORE
80  READ  O
90  DATA  8, 9, 10
100 PRINT  A, B, C, D, E, F, G, H, I, J, K,
    L, M, N, O
110 END
```

（五）用READ/DATA语句编写一个“加法器程序”，求12，43，－54，76，15.7五个数的累加和，并在每加一个数时打印出当时的累加和。

（六）编写一个程序，使从键盘上取得一个正数，以此数为半径，求圆面积和球体积。

第四章　程序流向的转移

在我们前面提到过的所有程序，都是严格地从最小的行号执行起，顺序执行后续的较大行号，最后执行最大的行号。然而在许多有用的程序中常常要突破这种限制，使程序执行时可以向前或向后跳过某些程序段。这一章将要介绍的GOTO,IF…THEN语句，主要就是为了实现这类转移功能的。本章还要向读者说明程序流程图的使用；关系运算符和关系表达式；逻辑运算符和逻辑表达式以及STOP、CONT、REM等语句和程序调试的一般方法等内容。

§1. 程 序 框 图

在日常生活中，一个事件的发展过程往往可以用框图清晰地展示出来。比如，给朋友发一封信，这是一个简单的事件。把这个事件的发展过程用框图一步步表示出来，可以有图4—1所示的那样一些步骤（当然不是唯一的方案）。

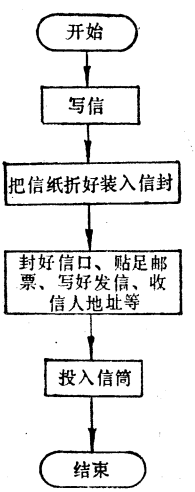


图4—1

BASIC程序也可以用框图画出。譬如前一章提到过的把一个学生5门考试成绩读入、求出并打印总分和平均分的问题，我们曾

为它写过两个程序，但它们的框图都可用图4—2示出。

这样的框图较细，在更多的情况下，框图可以更简练些，读起来更为清晰（特别是对于一些较长的程序来说）。下图（图4--3）是图4—2的简略画法。

图4—2和图4—3都称为程序框图（或称程序流程图）。图4—2是细框图，它甚至可以细到图中一个块对应于程序中的一个语句。先有这类细框图，再转写成程序行很容易。图4—3是粗框图，图

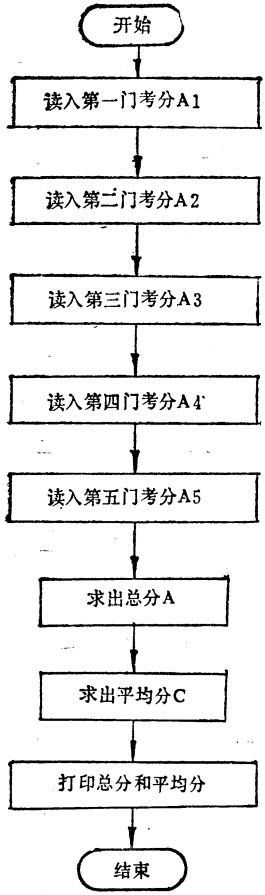


图4—2

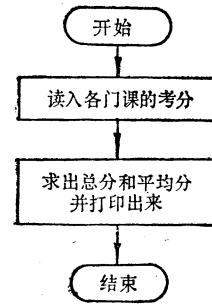


图4—3

中的一个块对应于程序中的一段程序行，它具有独立的一个或几个功能。读粗框图可以较快地归纳出程序的作用。

编写程序时（特别较复杂的程序），最好先画出框图，再写成程序。分析别人的程序时，也应先画出框图。这样既直观又清晰，

也会提高你读写序程的效率。

上述程序框图顺序是执行时的情况。开始和结尾用两边为圆弧的图块表示，一般功能块用矩形表示。有时程序流向有可能产生分支，这时我们用一个菱形框表示。

比如，在进行除法运算时，除数可能不为零，也可能为零。当除数为零时这个除法便不能进行。用框图画出的完善的除法框图如图4—4：

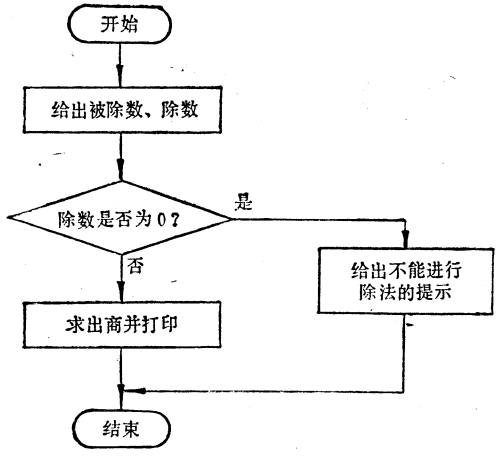


图4—4

在这个框图中，菱形方块表明：当其中的条件满足时转向右边的功能块，否则执行其下边的功能块。

归纳我们上面用过的图块，它们的意义分别由图4—5所示。

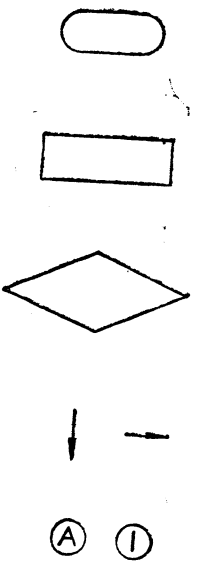


图4—5

始尾块：表示程序的开始或结束。

功能块：表明完成某种(些)功能。

判断框：因情况不同而转向不同支路。

流向线：表示程序执行的前进方向。

连接圈：当一个程序的框图中出现两个以上的圆圈，且其中字母或数字相同时，表明它们应连接在一起。(这可以减少连线)

§2. 无条件转向语句(GOTO)

假定我们设计这样一个程序，它从键盘上读入三个数，每读入一个数时，求一次累加和，并打印出来。

程序如下：

```
10  S=0
20  INPUT  X
30  S=S+X
40  PRINT S
50  INPUT  X
60  S=S+X
```

```
70  PRINT S
80  INPUT  X
90  S=S+X
100 PRINT  S
110  END
```

我们这里仅做三个数的累加，程序已写了十一行。如果要做100个或1000个甚至10000个数的累加，如此写出来的程序将会很长。

但我们不难发现，程序中有一段程序基本一样，那就是20～40这三行程序，反复用了三次。可以想象：若是在执行完40行之后，能让程序转到20行去，读入一个数、累加、打印，再转回20行，这样反复进行，任意多个数的累加、打印都可以实现（只要不出现溢出）。

GOTO语句的功能可以完成这件事。

GOTO语句的格式是：

〔行号〕 GOTO 行号

其功能就是无条件地转到指定的那个行号去继续运行。

这样，上面那个程序就可以改写成：

```
10  S=0
20  INPUT  X
30  S=S+X
40  PRINT  S
50  GOTO  20
60  END
```

程序框图如图4—6所示：

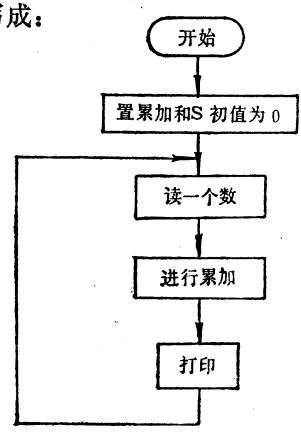


图4—6

GOTO语句的规则很简单，仅要求在保留字 GOTO 之后跟上一个合法的行号，而且这个行号必须在程序中存在。

立即执行的GOTO 语句，可以从指定的行号开始执行内存中的当前程序。它很象 RUN 之后跟上一个行号那种命令一 样。但GOTO行号不会清除内存中当前的一切变量。

下面举一个例子来说明 GOTO 语句的用法。把华氏温度F转化成摄氏温度C和绝对温度Z，可用如下两个公式：

$$C=(5/9)*(F-32)$$

$$Z=273.16+C$$

下面的程序可以完成若干个温度的转换：

```
10  PRINT "F",  "C",  "Z"
20  READ  F
30  C=(5/9)*(F-32)
40  Z=273.16+C
50  PRINT  F, C, Z
60  GOTO  20
100 DATA 21, 130, 31, 32, 90, 88
110 END
RUN
F              C              Z
21            -6.11111111     267.048889
130           54.4444444      327.604444
31            -.555555556     272.604444
32            0               273.16
90            32.2222222      305.382222
88            31.1111111      304.271111
? OUT OF DATA ERROR IN 20
```

这个程序反复执行六次之后，已将DATA中的所有数据读完。但它仍会从第60行转回第20行，第七次执 行 READ F，这是无法实现的。于是给出一行出错信息。虽然它不影响所希望得到的结果，但给出出错信息总不那么使人舒服。事实上，在一个程序

中仅用GOTO 语句来控制程序的转移，总会出现一些不十分完满的现象。在较前的一个程序中，由于GOTO 语句的使用，程序永远执行不到带有END的第60行。在运行那个程序时，永远不会结束，最后不得不由用户敲入CTRL—C或RESET键使其停止运行，否则它总在等待你从键盘上敲入数据。

§3．条件转移语句(IF…THEN)的概念

一、什么是条件转移语句

对于前面那个若干个数累加的程序，如果我们有言在先：当从键盘上输入一个0时，表示累加的结束。那么，那个程序可以改写如下：

```
10  S=0
20  INPUT  X
30  IF  X=0  THEN  END
40  S=S+X
50  PRINT  S
60  GOTO  20
```

这个程序的第30行，就是一个条件转移语句。它的意思是：当（由键盘输入的）X为0时，执行END语句，使程序结束；否则执行40行，50行…。这就是说，执行END语句是有条件的。在这里，条件就是X = 0。

其程序框图如图4—7所示。

用条件语句改写前面那个温度转换程序，可以避免运行时给出出错提示。我们约定，当READ F 语句读给F的值为－1000时（华氏零下1000度是不可能出现的），认为所有转换都做完了。

程序如下：

```
10  PRINT "F",  "C",  "Z"
```

```
 20  READ  F
 30  IF  F=-1000  THEN  110
 40  C=519*(F-32)
 50  Z=273.16+C
 60  PRINT  F, C, Z
 70  GOTO  20
100  DATA  21, 130, 31, 32, 90, 88, -1000
110  END
```

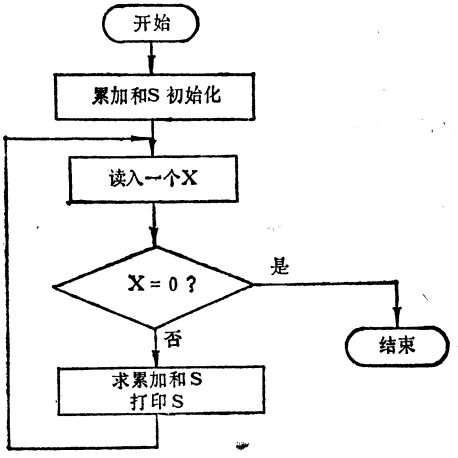


图4—7

这个程序中的第30行，也是一个条件转移语句，它的意思是：当（读入的）F值为负1000时，转移到第110行去。实现这一转移的条件是F = －1000。

我们把上面两个程序中控制转移的条 件：X = 0，F = －1000，称为条件式。

二、条件语句的格式及执行方法

条件语句的通用格式为：

〔行号〕 IF 条件式 THEN 语句〔{:语句}〕

其中，保留字THEN之后的语句，称为THEN子句。

条件语句的执行方法是：当条件满足时，执行THEN子句；当条件不满足时，跳过THEN子句，转后续程序行继续运行。

条件语句因THEN子句不同，书写时或执行时各有所异，大体有以下一些情形：

1.THEN子句仅为一个GOTO语句，即形如：

〔行号〕 IF 条件式 THEN GOTO 行号

对这种条件语句，苹果机上允许省略保留字THEN和GOTO中的任一个。

于是，可用

〔行号〕 IF 条件式 THEN 行号

或

〔行号〕 IF 条件式 GOTO 行号

代替

〔行号〕 IF 条件式 THEN GOTO 行号

执行这种条件语句时，如果条件满足，就转移到指定的行号去；否则执行本条件语句的后续程序行。

其流程图如下（图4—8）：

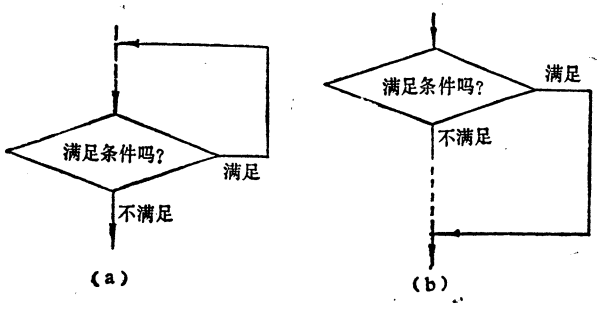


图4—8

其中图（a）转向较小的行号；图（b）转向较大的行号。

请读者考虑：下列程序行是什么意思？

```
100 INPUT X : IF X=0 THEN 100
:
```

2.THEN子句为多个语句，但其中不存在GOTO语句的情形，如：

〔行号〕 IF 条件式 THEN 语句1 ：语句2 ：语句3

其中语句1，语句2，语句3均非GOTO语句。

这种条件语句执行时，当满足条件，则依序执行语句1,语句2，语句3，再执行后续程序行；当条件不满足，则跳过语句1，语句2，语句3，直接执行后续程序行。

其流程图如图4—9所示：

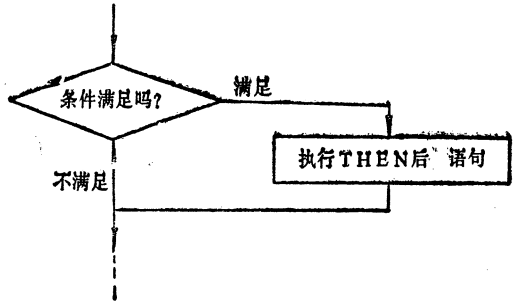


图4—9

3.THEN子句为多个语句，且最后一个语句为 GOTO 语句的情形，如：

〔行号〕 IF 条件式 THEN 语句1 ：语句2 ：GOTO 行号

其中语句1，语句2均非GOTO语句。

执行这种语句时，当条件满足，则执行语句1，语句2，然后执行 GOTO 语句，转移到指定的程序行；当条件不满足，则跳过THEN子句顺序执行后续程序行。

其流程图如图4—10所示：

图（a）是GOTO到较大的行号，而图（b）则是GOTO到较小的行号。

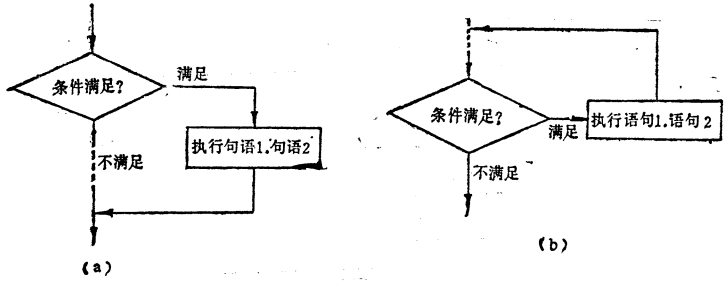


图4—10

请读者注意，当THEN子句中有GOTO语句，而又不是子句的最后一个语句时，其中GOTO 语句之后的语句，永远不会被执行。因此是毫无用处的。换言之，THEN 子句中，GOTO 语句之后不应再有语句。

4.THEN子句也可以是条件语句。这称为条件语句的嵌套。下列程序段的100行就用了嵌套的条件语句。

```
100  IF  A$="X" THEN IF A=0 THEN 120
110  PRINT  A$
120  PRINT  A
```

这段程序框图如下（图4—11）：

不难看出，从100行控制转移到120行，实际是两个并列的条件：A$="X" 和A=0。只要其中一个条件不满足，则顺序执行完110行之后才执行120行。

三、在苹果机上使用条件语句时，要避免在保留字THEN之左邻字符为字母A的情况。

比如

```
IF  B=A  THEN  行号
```

这是不允许的。因为这时解释程序把 AT 两个字母当作保留

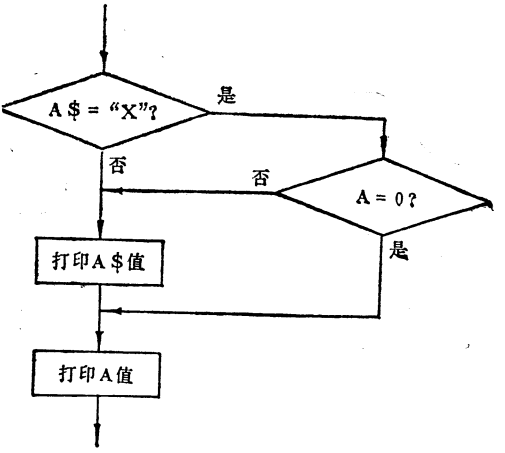


图4—11

字看待（苹果BASIC有AT保留字），这样就乱套了。

四、由于 IF 语句中条件式不成立时本行的后续语句不再执行，所以一个程序行中不允许有两个以上并立的IF语句。

如：

```
100 IF  A=5  THEN  300 : IF  A<5  THEN  200
 ⋮
```

是不允许的。解释程序遇到它，认为是错误的。

五、在判别一个算术表达式是否和另一个算术表达式的值相等时，一定要作慎密的考虑。因为两个算术表达式的值那怕只相差亿万分之一，也将被认为是不相等的。

譬如，3的4次方为81。但如果想通过运行下列程序，以求出3的多少次方为81，将是办不到的：

```
10  A=0
20  B=3↑A
```

```
30  IF  B=81  THEN  50
40  A=A+1 : GOTO  20
50  PRINT "3↑"; A; "=81"
60  END
```

这个程序的思路是正确的，但是 B=81这个条件将永远不会达到，因而程序会一直运行下去，直到B的值大到机器无法表示，产生溢出，并给出出错信息。50行是永远不会执行的。

这是为什么呢？这是因为机器在做3↑4时得到的结果并不是81，而是81.0000001。

类似的情况很多，比如7↑2，苹果机算出来的不是49，而是49.0000001。

下面我们再看一个程序：

```
10  A=-0.2+0.1
20  B=A+0.1
30  PRINT  A, B
40  END
```

运行后将输出

```
-0.1                    -4.36557457E-11
```

可见B的值并不是0，而是非常接近0的一个数。

因此，在对两个算术表达式作相等比较时，要作周密考虑，否则达不到目的。

对于求3的多少次方为81的程序,可以将30行改为下面的样子：

```
30  IF  INT(B+0.000001)=81  THEN  50
```

或

```
30  IF  ABS(B-81)<0.000001  THEN  50
```

都可以。它们都表明：只要 B和81非常接近时，就被认为是相等的。

§4. 条 件 式

在上节所举的一些例子中，我们用到了条件式X=0，F=-1000，A$="X"，A=0等，它们只是“相等”的一种条件。实际上条件要复杂得多。想要更好地描述它们，要用关系表达式和逻辑表达式。

一、关系运算符和关系表达式

关系运算符又称比较运算符，有以下六个：

=	等于	相当于数学上的=
>=或=>	大于等于	相当于数学上的≥
>	大于	相当于数学上的>
<	小于	相当于数学上的<
<=或=<	小于等于	相当于数学上的≤
<>或><	不等于	相当于数学上的≠

关系运算符可以用来对两个算术表达式（或两个字符串表达式）进行比较，或者说判断它们之间的某种关系是否成立。如A>5就是判断变量A大于5的这种关系是否成立；C=D就是判断变量C等于变量D的这种关系是否成立。当某种关系成立时(也可以说是满足某种条件)，我们说这种关系（或条件）是真的，否则就说是假的。这种“真”与“假”称为逻辑值。

换言之，两个数据类型相同（或相容）的表达式用一个关系运算符连接起来，就构成了关系表达式。关系表达式具有逻辑值：或为真，或为假。真用数字1表示，假用数字0表示。

下面三个式子，都是关系表达式，都具有逻辑值：

3<>2	成立	结果为1
2>=5	不成立	结果为0
A>A-3	成立	结果为1

最后一个表达式中，有两个运算符：＞和－，按运算符的优先顺序，应先作减（求出A－3的结果），再比较（A是否大于A－3的结果）。

下列各式均为非法的关系表达式：

"A"＞3＋2

5＝A$

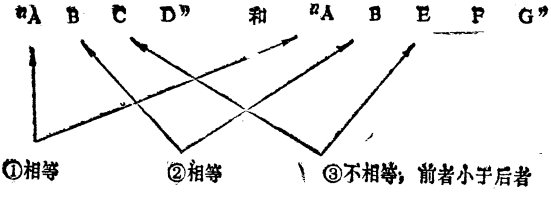
因为关系运算符前后的表达式属于不相容的两种数据类型。

两个字符串表达式进行比较时，是从左到右依次取两个字符串值（常量）中的对应字符的ASCII码进行比较的。

ASCII码是美国信息交换标准码（American Standard Code for Information Interechange）的简称。它对每个字符都给出了编码（详见附录1）。例如，字符A的编码为65，字符B的编码为66，…字符0的编码为48，字符1的编码为49，…。这里按编码大小，顺序列出有关字符：

空格	!	"	#	$	%	&	'	(
)	*	+	,	-	.	/	0	1
2	3	4	5	6	7	8	9	:
;	<	=	>	?	@	A	B	C
D	E	F	G	H	I	J	K	L
M	N	O	P	Q	R	S	T	U
V	W	X	Y	Z	[	\	]	↑

因此，下列两个字符串值进行比较时，将依①②③…的顺序进行：



于是，下列三个关系表达式

“ABCD”＜“ABEFG”

“ABCD”＜＝“ABEFG”

“ABCD”＜＞“ABEFG”

都成立，逻辑值为真，用1表示。

而另三个关系表达式

“ABCD”＞“ABEFG”

“ABCD”＞＝“ABEFG”

“ABCD”＝“ABEFG”

都不成立。其逻辑值均为假，用0表示。

对两个字符串作这种从左到右逐个字符比较时，只要出现某对应位置上的两个字符不同，它们的大小关系就是两个字符串的大小关系，余下的字符便不再进行比较。因此，所谓两个字符串相等，是指引号对内的字符（包括空格符）一个个顺序对应完全一样。下列两个字符串是不等的：

“ABCD”

“ABCD ”

即使它们之间只在尾巴上差一个空格。

二、逻辑运算符与逻辑表达式

逻辑运算符有以下三个：

NOT　　逻辑**非**（逻辑反）

AND　　逻辑**与**（逻辑积）

OR　　逻辑**或**（逻辑和）

用逻辑运算符连接具有逻辑值的表达式，构成逻辑表达式。逻辑表达式的值也是逻辑值，真用1表示，假用0表示。

设A与B分别表示两个具有逻辑值的表达式，那么逻辑运算规则如下：

A	B	A AND B
0	0	0
0	1	0
1	0	0
1	1	1

做 AND 运算时，只有该运算符前后两个表达式都具有真值，运算结果才真，否则为假。

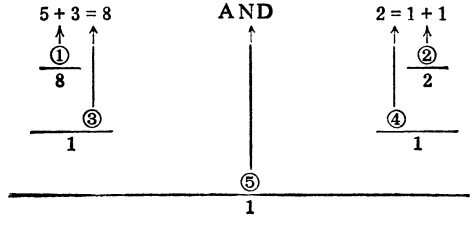
A	B	A OR B
0	0	0
0	1	1
1	0	1
1	1	1

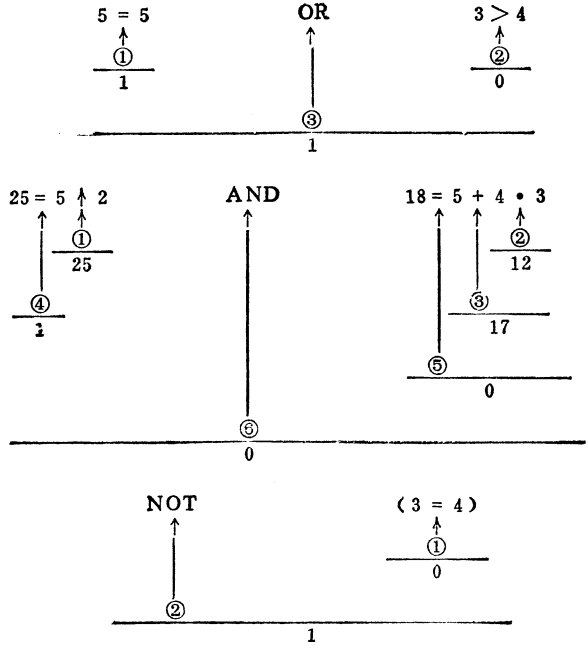
做 OR 运算时，只要该运算符前后的两个表达式中有一个具有真值，则运算结果为真，否则为假。

A	NOT A
0	1
1	0

逻辑运算符NOT是一个单边运算符，它只允许在其后带一个具有逻辑值的表达式。运算结果正好是那个表达式的逻辑值之反。

遵照运算符的优先顺序规则，不难算出下列各逻辑表达式的逻辑值（小圆圈中的数字表明运算的次序，下横线表示运算对象的范围，横线下的值为该次运算的结果）：





三、条件式的进一步描述

关系表达式和逻辑表达式都具有逻辑值，这里合称它们为条件式。

一个条件式成立与否，实质上就是看其逻辑值是否为真。为真时，条件成立；为假时，条件不成立。

进一步讲，既然逻辑值的真和假分别用算术值1和0表示，那么条件式（也就是关系表达式或逻辑表达式）是否可以称为算术表达式呢？可以的。事实上在苹果机的说明书上就是用算术表达式来描述条件式的。只是应该记住：凡由关系运算符或逻辑运算符连接成的这类特殊的算术表达式，它们的结果只能是1或0，不会具有1和0以外的任何算术值。

因此，下列各立即执行的PRINT语句所得到结果是很容易理

解的：

```
PRINT 5=3+2
1
PRINT (5=3+2)+10
11
PRINT (5=3+2)+10 +NOT 0
12
PRINT (5=3+2)+10+  NOT 10
11
PRINT (5=3+2)+10+("ABCD" ="ABCD")
12
```

而语句：

```
A=3=4
```

解释程序把左边的“=”号当作赋值号，把右边的“=”号当作关系运算符的等号。于是赋值号右部的表达式的值为0。这样这个语句成为合法的赋值语句，其结果是使A取得值0。

同样，语句：

```
A=B=C
```

也是合法的赋值语句，其结果是使变量A取得值1（当B=C时）或0（当B<>C时）。

反过来，一般的算术表达式，如果出现在语句中需要出现条件式的地方，它将取得逻辑值1或0。其取值规则是：当算术表达式的值非零时，逻辑值为真（取1）；当算术表达式的值为零时，逻辑值为假（取0）。从而算术值0和逻辑值假都用0表示，意义一致起来了。

因此，语句：

```
IF B THEN 行号
```

等价于语句：

```
IF B<>0 THEN 行号
```

而语句

IF NOT B THEN 行号

等价于

IF B=0 THEN 行号

§5. 应用程序举例

例1.某单位对用户每月用水按下述规定计费：

1. 每月以户为单位结算一次水费；

2. 当一户用水量平均每人不超过两吨时，每人收费一角；

3. 当一户用水量平均每人多于两吨时，超过两吨的部分每吨按0.16元计费；

4. 当一户用水量平均每人不足0.1吨时，以当月该户无人用水对待，只收该户0.10元维修费。

为编写此程序，先画出如下框图（见图4—12）：

程序如下：

```
10 INPUT S, N
20 IF S<0.1*N THEN T=0.1 : GOTO 70
30 T=0.1*N
40 S=S-N*2
50 IF S<0 THEN 70
60 T=T+S*0.16
70 PRINT T
80 END
```

运行这个程序时，一开始就要求操作者从键盘上输入某用户的用水量（S）及人数（N）。然后算出该户应交多少水费（T）并打印出来。

例2.设计一个程序，用来使低年级的小学生在计算机上作加

法练习。设被加数和加数均为100以内的整数。可以让每个学生

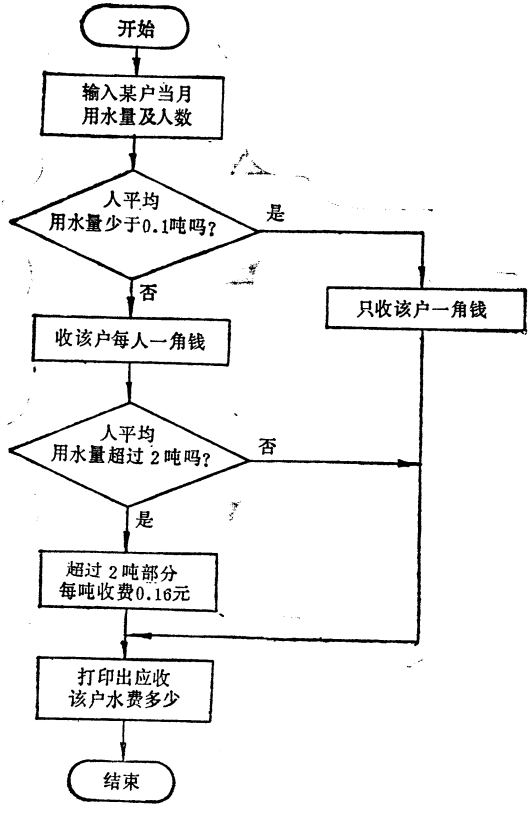


图4—12

做五题。每题最多允许做四次，第一次做对时给100分，第二次做对时给75分，第三次做对时给50分，第四次做对时给25分。四次都没做对，给 0 分。第一题做对后（或已做完四次）转做第二题。五题结束后，给出得分总分和平均分。为使题目的加数和被加数不是固定的，用随机函数来产生。

程序流程图如下（图4—13）：

使用随机函数RND可以产生0到1之间的随机数，将这种数乘

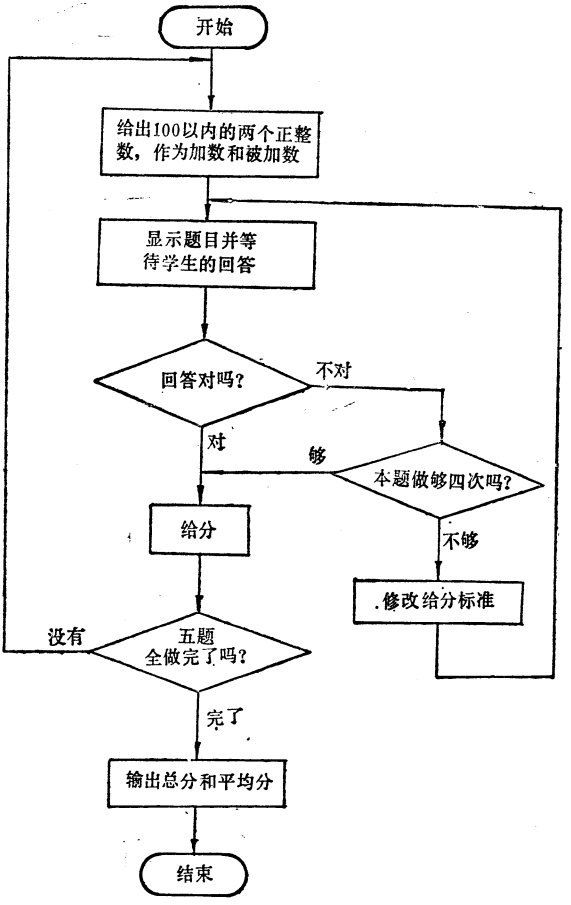


图 4—13

以100再取整，结果总是小于100的正整数。

程序可写成：

```
10 S=0 : M=1
20 N=1
```

```
30 A = INT (RND(1) * 100)
40 B = INT (RND(1) * 100)
50 HOME : PRINT:PRINT
60 PRINT "PROBLEM NO. "; M
70 PRINT A; "+"; B; "=";
80 INPUT C
90 PRINT
100 IF C=A+B THEN 150
110 N=N+1
120 IF N>4 THEN 160
130 PRINT "HMMM··· I GET A DIFFERENT ANSWER."
140 GOTO 50
150 PRINT "RIGHT ON ··· GOOD WORK!!!"
160 S=S+100-(N-1)*25
170 M=M+1
180 IF M<=5 THEN 20
190 PRINT : PRINT "TOTAL:"; S, "MEAN:"; S/5
200 END
```

在这个程序中，我们用变量M表示第几个题，程序 170 行的M = M + 1指明下次该做的题号；变量N表示某一题做的次数，110行的N = N + 1指明下次为第N次；变量 S表明得了多少分，给分标准在160行中给出。程序中的几个不带输出项的 PRINT 语句和HOME，是为了美化屏幕的输出方式。

类似地，不难写出减法、乘法和除法的练习程序。但减法和除法各有特殊性，在小学生没有学习负数的概念以前，要确保被减数大于减数；在学生没有学习分数和小数以前，也要确保被除数是除数的整数倍；此外除法还得确保除数不为零。

在程序中满足上面这些特殊性并不困难。比如说，当被减数A小于减数B时，交换 A、B的值就可以了。再比如，当产生的随

机数为0时，不能当除数，这时可重新产生随机数（只需其值不接近于零，就可作除数）。至于确保被除数为除数的整数倍，可以这样来处理：取机器产生的两个随机数分别当作做除法时的一个除数和一个商数，这时机器不难算出被除数应为多少（除数乘以商数之积）。这个算出的被除数必然为除数的整数倍。

例3．在一个程序中，用DATA语句把一个班的学生的姓名、性别、年龄都登记下来，通过程序的运行，把这个班女同学（或男同学）中，年龄在某个范围内（如14岁和15岁）的都显示出来。

这个程序的框图可以是（图4—14）。

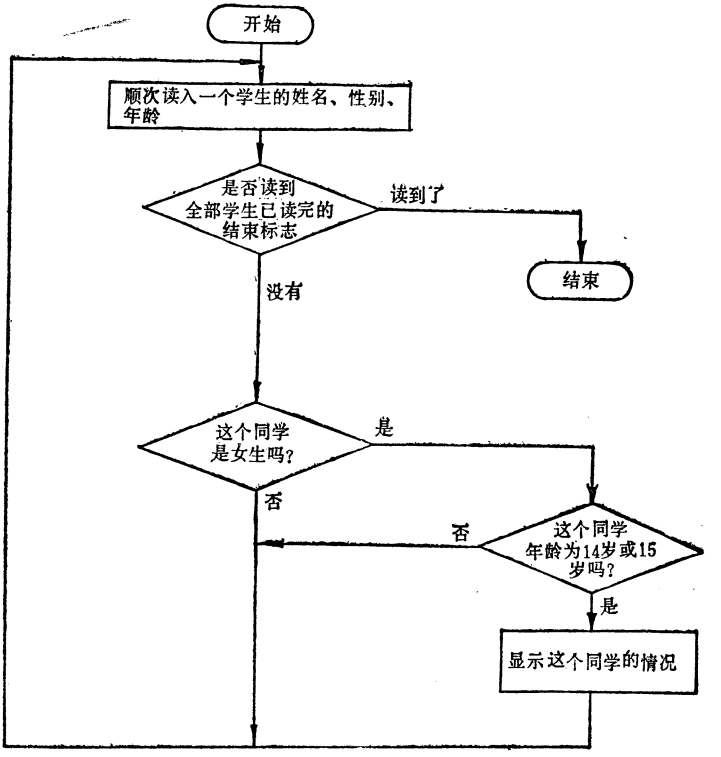


图4—14

假定：

第一个学生叫石 如 玉 (SHI RU YU)，男 (MALE)，15岁；

第二个学生叫廖燕 (LIAO YAN)，女 (FEMALE)，13岁；

……

最后一个学生叫姚致富(YAO ZHI FU)，男(MALE)，14岁。

把这些同学的情况依次放在DATA语句中，并把END，END，0作为结束标志写在DATA语句的数据项之尾。程序如下：

```
10  C$ = "FEMALE"
20  READ A$, B$, C
30  IF A$ = "END" THEN 200
40  IF B$ = C$  AND  (C = 14  OR  C = 15)  THEN PRINT
    A$, B$, C
50  GOTO 20
100 DATA "SHI RU YU", "MALE", 15, "LIAO YAN",
    "FELAME", 13
 ⋮
190 DATA "YAO ZHI FU", "MALE", 14,
    "END", "END", 0
200 END
```

象这种从若干个数据中查找所需要的内容的方法，叫做“查找”，或“检索”。

上面程序中的第40行，相当于下列程序段：

```
40 IF B$<>C$ THEN 50
41 IF C<14 THEN 50
42 IF C>15 THEN 50
43 PRINT A$, B$, C
```

可见，由于一个逻辑表达式中可以反映多种条件，能使程序大大简化，并能节省程序所占用的内存空间和执行时间。

例4．编写一个程序，只要给定A，B，C三个数，就能求出一元二次方程：

$$AX^2+BX+C=0$$

的两个根。

上一章我们举出过这个程序例，但那里作了两个限定：

$A\neq 0$；

$D=B^2-4AC>0$

目的是为了求得两个实根。这里我们把第二个限定取消，使得到的根，可能是两个实根，也可能是两个复根，还可能是重根。

框图（图4—15）如下：

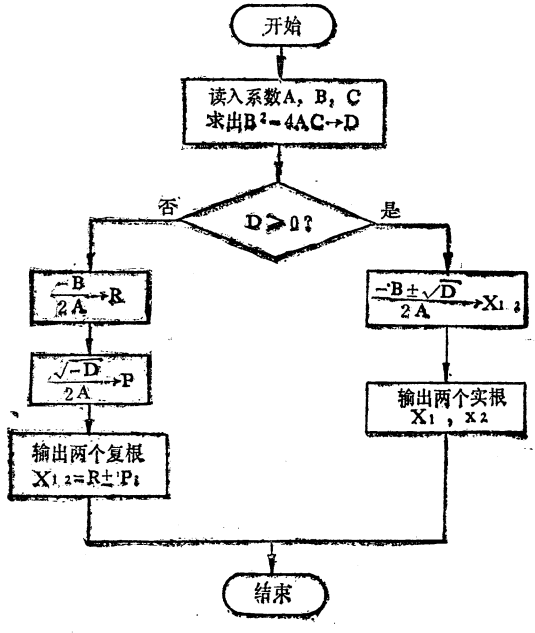


图 4—15

相应的程序如下：

```
10  INPUT "A, B, C"; A, B, C
20  D=B↑2-4*A*C
30  IF D<0 THEN 80
40  X1=(-B+SQR(D))/(2*A)
50  X2=(-B-SQR(D))/(2*A)
60  PRINT "X1=";X1,"X2=";X2
70  GOTO 110
80  R=-B/(2*A)
90  P=SQR(-D)/(2*A)
100 PRINT "X1=";R;"+I";P,"X2=";R;"-I";P
110 END
```

§6. 暂停语句 (STOP) 及调试程序的一般方法

暂停语句的格式是：

〔行号〕 STOP

STOP语句的功能是：当程序运行中遇到这个语句时，会使程序暂停运行，并给出停在那一个程序行（实际上就是STOP语句所在的行号）的信息。

STOP语句与END语句的区别在于：用STOP语句停下来的程序，可以从停下来的地方继续运行下去。而END停下来的程序，一般不能从停下来的地方继续运行下去（除非指明从那个行号开始运行）。

CTRL—C 命令（由键盘上直接敲入的）相当于插入一个STOP命令。它起STOP命令的作用。只是它中断的地方不是由用户确定的某一程序行，而是敲入CTRL—C命令时程序刚执行完的那一行。

由STOP或CTRL—C暂停执行的程序，可用专设的命令CONT

使程序由中断处继续执行下去。

CONT的格式是：

〔行号〕 CONT

虽然CONT和STOP都允许使用于两种执行方式中，但 CONT主要用于立即执行方式，而STOP则主要用于延迟执行方式。

STOP和CONT常常用来帮助用户在机器上调试一个程序。当用户对一个程序不放心，或者运行中有错误时，可以在程序中适当的地方插入一个带STOP语句的程序行，然后运行这个程序。当执行到 STOP 语句时，程序暂停运行。这时你可以用立即执行的PRINT语句输出有关变量的当前值予以检查，以确定前面一段程序是否正确。如发现有错（有关变量当前值不对），分析错误的原因，修改错误的程序行（就是重新敲入正确的程序行，或删除错误的程序行），然后再从头试运行。如果STOP停下来时检查各有关变量正确无误，可在其后的适当地方再插入带 STOP 语句的程序行做类似的检查。当某次 STOP 停止了程序的运行，而又没有发现错误，可直接敲入CONT 命令使程序继续运行下去（从头运行会浪费时间）。

当然，任何先于STOP语句执行的程序段如果没有错误时，这个STOP语句可以删去，以免这种不必要的暂停浪费机时。不言而喻，整个程序全部调试完毕后，所有 STOP 语句均应删除（除非把STOP当END使用）。

用立即执行的GOTO语句也可使暂停下来的（甚至是END 停下来的）程序继续执行。但保留字GOTO之后要跟上适当的行号。行号给的不对，会产生不正确的结果。要特别注意。

§7. 注释语句 (REM)

注释语句的格式是：

〔行号〕 REM {字符}

REM是 REMARK（注释）一词的缩写。其作用是用来对程序或程序段作一些简要的说明，使读程序的人知道这个程序或这段程序是干什么的。这段说明性的文字直接写在保留字 REM 之后，不必用引号对括起来，因此引号也可以是文字中的内容。这段文字是以RETURN为结束符的，所以冒号，逗分，分号、括号甚至保留字等也可写入。

注释语句是非执行语句，在程序运行时毫无作用，解释程序会自动跳过去。

例5. 下面是一个统计程序。将一批数据写入DATA 语句中，程序可以把数据的个数，总和及平均值计算出来。在DATA 中用1E＋38作为"终结"的指针。

程序中用了几个注释语句，说明程序的功能和各段程序的意义。这样的程序就好读多了。

```
100 REM A FRIENDLY "MEAN" PROGRAM
110 REM INITIALIZE T(FOR TOTAL) AND
    N(FOR NUMBER OF NUMBERS)
120 T=0 : N=0
130 REM READ A NUMBER, X. IF IT IS THE FLAG
    1E+38
140 REM GOTO PRINTOUT. OTHERWISE, UPDATE T
    AND X
150 READ X : IF X=1E+38 THEN 180
160 T=T+X : N=N+1 : GOTO 150
170 REM PRINT ANSWERS
180 PRINT "N=";N
190 PRINT "TOTAL=";T
200 PRINT "MEAN=";T/N
250 DATA 10, 3, -9, -15, -23, -25, -30, 1E+38
```

```
300 END
RUN
N=7
TOTAL= -89
MEAN= -12.7143
```

可以想象，这个程序写好后，即使搁置若干年以后再来读它，也一目了然。如果你写好这样一个程序，让另一个人来读，无需作任何交代，人家也会很容易读懂它。

较为复杂的程序，最好在每一个自然程序段之前加上注释语句。

习　题

(一)　给出下列各表达式的逻辑值：

(1) 5>3 AND 6>4

(2) 3+5>6 OR 7=6

(3) A>A AND 4↑2>2

(4) 5 OR 3<3

(5) 5=6=7=0

(二)、指出下列语句中的错误：

(1) 10 IF X≠0 THEN X=X+1

(2) 10 IF A≥3 THEN PRINT A

(3) 10 IF B>A THEN A=B

(4) 10 IF A$=ABCD THEN PRINT ABCD

(5) 10 IF A=A THEN 10

(三)　指出下列程序的运行结果：

(1)

```
10 K=1
20 PRINT K
30 IF K<5 THEN K=K+1 : GOTO 30
```

```
40 PRINT K
50 END
```

(2)

```
10 READ X, Y
20 IF X= -1 THEN 70
30 IF X*Y<0 THEN 50
40 PRINT X, Y
50 GOTO 10
60 DATA 1, 2, -3, 4, 4, 6, 9, -8, -1, -1
70 END
```

（四） 编写程序，由键盘输入弧度X值，按下列公式求Y：

$$Y=\begin{cases}\dfrac{\sin(X)+\cos(X)}{2} & X\geqslant 0\\ \dfrac{\sin(X)-\cos(X)}{2} & X<0\end{cases}$$

（五） 一个三角形的三个边长给出后，可通过下列公式求出三角形的面积：

$$S=\sqrt{P(P-A)(P-B)(P-C)}$$

其中

$$P=\frac{A+B+C}{2}$$

试编写此程序。边长A，B，C的值可用语句INPUT给出。要求程序能检查出：当给出的三个边长构不成三角形的情况，并提示数据错，再由用户重新给出三个边长。

（六） 某学校规定：期终考三门课，其中两门为主课。凡考分符合下列条件之一的学生，发给“优良成绩奖”：

1.三门课总分在280以上者；

2.两门主课成绩均在95分以上，其余一门课考分在80分以上者；

3.有一门主课为100分，另两门合计不低于160分者。

试编写一个程序，对得奖者能打印出学号与各门课的考分。对无奖者只打印出学号来。（学号、考分自选）

（七） 试写一个程序，把下列数据中的每个负数都打印出来，并打印所有正数之和：

－2，3，15，－86，－7.3，90，－80，7，42，－8

（八） 写一个程序，用来产生一批从0到20的整随机数，当得到的随机正好为15时，结束程序。

（九） 某人写了如下一段程序，其意图是：从键盘输入一个数，只要这个数不小于5，就弃之不用，重新从键盘上输入新数。仍做上面的比较，直到找出一个不满足那个条件的数为止，打印这个数，停止程序运行。但他在运行这个程序时，曾敲入成百上千个数（确实敲了不少理应打印出来的数），但始终没打印出一个数来。为什么？

```
10 INPUT "X = "; X
20 IF NOT X<5 THEN 10
30 PRINT X
40 END
```

第五章　程序的循环

本章将介绍循环程序设计的概念。循环语句是专用于这种程序设计的，由于它有着固定的格式和执行方法，使用起来十分方便。循环语句涉及的保留字有FOR、TO、STEP、NEXT。通常把它们缩写成FOR…NEXT。本章除了介绍它们以外，还有 TRACE和NOTRACE。

§1．循环语句的基本概念

一、循环语句的引入

有前面所学知识，编写一个计算$\sum_{n=1}^{10} n$（即求1＋2＋3＋4＋…＋10）的程序是不难的。

让我们设变量N为被加的那一项数据，设变量S为累加和。程序如下：

```
10 S=0 : N=1
20 S=S+N : N=N+1
30 IF N<=10 THEN 20
40 PRINT S
50 END
```

程序中的20行、30行，实际上要反复执行10次。这种反复，通常也叫循环。

上列程序的循环，是由30行的IF语句控制实现的。

如果把上面的程序，改用循环语句来写，将成为下面的样子：

```
10 S=0
20 FOR N=1 TO 10 STEP 1
30 S=S+N
40 NEXT N
50 PRINT S
60 END
```

以上两个程序具有同等效果。

二、循环语句的格式

循环语句的格式是：

〔行号〕 FOR 实型变量名 = 算术表达式1
TO 算术表达式2〔STEP 算术表达式3〕
⋮ } (循环体内各语句)
〔行号〕 NEXT〔实型变量名〕

格式中的实型变量名，称为循环变量或控制变量；

算术表达式1，称为初值；

算术表达式2，称为终值；

算术表达式3，称为增量或步长。

NEXT后跟的实型变量名要同FOR后跟的实型变量 名一致。

三、循环语句的执行过程

循环语句的执行过程，在不同的BASIC版本中有所区 别。苹果BASIC的执行过程如下(参看图5—1)：

1.循环变量取初值，并登记终值和步长；

2.执行循环体内各语句；

3.遇NEXT语句时，参照对应的FOR语句，将循环变量 的值加上步长再赋值给循环变量；

4.检查循环变量的新值是否已越过终值。如未越过，则返回到第 2 步继续做。否则，执行NEXT 的后续语句。这时循环过程也就结束了。

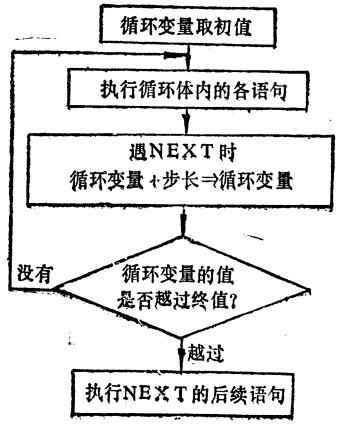


图 5—1

初学循环语句时，可以把循环语句运行时的情况列出一个表来。象前面那个求$\sum_{n=1}^{10} n$的程序，运行时情况如下表所示：

表5—1

循环次数	N的值	N越过终值了吗	S的值
未进入循环前			0
1	1	不比较	1
2	2	否	3
3	3	否	6
⋮	⋮	⋮	⋮
9	9	否	45
10	10	否	55
11	11	越过，结束循环	55

这类表，可以帮助你清晰地了解各有关变量在循环过程中的

改变情况。

在某些BASIC版本中，循环语句执行过程与上面介绍的略有不同(参看图5—2)：

1．循环变量取初值，登记终值和步长；

2．检查循环变量的当前值是否越过终值。如已越过，则退出循环，执行NEXT的后续语句，否则，做下面的工作；

3．执行循环体内的各语句；

4．遇NEXT语句时，参照对应的FOR语句，将循环变量的值加上步长再赋值给循环变量，返回第2步进行比较。

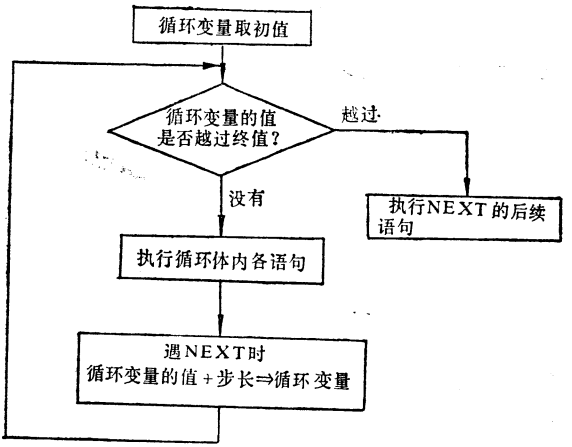


图 5—2

四、各种循环情况的进一步说明

1．当增量（即步长）为正1时，STEP 1可予省缺。我们在格式说明中把STEP算术表达3作为可选项，就是指的这个意思。

2．增量可以是负（即负步长）。

我们还举那个求$\sum_{n=1}^{10} n$的例子。显然用

$10+9+8+\cdots+2+1$

结果是相同的。在这里各项的值是从10开始，逐个减1。用负1作步长，程序可改写如下：

```
10 S=0
20 FOR N=10 TO 1 STEP -1
30 S=S+N
40 NEXT N
50 PRINT S
60 END
```

3. “越过”终值的含义。

上面我们用循环语句写出了两个求$\sum_{n=1}^{10} n$的程序。它们分别用了不同的FOR语句：

```
FOR N=1 TO 10 STEP 1
FOR N=10 TO 1 STEP -1
```

这两个语句的区别仅仅在于：循环变量N的值是从1逐次取到10呢，还是从10逐次取到1。

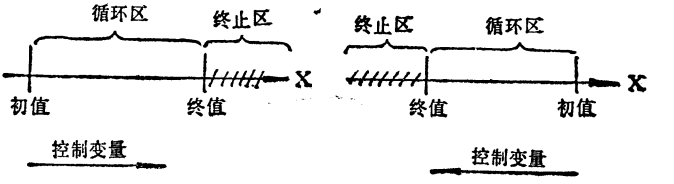
如果我们把1，2，3，…，9，10画在X坐标轴上，那么在逐次执行第一个FOR语句时，N的取值是从1开始逐次加1的，也就是在x轴上逐个向右的。当N取10时，正好在终值的位置上，不算越过终值，下次N取得11时，它比终值大了，也就是N从终值的左侧“越过”终值而达于终值的右侧。这时就退出循环。

类似地，逐次执行第二个FOR语句时，步长为负，N的取值从右向左。当它从终值右侧“越过”终值而达于终值之左侧时，结束循环。

此种情况，可由图5—3形象地示出。

从这里我们可以引伸出如下一个事实：一个循环经由NEXT结束以后，循环变量的值正好是越过终值后的第一个值(或者说，

是循环区内最后一个值加步长)。



(a) 步长为正　　(b) 步长为负

图 5—3

因此上述求$\sum_{n=1}^{10} n$的程序，加上如下一个程序行：

```
45 PRINT N
```

它刚好放在NEXT N之后。对于步长为1的情况，它打印出11；对步长为负1的情况，它打印出0。

我们知道，任何变量的当前值，总是要登记入变量表中的。而循环变量也是变量，因此，它的每次取值，都要记入变量表(注意，一个变量取得新值时，旧值被冲掉)。当循环变量取得越过终值的第一个值时，尽管它导致了循环的结束，但在变量表中却被保留了下来。

4. 初值已越过终值的情况。

考察下列程序：

```
10 FOR I=2 TO 1 STEP 1
20 PRINT I
30 NEXT I
40 END
```

这个循环结构中，步长为正，而初值为2，终值为1，初值已越过终值。

不要以为在这种情况下循环体内的PRINT语句执行不到。按

苹果机上循环语句的执行过程，20行的语句至少执行一次。因为检查循环变量取值是否越过终值，是在NEXT返回后才进行的。按图5—1的过程试一下，这点不难理解。

5. 循环语句中的初值、终值和增量，分别为三个算术表达式，并不限定它们只能取常量，也不限定它们只能取整数。

因此，下面三个程序结果完全一样。

程序1：

```
10 FOR I=0 TO 360
20 PRINT SIN (I*3.1416/180)
30 NEXT I
40 END
```

程序2：

```
10 FOR I=0 TO 2*3.1416 STEP 3.1416/180
20 PRINT SIN (I)
30 NEXT I
40 END
```

程序3：

```
10 A=3.1416
20 FOR I=0 TO 2*A STEP A/180
30 PRINT SIN (I)
40 NEXT I
50 END
```

它们都会把0°到360°每度的正弦值打印出来。

6. 循环变量可以出现在循环体内，也可以不出现在循环体内。

循环变量出现在循环体内，又有两种情况：

(1) 在循环体内，循环变量的值不被改变。

上面所举各例，循环变量都在循环体内出现，但仅取其值来使用，并未改变它们的取值。这是使用循环语句编写程序时的主

要用法。

(2) 在循环体内，循环变量的值被改变。

请看下例：

```
10 FOR I=1 TO 5
20 I=I+1
30 PRINT I
40 NEXT I
50 END
PUN
2
4
6
```

执行的结果是打印2，4，6三个数。而1，3，5（虽然它们曾赋给变量I）却没打印出来。循环次数也减少了（循环体内的语句只执行三次）。这些都是由于在循环体内改变了循环变量的值的缘故。

当然，这是一个不好的程序。若是确实想通过循环程序设计打印2，4，6三个数，使用下面的程序会更好些：

```
10 FOR I=2 TO 6 STEP 2
20 PRINT I
30 NEXT I
40 END
```

在循环体内改变循环变量的值，往往会使循环变量失去控制，甚至使程序陷入“死循环”。

如程序：

```
10 FOR I=1 TO 2
20 I=I-1
30 PRINT I
40 NEXT I
```

```
50 END
```

这个程序运行时，会无休止地循环下去，每循环一次打印一个0。这就是由于20行中改变循环变量的取值造成的。

所以，在一般情况下（特别是对初学者来说），不要轻易使用在循环体内改变循环变量取值的方法。

至于循环变量不出现在循环体内的情况，循环变量仅用来控制循环的次数。

用这种方法改写求$\sum_{n=1}^{10} n$的程序，可以是：

```
10 A=0 : S=0
20 FOR I=1 TO 10
30 A=A+1
40 S=S+A
50 NEXT I
60 PRINT S
70 END
```

在这个程序中，循环变量为I，它在循环体内不曾出现。循环体内的A，表示每次要加入累加和S的数。循环变量I仅用来保证循环体内的语句执行10次，从而实现$\sum_{n=1}^{10} n$的目的。

7. 对于循环体内的各语句（合称为循环体），没有给出什么限制，原则上所有语句都可以使用，甚至可以没有语句。

这里我们着重说明以下三种情况：

(1) 循环体内没有任何语句。

例如程序段：

```
100 FOR I=1 TO 1000
110 NEXT I
```

或将它们合写成一个程序行：

```
100 FOR I=1 TO 1000 : NEXT I
```

这种循环，通俗地称为空循环。虽然它没有做任何事，却一样要循环1000次，这是要花费时间的。因此它实际上起到了延时作用。这在某些程序中是需要的。它可以使屏幕的显示滞留一段时间，以便让用户看得更清楚些。延时长短，和循环次数成正比。

(2) 循环体内的IF语句

循环体内允许用IF语句进行条件转移。

下列程序，可以把100(含)以内的所有能被7整除或能被5整除的正整数，一一打印出来：

```
10 FOR N=1 TO 100
20 IF INT (N/5) <>N/5 AND TNT (N/7) <> N/7
  THEN 40
30 PRINT N
40 NEXT N
50 END
```

它是通过IF语句来控制实现的。

上述程序中的20行和30行合并成一个程序行，如：

```
20 IF INT(N/5)=N/5 OR INT(N/7)=N/7 THEN PRINT
   N
```

效果完全一样。

但是如果有人把原来的第20行程序改写成：

```
20 IF INT (N/5) <> N/5 AND INT (N/7) <> N/7
  THEN 10
```

则是错误的，是程序设计不许可的。请看：第一次进入循环体时，N的取值为1，它既不能为5整除，也不能为7整除，因而转回第10行。这是第二次进入该循环，这时N又从1开始取值，经20行三次进入该循环。这样可以没完了运行下去，永远得不到结果。这也是一类死循环。如果你在15行上加上一句：

```
15  PRINT N
```

程序会没完没了地打印出1。

上述条件转移，实际上是转出了循环体。也就是从循环体内转到了循环体外。应该说，从循环体内转到循环体外是允许的，但是，从循环体内未经NEXT直接转向本循环语句的入口，必然导致程序的逻辑错误（不是语法错误），造成死循环。

合理地从循环体内转出循环体外的情况，可见下例。

比如，希望从自然数中找出13，27，33的最小公倍数来，程序可以写成：

```
10  FOR  I=33  TO  13*27*33
20  IF  INT (I/13)=I/13  AND  INT (I/27)=I/27  AND
    INT (I/33)=I/33  THEN  40
30  NEXT  I
40  PRINT  I
50  END
```

这个程序中的初值取33，是因为所求的最小公倍数不会小于33；终值取13＊27＊33，是因为所求的最小公倍数也不会大于这个积。这个积虽然是13，27，33的公倍数，但不一定是最小的公倍数。这个程序通过IF语句，可以找出最小公倍数来。只要这个结果求得，就没有必要再循环下去了，于是转出循环体外。

（3）循环语句内的循环语句

循环语句内套循环语句，称为循环语句的嵌套，或称之为多重循环。关于这一部分的内容，我们将在第3节里专门叙述。

§2．循环语句应用举例

例1．打印乘法表的$1\times1=1$，$1\times2=2$，$1\times3=3$，…，$1\times9=9$

程序如下：

```
10  A=1
20  FOR  B=1  TO  9
30  PRINT  A; "*"; B; "="; A*B
40  NEXT  B
50  END
RUN
1*1=1
1*2=2
⋮
1*9=9
```

如果把第10行的A=1分别改写成A=2，A=3，…A=9，并分别运行这个程序，可以打印出整个乘法九九表来。

例2．数列

0，1，1，2，3，5，8，…

称为波契纳级数。它的头两个数分别为0和1，从第三个数开始，每一个数为前两个相邻数之和。

编写一个程序，以便求出这个数列，直到第100个数或所求的数达到（或超过10^{10}）为止。

程序如下：

```
10  A=0 : B=1
20  PRINT A, B,
30  FOR I=3 TO 100
40  C=A+B
50  IF C>1E 10 THEN 90
60  PRINT C,
70  A=B : B=C
80  NEXT I
90  END
RUN
0                1                1
```

2	3	5
8	13	21
34	55	89
144	233	377
610	987	1597
2584	4181	6765
10946	17711	28657
46368	75025	121393
196418	317811	514229
832040	1346269	2178309
3524578	5702887	9227465
14930352	24157817	
39088169	63245986	
102334155	165580141	
701408733	1.13490317E+09	
1.8363119E+09	2.97121507E+09	
4.80752698E+09	7.77874205E+09	

上述程序运行后输出的格式中，后六行只有两个数，第三列数的位置没有数。这是因为在那一区（第三区）输出的数已达到8个或8个以上字符位置，故跳过这一区自动下移到下行的左端。另外，最后五个数因超过规定的九位有效数字，而自动采用了科学计数法。

例3．相传古代印度有个宰相叫达依尔，是国际象棋的发明者，非常聪明能干。国王舍罕准备奖励他，问他想要些什么。达依尔说："国王只要在国际象棋的棋盘上的第一个格子里放上一粒麦子，第二个格子里放两粒麦子，第三个格子里放四粒麦子，以后每个格子加一倍，一直放完第64格，我就感恩不尽了。"国王想："64格有多少，这还不容易！"于是令人扛来一袋麦子，结果很快就用完了。又扛来一袋，很快又用完了。国王又命令扛来了几十袋麦

子，结果按达依尔的要求还差得很远。国王非常奇怪，怎么需要这么多？

那么，到底需要多少麦子呢？这个问题可以通过下列程序来求出。

64个格子中应放的麦子的粒数T，可用下面的公式算出：

$$T = 2^0 + 2^1 + 2^2 + 2^3 + \cdots + 2^{63}$$

假定1立方米的麦子约有1.42×10^8粒。

那么，T粒麦子的体积V应为

$V = T/(1.42 \times 10^8)$立方米。

这样，解此题的程序为：

```
10  T=0
20  FOR N=0 TO 63
30  T=T+2↑N
40  NEXT N
50  PRINT "T=";T
60  PRINT "V=";T/(1.42*10↑8)
70  END
RUN
T=1.84467441E+19
V=1.29906648E+11
```

别小看这个数字。如果把这么多麦子铺撒在我国960万平方公里的土地上，足有1.3厘米厚。它相当于我国几百年的产量。

例4． 多项式求值问题。

一般，求多项式

$$a_0x^n + a_1x^{n-1} + \cdots + a_{n-1}x + a_n$$

的值，只要赋予各个系数和x的值，是不难求出的。可以用一个赋值语句，赋值号的右部直接用上面的多项式（当然，要把a_0，a_1等改写成合法的变量名，把＊号加到式中相应处，使上面的式

子变成合法的BASIC表达式）就行了。

然而这样的程序上机运行时，求每一项，都要求x的连乘积，重复计算太多，因而执行时间较长。

但我们可以把上述多项式作如下的变形：

$$\begin{aligned}y&=a_0x^n+a_1x^{n-1}+a_2x^{n-2}+\cdots+a_{n-2}x^2+a_{n-1}x+a_n\\&=(a_0x^{n-1}+a_1x^{n-2}+\cdots+a_{n-2}x+a_{n-1})x+a_n\\&=((a_0x^{n-2}+a_1x^{n-1}+\cdots+a_{n-3}x+a_{n-2})x+a_{n-1})x+a_n\\&=\cdots\cdots\\&=(\cdots((a_0x+a_1)x+a_2)x+\cdots+a_{n-1})x+a_n\end{aligned}$$

最内层的括号中为a_0x+a_1

设$b_1=a_0x+a_1$，则次括内为b_1x+a_2

再设$b_2=b_1x+a_2$，则再外一个括号内为b_2x+a_3

…

最后，令$b_{n-1}=b_{n-2}x+a_{n-1}$则最后$y=b_{n-1}x+a_n$

可见，计算y的过程，归结为逐次计算多项式：

$$b_{i-1}x+a_i$$

的过程了。

比如要求计算

$$y=2.5x^5+4.1x^4+7.8x^3-4.45x^2+3x-6.8$$

可改写成

$$y=((((2.5x+4.1)x+7.8)x-4.45)x+3)x-6.8$$

这时可以把多项式的最高次数和各系数的值写入DATA语句中，而X的值由键盘输入，写成下面的程序：

```
10  INPUT "X=?";X
20  READ N
30  FOR I=0 TO N
40  READ A
50  Y=Y*X+A
```

```
60  NEXT I
70  PRINT "Y=";Y
80  DATA 5, 2.5, 4.1, 7.8, -4.45, 3, -6.8
90  END
RUN
X=?  0.6
Y= -4.19144
RUN (再次运行)
X=?  2.8
Y= 820.20576
```

可见，每次运行这个程序，只要给出X的值，就能求出相应的多项式的值来。

把这个程序写得更一般化些，可以把X的最高次数（用变量N），X的值及各系数的值都用键盘敲入。这样程序就能适应各种多项式的求值问题：

```
10  INPUT "X=?";X
20  INPUT "N=?";N
30  FOR I=0 TO N
40  PRINT "A";I;"=";
50  INPUT A
60  Y=Y*X+A
70  NEXT I
80  PRINT "Y=";Y
90  END
```

§3. 多重循环

第一节我们提到过，在循环体内再用循环语句，构成了循环语句的嵌套使用。大循环内套一个小循环，小循环内还可以再套

一个小小循环。一个套一个，所套的次数，叫做循环嵌套深度。苹果机上允许的循环嵌套深度最大为10。

多重循环在解决应用问题中，是非常有用的。

比如上节例1，在那里我们说过，要求出所有九九表，得把那个程序的第10行改写9次，而且还得分别运行那个程序9次。如果改用双重循环，那就方便多了：

```
10  FOR A=1 TO 9
20  FOR B=1 TO 9
30  PRINT A;"*";B;"=";A*B;"  ";
40  NEXT B
50  PRINT
60  NEXT A
70  END
```

这个程序在输出格式上作了一些考虑。第30行的打印语句，在打印出数据之后，空出一些空位来，以便使下次输出能与之保留距离。第50行的PRINT 语句，使以后的输出能“另起一行”。可惜每行只能40个字符位置，每执行50行之前，已输出的数据已被自动输出在两行中了（中间有个到达右端后的自动换行）。

双重循环语句的执行过程，沿用单循环的执行原则。程序总是先遇最外层那个循环。这时你可把内层循环当作一个语句来看待。最外层循环执行过程，沿用前面的方法，而遇到内层循环时，我们既把它当作一个语句，也一样要执行这个语句。然而它毕竟是个循环语句，所谓执行它，就意味着从FOR 进入而从NEXT退出。这一过程当然还是一般循环语句的执行方法。因此，就刚刚列出的打印九九表的程序而言，每对A循环一次，就要对B完整地循环9次。总共要对B循环9＊9＝81次。

三重、四重等循环，执行过程以此类推。

对于嵌套起来的循环，要求大的要套住小的。也就是说，循

环语句不得交叉使用。如下图（5—4）的说明。

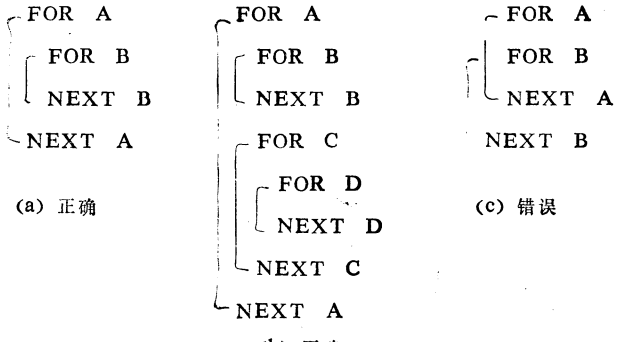


图 5—4

下面列举几例，说明多重循环的程序设计方法。

例 1．某学习小组的5名同学参加高考，6门课的考分分别如下：

第 1 位：95，99，93，89，92，90

第 2 位：78，82，83，79，89，85

第 3 位：97，92，89，87，91，88

第 4 位：88，90，79，85，90，84

第 5 位：65，73，80，78，72，75

编写一个程序，把总分在530分以上的同学找出来（按1,2，3，4，5编上序号），并打印出其总分。

程序可用双重循环，外层循环对着每个人，内层循环对着各门考分。

程序如下：

```
10  FOR I=1 TO 5
20  READ K
30  S=0
40  FOR J=1 TO 6
```

```
50  READ A
60  S=S+A
70  NEXT J
80  IF S>=530 THEN PRINT K, "S="; S
90  NEXT I
100 DATA 1, 95, 99, 93, 89, 92, 90
110 DATA 2, 78, 82, 83, 79, 89, 85
120 DATA 3, 97, 92, 89, 87, 91, 88
130 DATA 4, 88, 90, 79, 85, 90, 84
140 DATA 5, 65, 73, 80, 78, 72, 75
150 END
RUN
1                S=558
3                S=544
```

如果将上述程序中的变量K改为K$，而将5个DATA语句中的第1个数据（序号）改用学生姓名的字符串常量，那么在输出的结果中，最左边一栏为相应考生的姓名。

例2．求3到100之间的所有素数。

我们知道，所谓素数，是指除能被1和自身整除外，不能为任何整数整除的数。这样，我们对3到100的每个数进行这样的检查：看这个数能否被从2到比它小1的任何整数所整除。若能整除，此数非素数，否则它就是一个素数。这是最容易理解的一种算法。

用双重循环很容易设计出这个程序：用外层循环控制对3到100的每个数的检查，用内层循环来具体检查这个数可否被2到比它小1的数整除。

程序如下：

```
10  FOR I=3 TO 100
20  FOR J=2 TO I-1
30  IF INT(I/J)=I/J THEN 60
```

```
40  NEXT J
50  PRINT I,
60  NEXT I
70  END
```

程序中的30行就是判断所取的I是否能为J整除。如能整除，转到60行，从而返回再取下一个I值。如不能被任何一个J整除，则I即为素数，在运行50行时打印出来。

敲入RUN 命令，输出如下结果：

```
3           5           7
11          13          17
19          23          29
31          37          41
43          47          53
59          61          67
71          73          79
83          89          97
```

例 3．某机关对所属15个基层单位进行卫生大抽查，对每个单位抽查 8 个办公室，每个办公室检查10个项目，每项满分为100分，50分（不含）以下为不合格。最后以积分多少评出最好的和最差的单位来。但规定，任何一个单位的任一办公室若有两项不合格者，或者一个单位累计有 6 个项目不合格，该单位无资格被评为卫生最好单位。

我们同样给每个单位规定一个号码。用DATA 语句来记载这些分数。把每个单位的得分排在一起，其中第一个数据为单位号码，其后的数据为所得的分数。这些分数又以一个办公室为一组，排在一起。换言之，每个单位有81个数据放在连续的DATA语句中，第一个数据是该单位的代号，接下去是该单位某一办公室所得的10个分数，再下去是另一办公室所得的10个分数，…。

程序可以这样写：

```
10  U=0 : V=8000
20  FOR I=1 TO 15
30  READ L
40  S=0 : N=0 : T=0
50  FOR J=1 TO 8
60  M=0
70  FOR K=1 TO 10
80  READ A
90  IF A<50 THEN M=M+1 : N=N+1
100 IF M>=2 OR N>=6 THEN T=1
110 S=S+A
120 NEXT K
130 NEXT J
140 IF T=1 THEN 160
150 IF U<S THEN U=S : X=L
160 IF V>S THEN V=S : Y=L
170 NEXT I
180 IF U=0 THEN PRINT "NO" : GOTO 200
190 PRINT X, U
200 PRINT Y, V
210 DATA ···
500 END
```

这个程序用了三重循环：第一层（外层）循环用来控制对15个单位的分别处理；第二层（中层）循环用来控制对某单位的8个办公室所得分数的处理；第三层（内层）循环用来控制任一办公室所得的10个分数的处理。层次是很分明的。

变量M，用以记录任一办公室不合格项目的个数；变量N用以记录任一单位不合格项目的个数。所以在开始处理一个单位的数据时，先使N取0值（40行），在开始处理一个办公室所得分数时，也使M取0值（60行）。当发现一个不合格分数时，由90行进行统

计。当发现某单位有一个办公室得了两个以上的不合格分或该单位累计有六个不合格分时，通过100行，设置一个 T = 1 的标志（当开始对某一单位的得分进行处理时，通过 40行使T的初值取0）。

变量S用来累加每个单位积分。所以在40行上总是使它置为0。

变量L 是被处理单位的代号，它总是在开始处理某单位得分时由30行的READ读入。变量V用来记录当前积分最高的那个单位的分数的，变量X则为相应的单位代号（注意，若是某个单位已得到 T = 1 的标记，即使它积分最高，也不会用V表示，因为它被取消夺魁资格——这一工作，由 140行控制实现）；而变量V用来记录当前积分的下限，变量Y则是相应单位的代号。

至于程序第10行对U和V所赋的初值，是为了保证只要U和V取得第一个值时，都能改变它（因为任何一个单位的积分都不会是0，总是大于 0 的；任何一个单位的积分也不会大于8000。因而首次执行150行时，将会使U取当时的S 值。而首次执行160行时，同样会使V取当时的S 值），使之分别作为单位最高积分和单位最低积分的真正初值（这里用真正二字，意思是指这种积分是从某个单位得分中累加出来可用的真实分数，而不是虚设的0或8000）。

程序180行是为了处理这样一种极为特殊的情况：可能所有15个单位都得到T = 1的标记，从而找不出优胜者。

程序中，我们略去了DATA语句的全部数据。这对读程序是没有影响的。但要记住DATA中数据的结构方式。

这个程序还是有一定缺点的。比如，若是有并列第一名或并列的末名时，这里分别只取一个；另外，程序也没有把中间的名次排出来。

§4. 循环语句的深入讨论

因为循环语句具有一定的语句格式和一定的执行过程，所以

一般使用起来十分方便。然而循语句毕竟要比我们前面学过的语句复杂些，特别是在循环体内允许使用转移、循环等语句，这样，复杂程度更趋严重。在使用循环语句时，稍不小心，将会导致意想不到的后果。这一节，我们打算对循环语句的执行细节作进一步的讨论，并从中引出一些使用循环语句时值得注意的问题。另外，这一节还将给出检查死循环的一般方法。

一、循环语句的执行细节

一个BASIC程序运行时，对于循环语句的处理，是通过循环栈的辅助作用来实现的。

所谓"栈"，这是计算机软件中的一个非常重要的概念。它是一种先进后出(或称后进先出)的结构。好象一个货栈一样，最先放入的货物，总是放在最底下，再放入货物时，往上堆；而往出取时，总是把最上层的货物先取走(后进先出)，最后才能取出最底下的货物(先进后出)。

计算机中的栈，是存放数据的，它总是开辟在内存中一个固定地址开始的一片连续单元中。这个固定地址，称为栈底。另外有一个专门的指针(称为栈指针)，在程序未运行前指向这个栈底。当一个数据需要存入栈中（称为进栈）时，就存入栈指针所指向的地址中，然后栈指针上移一个数据位置。如果再有数据要进栈，作类似的处理。反过来，如果要从栈中取出一个数据(称为退栈、出栈)时，栈指针首先下移一个数据位置，把所指向的那个数据(实际上是刚才最后一次存入的数据)取出供使用(注意，虽然叫退栈，但这个数据并未从刚才存放的内存位置上消失，只是它已处于栈指针之外，成为无效数据罢了)。可见栈指针是浮动的。进栈、退栈总是通过栈指针找到相应位置的，而栈指针所指向的位置又是通过进栈和退栈改变的。任何时刻栈指针所指向的位置，通常称为栈顶或栈的当前位置。

所谓循环栈，是专用于处理循环语句的栈结构。栈中的数据

是成组存放的，它们有循环变量名、FOR语句所在的位置等一组必要的信息(数据)。

现在我们可以从栈的角度进一步阐明循环语句执行时的细节了。

在程序运行中发现FOR语句时，首先要从循环栈的栈顶到栈底逐组检查一遍(称为遍历)，看有无当前待处理的FOR保留字所用的循环变量名，若是没有(在程序运行中遇到第一个 FOR 语句时，肯定是这样。因为这时栈顶和栈底重合，栈中尚无数据)，则使待处理的FOR的有关数据进栈。然后循环变量取初值，执行循环体内的各语句。如果在循环体内又出现了FOR语句，重复上述过程：遍历循环栈，查找有无待处理的同名循环变量。若还是没有，第二个FOR的有关数据进栈(这时栈中有两组数据了)。同样地，内层的(即第二个)循环变量取初值，执行内循环体内的各语句。如果这个内循环体中又出现了FOR语句，不言而喻，照上面的办法去处理。问题是，如果遍历循环栈时，发现栈中已有待处理的同名循环变量名，该怎么办？苹果机提供的处理方法是：将循环栈中的这一组有关数据，由当前待处理的FOR的有关数据取代，并以此数据作为当前栈顶位置(即把指针移到这组数据的上方)。当然这样做的结果，会使栈中的一些数据失去效力（由于栈指针的下移，它们被排到栈外了。这也相当于一些数据退栈)。

通过上面的叙述，可以看到，对于循环栈而言，FOR的作用，是在循环栈中登录一些有关数据备查。对于多重循环语句，第一重(最外层)循环的有关数据总是在栈底；最后一重(最内层)循环的有关数据总是在栈顶。

下面，我们再来叙述一下NEXT的处理过程。

假定我们处理的是一个单循环，这时循环栈中当然也只有一组数据，这组数据正处在栈指针的当前位置上。程序运行遇到保留字NEXT 时，它首先查看NEXT之后是否跟着一个变量名。如

果没有，则将栈指针所指向的那组数据供NEXT使用。因为那组数据中有着相应FOR的有关信息，从而不难找到这个循环的终值、增量、以及循环体的入口处。这时循环变量加上增量再赋给循环变量，并检查它是否越过了终值。如果它没有越过终值，则重新执行循环体内的语句，重复上面的执行过程，直到循环变量越过终值时，退出循环。这是在保留字NEXT之后没有跟变量名时的情况。如果NEXT之后跟有变量名，则遍历循环栈，看栈中有无同名循环变量名。若是没有，则给出出错提示(?NEXT WITHOUT FOR ERROR…，意思是NEXT没有对应的FOR)。若是栈中的数据里有NEXT后跟的变量名，则将栈指针指向这组数据，并提供NEXT使用。以下的执行过程同前面叙述的一样。

那么退出循环又做些什么事呢？很简单，那就是将栈针下移一组数据的位置(对于单循环而言，这时栈指针又返回栈底，回复到初始状态了)。这也就是使刚刚用过的那组数据退栈了。

知道了单循环的NEXT处理过程，对于多重循环的NEXT处理过程，也就迎刃而解了。这里仅就双循环的情况稍作说明。

执行BASIC程序时，最先要处理的NEXT，一般是属于最内层的那个循环的。对于它的处理，可以按上面处理单循环的方法去做，直到内层循环结束，相应的数据退栈，外层循环的有关数据处于循环栈顶。如果程序运行中又遇到了NEXT，一般它属于外层循环的，也可按单重循环的办法处理，而当它不满足退出条件时，总要返回外循环体的头上继续执行。这样将会再次遇到内循环的FOR，这个FOR的有关数据，也就再次进入循环栈中。往下运行时如果又遇到内循环的(而不是外循环的)NEXT，则内循环再次被执行，直到退出。内循环退出后，再遇到的NEXT便属外循环的了，再重复上面的处理过程，直到退出外循环。可见，每遇到一个外循环的NEXT，只要返回其循环体，内循环的有关数据将进栈一次；而每退出一次内循环，这组数据也就退栈一次。

从而通过这种进栈、退栈，实现了对循环层次的控制。

上面叙述的循环语句的执行方法，由图5—5示出。可参照一阅。

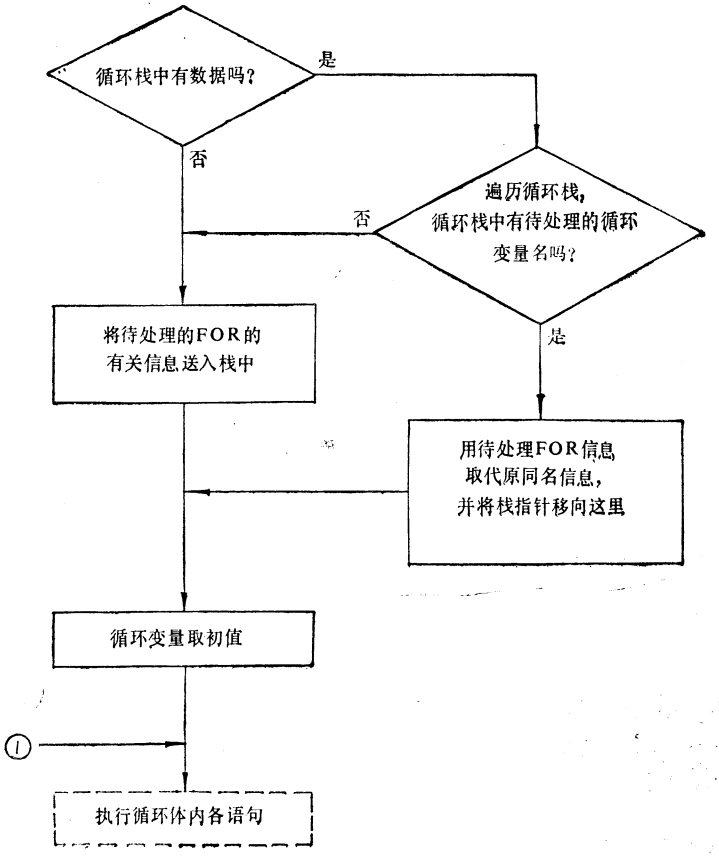


图5—5(a)　FOR过程示意图

二、使用循环语句应注意的问题

对于任何 FOR、NEXT，都可以按上面方法去处理。然而在

程序中的非正常的FOR、NEXT 使用，会带来一些特殊的问题。

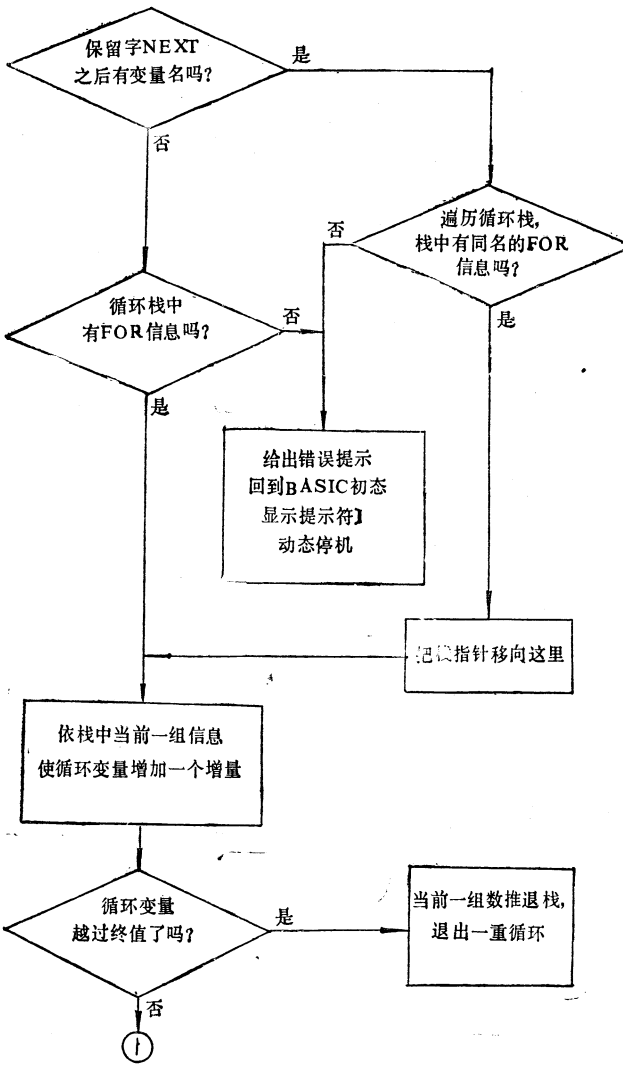


图5—5(b)　NEXT过程示意图

由于循环语句不是以单语句出现的，而且循环体内又可以使用转移语句等，这就造成了FOR 和NEXT非正常出现的可能性。这也构成了使用循环语句时应注意的一类主要问题。

1.关于多重循环不得交叉的问题

这是使用多重循环语句时必须严格遵守的一条规则。它的道理，通过循环栈的概念，非常容易理解。

试考察如下程序：

```
10  FOR I=1 TO 5
20  FOR J=10 TO 15
30  PRINT I, J
40  NEXT I
50  NEXT J
```

这是一个非常简单而又非常典型的循环交叉程序段。程序运行时是怎样处理的呢？

执行10行和20行时，会在循环栈中进入两组数据：有关I的一组被压在栈底，有关J的一组放在栈顶(栈指针指向着它)。执行30行时，打印当时的I值和J值。执行40行，语句为NEXT I，这个NEXT语句，是执行上述循环结构程序中第一次遇到的NEXT，它本应当属于内循环的，但却又用了外循环变量。按照前述的方法，首先遍历循环栈，而且确实找到了有关FOR I的信息（因为第一个FOR I的有关数据在栈中），便让栈指针指向它，并提供这组数据为NEXT I使用。当I得到的新值不越过终值(5)时，再执行对于I的循环体（即外循环体），也就是重新执行20行，再次将FOR J的有关数据进栈（这次进栈的数据，实际上和上次FOR J进栈的数据一样，在栈中的位置也相同。只是由于对I的遍历作用，此前把它推出栈外罢了），经30行打印I,J值，再处理NEXT I，同样把FOR J信息推出栈外。直到I的值超过5，退出对I的循环，将指针复位到栈底。这时遇到50的NEXT J，遍历循环栈时已找

不到任何FOR信息，于是给出出错信息。

这样，这个程序的输出结果将是：

```
 1                10
 2                10
 3                10
 4                10
 5                10
? NEXT WITHOUT FOR ERROR IN 50
```

如果删去50行，运行这个程序时，除显示上列5行10个数外，正常结束。

可见这个程序中的20行，虽然每次也将其有关数据进栈，但由于没有相应的NEXT J 为它控制，这些数据不会被用到，这个FOR J根本就循环不起来。第20行程序除了为30的打印语提供了一个具体数(10)以外，在这个程序中没有别的用处。

通过这个例子，我们还可以引伸出另外一个规则：在多重循环中，在不同的深度上，不得选用相同的变量名。

比如程序：

```
10  FOR I=1 TO 3
20  FOR J=10 TO 13
30  FOR I=100 TO 102
40  PRINT I : PRINTJ : PRINTI
50  NEXT I
60  NEXT J
70  NEXT I
80  END
```

这个程序运行时会输出：

```
100
10
100
```

```
101
10
101
102
10
102
? NEXT WITHOUT FOR ERROR 60
```

这一输出并不奇怪。因为，程序执行完前两行后，循环栈中有两组循环信息：有关I的放在栈底，有关J的放在栈顶。运行30行时，对FOR I将从遍历循环栈中找到栈底那组数据，将其按30行的信息改写，并把栈指针指向它。这样循环栈中仅剩30行的循环信息。于是30～50行构成一个循环。执行到60行时，循环栈中已无信息，于是给出错误提示。

至于J的值，在执行20行时（只执行过一次）已把它的初值记入变量表中，所以输出中为其初值。

2.关于内外转移问题

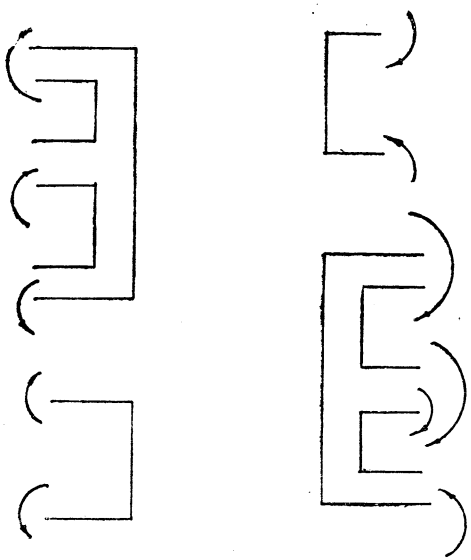
一般，对循环语句有一条规定：允许从循环体内转出循环体外，但不允许从循环体外转入循环体内。示意图如图5—6。

从循环体外转入循环体内，原则上是绝对不许可的。因为没经过FOR在循环栈中登记有关信息，NEXT就无法实现。

但在实践中有时会出现意外的情况。

试考察下面一个程序例：

```
10  FOR I=1 TO 5
20  IF I=3 THEN 60
30  PRINT I
40  NEXT I
50  END
60  PRINT "A"
70  GOTO 90
```



正确　　错误

图5—6

```
80  FOR I = 10 TO 15
90  PRINT I
100 NEXT I
110 END
```

这个程序运行结果是：

```
1
2
A
3
4
5
```

程序20行是个条件转移语句。这个条件总会出现的。这将出现一次从循环体内转向循环体外的情况。按规则，这是允许的。

而70行是一个无条件转移语句，它却转向一个循环体内了。按规则这是不允许的。还有，这个程序运行到后来，又转回前面一个循环体内了(最后输出的4和5，只能由那个循环实现)，这又是一次从循环体外转入循环体内的现象，也是不允许的。但是，程序运行完了，输出了一批结果，而没有任何出错信息。

为什么会出现这种情况？让我们还用循环栈的概念来说明它。

程序运行时先对前面那个循环执行两次，打印出1和2，执行第三次循环时由于I = 3，转到60 行打印出字符A来。这些都符合规则，好理解。但要知道，这时循环栈中只有 I 的一组数据，且栈指针正好指向它；而变量表中的 I 的值为 3 。执行70行时无条件地转到 90 行，打印I的值(就是变量表中的I，值为3)。接下去100 行是NEXT I，遍历循环栈，栈指针正好指向I的一组数据(尽管它不是第二个FOR登记的，但它却同名)，于是提供给 I，使程序转回到20行去，继续完成(正常的)第一个循环的后续工作，由于I不会再取值3，于是经30行把4和5打印出来，当I 变到6时，退出循环(栈指针复位)。程序由50行的END正常结束。

从这里我们看到这样一个事实：对于任何一个单循环来说，当它不是经由相应的NEXT 退出时，它的有关数据仍在栈中。只要遇到同名的NEXT 时，又返回该循环体来。正是由于相应的有关数据仍在栈中，实际上可以认为：未经NEXT 退出单循环体，仅是一个形式上的表面现象，而实质上仍在该循环体中。换言之，单循环变量不越过终值，是退不出单循环体的。

事实上，把上面的程序改写一下，可以是这样一个等效程序：

```
10  FOR I = 1 TO 5
20  IF I = 3 THEN 22
21  GOTO 30
22  PRINT "A"
25  PRINT I
```

```
27  GOTO 40
30  PRINT I
40  NEXT I
50  END
```

原来程序中的80行，110 行，根本没执行到过，所以在改写的程序中没有它们。

这个程序运行结果和原来程序运行结果一样。但它没有从循环体内转向循环体外的情况出现。

因此，对任一个单循环，只要还没有退出它，可以用任何转向语句，形式上转到循环体外来，在循环体外允许用任意语句写出一个程序段(其中还可以是循环，甚至多重循环)，然后再转回那个单循环中去。因为这时你可以把单循环体外的一切，当作仍在该循环体内那样对待。当然，如果在单循环体外遇到一个同名循环变量的 NEXT，又会返回原来的单循环体内继续循环下去(这一点，有时可以适当利用)。

对于多重循环，有时也会出现这种情况：从某个内层循环体内转出来（转至外层或者整个循环之外），作一段别的处理后，再转回那个内循环中。而这种情况也只能出现在这样的前提下：在转出以后所执行的那段（外部的）程序中，不得用NEXT处理从循环栈底到那个内循环的任何一个循环变量（即不能遇到这样的NEXT：在它的后面有任何一个从外循环到那个内循环的循环变量)。这样，从内循环转出来执行的所有程序，仍可认为是属于那个循环体的，因而允许转回去。

3．关于NEXT的省略写法

在苹果BASIC中，NEXT后跟的变量名，一般可以省去。如果程序中有两个连续的NEXT语句，可以合并在一起，即用一个NEXT，后跟两个变量名（它们之间加逗号分隔开)。这样做，可以减少运行时间。

如程序：

```
10  FOR I=1  TO 5
20  PRINT  I↑2
30  NEXT
40  FOR  I=1  TO  5
50  FOR  J=10  TO 15
60  PRINT  I*J
70  NEXT  J, I
80  END
```

其中 30行的NEXT相当于

```
30  NEXT  I
```

70行的NEXT J,I相当于

```
70  NEXT J
71  NEXT I
```

根据我们前面的叙述，NEXT后边如无变量名，总是取栈顶那组数据为NEXT使用的。一般情况不会出什么问题。但有时不注意，也会出现一些奇异现象。

比如程序一：

```
10  FOR  I=1  TO 5
20  FOR  J=10  TO 12
30  PRINT  I, J
40  IF  I=3  THEN  60
50  NEXT  J
60  NEXT  I
70  END
```

它的运行结果为：

```
1              10
1              11
1              12
2              10
```

```
2             11
2             12
3             10
4             10
4             11
4             12
5             10
5             11
5             12
```

而程序二：

```
10  FOR  I=1  TO  5
20  FOR  J=10  TO 12
30  PRINT  I, J
40  IF  I=3  THEN  60
50  NEXT  J
60  NEXT
70  END
```

这个程序运行结果是：

```
1             10
1             11
1             12
2             10
2             11
2             12
3             10
3             11
3             12
```

这两个程序的唯一区别是60行的NEXT后面带不带循环变量名。按理说，是可以省略的，但两者为何又输出不同结果呢？

其关键是在从40行转到60行时造成的。当着40行条件满足的那个时刻，输出为前7行，而循环栈中有着两组数据：有关I的一组在栈底，有关J的一组在栈顶。两个程序运行至此，情况完全一样。

对第一个程序而言，60行是NEXT I，这时遍历循环栈，同名(I)的是栈底上的第一组数，于是栈指针指向这组数据(关于J组信息排斥于栈外)，并用以为NEXT提供使用，使之转循环变量I的循环体，再执行20行，J的循环信息进栈，打印第8行输出。以后40行的条件转移不再出现，程序正常运行，打印出后续输出。直到结束。

而对第二个程序来说，从40行转60行时，遇到的NEXT不带变量名，因而它不遍历循环栈，而直接拿栈顶数据使用，这时栈顶为J的数据，从而使程序返回J的循环体（30行开始，可见它把60行当作NEXT J处理了），打印出第8行输出，这时I的值仍为3，又转60行，同样当NEXT J来处理，再返回30行，打印第9行输出。再由40行转60行，由于J的值越过12，退出J的循环，循环栈中I的有关信息处于栈顶。由于60行之后没有NEXT语句，I的循环信息不再被使用。执行至70行程序结束。

由于NEXT省略写法可能带来象上面第二个程序那样的副作用，通常在多重环循中，最好还是在每个NEXT之后加上相应的变量名。

4．关于THEN子句中的FOR

如果一个THEN子句中有FOR，那么这个子句中也必须有相应的NEXT。换言之，不可把FOR写入THEN子句中，而应把相应的NEXT写在另一个程序行中。因为当着条件不满足时，FOR执行不到，而后续程序的NEXT也就找不到对应的FOR（循环栈中没有THEN子句中的FOR信息）。

同样的道理，对于立即执行的循环语句，FOR和NEXT必须都敲入之后才能敲回车键。

三、跟踪命令的使用

在前面的介绍中，我们提到过“死循环”问题。然而有些死循环并不是一下子就能查出来的。有时可以通过机器来帮助查找。

用STOP命令固然也可以侦察死循环。比如在适当行号上放上一个STOP语句，当程序永远执行不到这个STOP语句时，可以认为死循环在这个STOP之前。但用STOP找死循环是较费力的。你可能要一次次改变STOP的位置，并一次次运行程序。你还不好确定执行多长时间算作“死”，…

用跟踪命令侦破死循环较有效。

跟踪命令与非跟踪命令的格式是：

〔行号〕 TRACE

〔行号〕 NOTRACE

当一个程序运行前，先敲入跟踪命令TRACE，然后再敲入RUN命令。随后在机器执行时，可以把一次次执行的行号，依次显示在屏幕上（也可打印出来）。这样你可以判别：当着所显示的行号总是在一个固定的范围内重复时，就有可能是一个死循环。这时再通过某些变量的检查，可以明确作出定论。

在使用跟踪命令时，程序执行时间加长了。所以不应动不动就使用这一命令。用NOTRACE命令可以撤消跟踪状态。

注意：使用跟踪命令时，输出的行号和PRINT语句输出的内容将参杂在一起。但所输出的行号都带有#号，所以还是可以区别开来的。

跟踪、非跟踪命令，属于苹果BASIC的扩展部分。

习　题

（一）下列程序的运行结果是什么？

```
10  A = 0
```

```
20  FOR  I=0  TO  10
30  I=I+1
40  A=A+1
50  NEXT
60  PRINT  I, A
70  END
```

（二）改正下列两个程序中的错误：

（1）求5!的程序。

```
10  FOR  I=1  TO  5
20  A=1
30  A=A*I
40  NEXT  A
50  PRINT  "N!=";  A
60  END
```

（2）求$1+\frac{1}{1!}+\frac{1}{2!}+\frac{1}{3!}+\cdots+\frac{1}{n!}$的程序。

```
10  A=1 : V=1
20  INPUT "N=?";  N
30  FOR  J=1  TO  N
40  FOR  I=1  TO  N
50  A=A*I
60  V=V+1/V
70  NEXT  J
80  NEXT  I
90  PRINT  V
100 END
```

（三）编写程序，用来计算并打印100个数中正数的平方根的算术平均值。（100个数可由键盘输入，它们有正，有负）。

（四）编写一个程序，求出乘积为399的两个相邻奇数。

（五）鸡兔同笼。共有头M个，脚N只，编写一个程序，求鸡兔各多少只

(M，N可由键盘输入)。

(六)100匹马驮100担货，规定大马一匹驮3担，中马一匹驮2担，小马2匹合驮1担。要100匹马正好驮完这100担货，且不许有一匹马驮不够规定数。试编写一个程序，求大、中、小马各应有几匹？

(七)计算 $S=1+\frac{1}{1\times 2}+\frac{1}{2\times 3}+\frac{1}{3\times 4}+\cdots+\frac{1}{n(n+1)}$ 直到最后一项小于 10^{-8}，如果累加到20项时最后一项仍大于 10^{-8}也停止计算。试写出此程序，要求输出n，最后一项的值，累加和S。

(八)编写一个程序，计算自然数 M，N 的最大公约数(M，N由键盘输入)，并打印出来。

(九)用循环语句写一个程序，以计算把1元人民币换成零钱有多少种换法。

第六章　程序模块

小孩子搭积木，把一个个不同形状、不同大小的木块作不同的组合，可以搭出各种各样的模型来。把一个个具有特定功能的程序段作为若干个独立块，再把它们作不同的组合，也可以构成一个完整的程序，这种方法称作模块化结构程序设计方法。BASIC语言不是模块结构语言，但其中某些语句具有这一类性质。当然这些模块也是很不完善的，然而有了它们，可以简化程序的设计。

本章将介绍GOSUB、RETURN、POP、ON—GOTO、ON—GOSUB、ONERR、DEF、FN等保留字。其中前两个和后两个属于基本BASIC。

§1. 子　程　序

当一个具有特定功能的程序段在一个程序中多次被使用到时，如果在每次使用它的地方都写一遍，不仅写起来麻烦，输入机器时也费时间，还容易增加出错机会。而且这种程序较长，占用内存空间大，读起来也讨厌。可以设想，如果能只把它们写入一次，需要使用它们时就调用它们一次，那不就简单了吗？

在BASIC语言中，子程序的方法，就是适应这一要求产生的。

一、转子语句（GOSUB）和返回语句（RETURN）

把需要反复使用的那个程序段，单独编写一次。这个程序段称为子程序。子程序以外的程序称为主程序。

主程序应从程序中的最小行号写起，以便确保执行这个程序时，总是先从主程序开始。当主程序中需要执行那个具有特定功

能的子程序段时，就用转子语句（GOSUB）去调用（即执行）那个子程序。子程序结束后，通过一个返回语句（RETURN），返回到主程序中调用它的GOSUB语句的后续语句去，继续执行主程序。

转子语句的格式是：

〔行号〕 GOSUB 行号

保留字GOSUB后面的行号，是子程序的首行号。

返回语句的格式是：

〔行号〕 RETURN

保留字RETURN应写在子程序的结尾（即作为子程序的最后一个语句）。

注意，这里的 RETURN，应从键盘上将一个个字符顺序输入，而不能用回车键来代替（前者是语句，后者是一个字符）。

现在举一个例子，用以说明子程序的使用方法。

假定我们希望输出下面四个数据：

1 2 3 4 5，ABCDE，3 + 5↑2，3 = 5 + 3

要求在每数据之后跟上15个“*”号，接着输出一行“？”号，再输出一行“∧”号，再输出一行“=”号。

按照我们过去用的一般方法，你得用4行PRINT 语句分别输出那4个数据，且在每个数据之后加上15个“*”，数据之后的那三行“？”、“∧”和“=”，也得用4＊3＝12 个语句，重复写四次。

如果用子程序的方法来写这个程序，会简单些。

程序可以写成如下的样子：

```
10  PRINT 12345;
20  GOSUB  100
30  PRINT  "ABCDE";
40  GOSUB  100
50  PRINT  3+5↑2;
```

```
60  GOSUB 100
70  PRINT 3=5+3;
80  GOSUB 100
90  END
100 PRINT "* * * * * * * * * * * *"
110 PRINT "? ? ? 
```

```
? ? ? 
```

```
^^^^^^^^^^^^^^
      #130 = = =
```

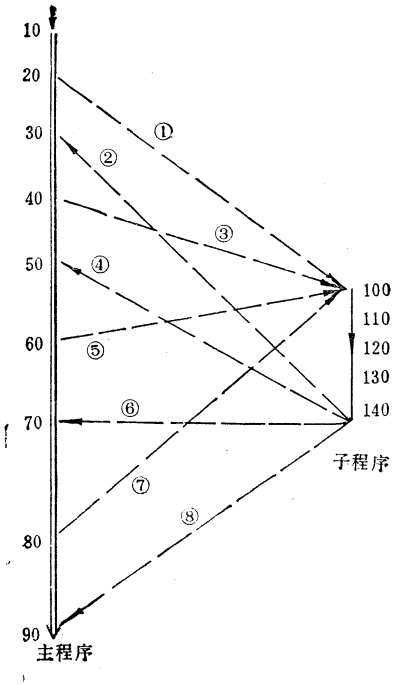


图6—1　执行顺序示意

GOSUB信息被RETURN使用过之后，也随之退栈。

二、子程序的用途

1．正如我们引入子程序概念时所说的，当一个有特定功能的程序段在程序中反复使用时，可以将这段程序写成子程序，凡是需要时就调用它一次。这样可以缩短程序长度，减少占用的内存空间。对输入和修改程序也带来一定好处。

2．假定某一个程序可以由若干个具有相对独立性的程序段组成时，也可以把这些程序段分别写成子程序。主程序中只要有几个GOSUB语句，就可以调用它们。如果给这些子程序加上一些注解，程序显得更为清晰。

例如，某水果店派出五部货车，到居民点去销售苹果(APPLE)、桔子(ARANGE)和香蕉(OANANA)。现就这三种水果的单价、利润和五部货车的销售情况，用表6—1列出：

表6—1

品　　名	苹　果	桔　子	香　蕉
每斤售出单价	0.52	0.68	0.75
每斤可收利润	0.10	0.13	0.16
第一货车卖出斤数	35	52	35
第二货车卖出斤数	28	48	57
第三货车卖出斤数	41	57	48
第四货车卖出斤数	28	30	65
第五货车卖出斤数	33	42	38

收市后结账，可用这样一种办法：报出各货车销售斤数，计算每种水果售出总斤数及金额，再算出利润来。最后打印出一个综合报表。

这样，把每步写成一个子程序，由主程序逐个调用就成了。

程序如下：

```
10  REM  MAIN  PROGRAM
20  GOSUB  100
30  GOSUB  200
40  GOSUB  300
50  END
100  REM  READ  DATA
110  READ  A1, A2, A3, A4, A5
120  READ  B1, B2, B3, B4, B5,
130  READ  C1, C2, C3, C4, C5
140  READ  A, B, C, D, E, F
```

```
150 RETURN
200 REM TOTAL OF UNITS SOLD AND OF SALES
    IN RENMINBI YUAN
210 A0 = A1 + A2 + A3 + A4 + A5
220 AA = A0 * A : AB = A0 * D
230 B0 = B1 + B2 + B3 + B4 + B5
240 BA = B0 * B : BB = B0 * E
250 C0 = C1 + C2 + C3 + C4 + C5
260 CA = C0 * C : CB = C0 * F
280 G = AA + BA + CA : H = AB + BB + CB
290 RETURN
300 REM REPORT VERSION
310 PRINT "NO. APPLE ORANGE BANANA"
320 PRINT " 1 "; A1; "          "; B1; "          "; C1
330 PRINT " 2 "; A2; "          "; B2; "          "; C2
340 PRINT " 3 "; A3; "          "; B3; "          "; C3
350 PRINI " 4 "; A4; "          "; B4; "          "; C4
360 PRINT " 5 "; A5; "          "; B5; "          "; C5
370 PRINT "TOTALS:    "; A0; "      "; B0; "      "; C0
380 PRINT "..........................................................."
390 PRINT "TOTALS YUAN: "; G
400 PRINT "PROFITS:              "; H
410 RETURN
500 DATA 3 5, 28, 41, 28, 33
510 DATA 52, 48, 57, 30, 42
520 DATA 35, 57, 48, 65, 38
530 DATA 0. 52, 0.68, 0. 75
540 DATA 0. 10, 0. 13, 0.16
```

这个程序的运行结果，输出：

```
NO.    APPLE  ORANGE  BANANA
 1       35      52      35
```

```
2        28        48        57
3        41        57        48
4        28        30        65
5        33        42        38
TOTALS:  165       229       243
.....................................................................
TOTALS  YUAN:  423.77
PROFITS:       85.15
```

可以看出，这个程序由三个子程序构成，每个子程序完成一部分工作，由主程序控制。执行完这些子程序也就完成了整个任务。读这个程序时，只要看一下三个子程序的注解，就一目了然了(也可以把三个注释语句分别加在20、30、40行之后)。

同时从程序中还可以看出，三个子程序的独立性是从功能上讲的，它们的独立性是相对的，子程序之间要通过同名变量传递数据。

3．有些程序（特别是常用算法程序）可能经常被用到。它们的计算过程一样，只是每次的初始数据不同。如果对这类程序每用一次编写一次，就非常麻烦了。当然可以把这类程序存入磁盘，供以后调出使用。但每次使用时，原始数据总不一样，因此原始数据就没有必要保存(有时这些数据可能还有机密性，用后必须清除)。这就可以把计算过程编成一个子程序，存入磁盘。每次只需调出此子程序。另写一个主程序，通过GOSUB语句调用此子程序即可。当然主程序中要给出原始数据。用这种方法可以在磁盘上建立一个子程序库，以后使用就方便多了。

比如，关于多项式求值问题，可有如下程序（还用前面那个求$Y=2.5X^5+4.1X^4+7.8X^3-4.45X^2+3X-6.8$的例子)：

```
10  INPUT "X=?"; X
20  DATA 5, 2.5, 4.1, 7.8, -4.45, 3, -6.8
30  GOSUB 100
```

```
40  END
100  READ  N, Y
110  FOR  I=1  TO  N
120  READ  A
130  Y=Y*X+A
140  NEXT
150  PRINT "Y="; Y
170  RETURN
```

这个程序运行时，只要输入一个X的值，就能求出

$Y=2.5X^5+4.1X^4+7.8X^3-4.45X^2+3X-6.8$的值。

其中100～170行为计算过程的子程序。若是把这个子程序存入磁盘，以后可以再次使用。

子程序存入磁盘的方法是先把主程序删掉（用DEL命令），再用SAVE命令存入磁盘（不要忘了给出名字）。

假如下次计算多项式

$Y=3X^3+5X-8$

时，可用LOAD命令（也别忘了给出名字）把子程序调出来，然后敲入下列一段主程序，就可以运行了：

```
10  DATA 3, 3, 0, 5, -8
20  INPUT "X=?"; X
30  GOSUB 100
40  END
```

三、使用子程序时应注意的几个问题

1．子程序可以嵌套，嵌套深度不得超过24。

与循环嵌套类似，子程序中再调用另一个子程序，叫子程序嵌套。

比如有这样一个题目：从键盘上依次读入若干个数，求出它们的总和与平均值，并在一行上打印出来。当读入的数为0时，标

志读数结束。

这个题目用子程序方法可写成：

```
10  GOSUB  100
20  GOSUB  200
30  END
100  N=0 : S=0
110  INPUT  A
120  IF  A=0 THEN  150
130  GOSUB 160
140  GOTO  110
150  RETURN
160  S=S+A : N=N+1
170  RETURN
200  PRINT  S, S/N
210  RETURN
```

这个程序就是在100行开始的子程序中，又调用了160开始的子程序。

程序执行顺序，由图6—2示出。

前面我们提到过，子程序的返回地址，是由子程序栈控制的。这里我们用图6—3来描绘一下这个程序使用栈的过程。

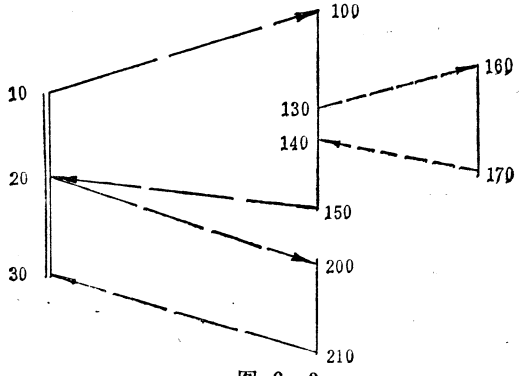


图 6—2

执行程序前，子程序栈中无数据，栈指针指向栈底(如图6—3 (a))。执行第10行时，遇到 GOSUB 100，有关数据进栈，栈指针上移（如图6—3(b))，转从100行执行。如果120行的条件不成立，则执行130行，又遇 GOSUB 160，有关数据又进栈（如图6—3(c))，再转160行。执行到170行时，遇RETURN，将栈顶数据取出，控制返回140行，栈指针下移（栈顶那组数据退栈，如图6—3(d))。由140转110行，120行，当120条件不满足时，栈状态又变成图6—3(c)的样子，执行170行时，RETURN又使栈状态变成图6—3 (d)。如此反复。直到120行的条件成立，转 150 行，遇 RETURN，栈状态变为图6—3(e)。这时控制程序做 20行，又遇 GOSUB 200，栈状态如图 6—3(f)，程序经由 200 行执行起直到210行遇RETURN，栈状态变为图6—3(g)，程序转30行结束。

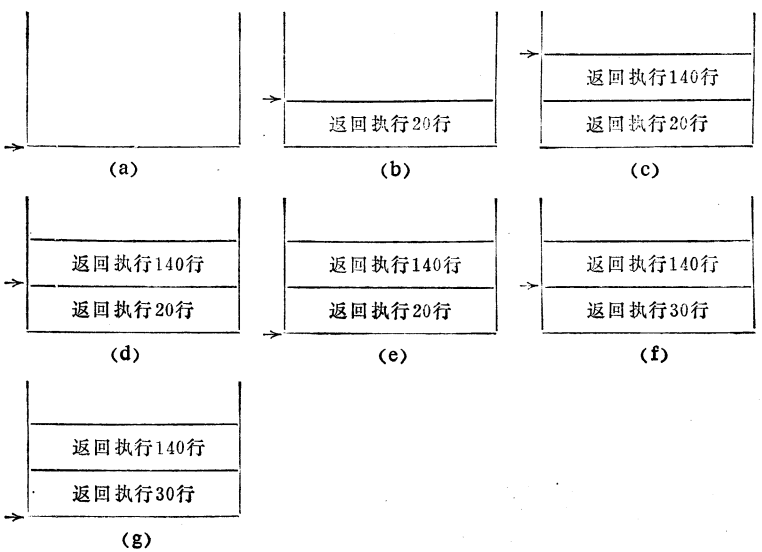


图6—3

各图中栈指针以上的数据是无效的（在栈外)，我们故意留下它们，表明并不是在内存中真正取消了它们的存在。

可见，用栈中数据控制子程序的返回是很有效的。再多嵌套几层也不会搞乱。

2．子程序虽可再次调用子程序，但BASIC语言中规定：子程序不能调用自己。既不能直接调用，也不能间接调用。

因为这样调用时，很容易导致嵌套深度的越界（超过24）。

比如下列程序：

```
10  A = 0
20  GOSUB  100
30  END
100  A = A + 1
105  PRINT  A
110  GOSUB  100
120  RETURN
```

这个程序运行时，输出从1到24这24个数(每个数占一行)，最后一行显示

```
? OUT  OF  MEMORY  ERROR  IN  100
```

因为执行20行时，返回 30 行的信息进栈，程序转去执行100行，A累加，105行打印A，到110行又遇GOSUB 100，返回120行的信息进栈。转到100行，A再累加，打印A，又遇110行的GOSUB 100，返回120行信息，再进栈（栈指针上移）。如此重复下去。由于上述过程只有进栈，没有退栈，使栈中信息超过24组，故在打印A值为24之后，显示存贮器溢出的出错信息。它实际上是指栈溢出。

然而，明确了机器的处理过程，作适当控制，子程序自己调用自己也是可能的。用的好可以简化程序设计。

例如下例：

```
10  A = 0
20  GOSUB  100
```

```
30  END
100  A=A+1
110  PRINT  A,
120  IF  A>6  THEN  140
130  GOSUB  100
140  RETURN
```

这个程序运行时，栈的状态变化如图6—4所示（其中(a)为栈初态)：

执行20行后，栈状态如(b)，程序转去执行100行，110行，当120行的条件不满足时，执行130行，栈状态变为(c)。这时又转回100行，继续执行，到120行条件仍不满足，再执行130行，栈状态变为(d)。如此直到栈状态变为(g)。这时120行的条件满足，转140行，遇RETURN，依栈指针指的内容(即栈顶信息)，控制程序返140行，栈指针下移(退栈)，成(h)。140行又是RETURN，再依栈顶信息转140行，并退栈。如此直到栈状态变为(*l*)。这时遇RETURN时，程序转30行结束，栈状态恢复初态(*m*)。

这个程序运行时，输出如下：

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

可见，由于120行的控制作用，子程序没有无限嵌套下去。

这种子程序调用它自己，称作递归调用。递归调用是算法语言中功能较强的一种方法，BASIC语言没有提供这种功能，当然是一大憾事。然而，如能对嵌套深度得到控制，简单的递归调用也还能够实现。上面的例子就是。

对于初学者而言，我们建议还是遵守BASIC的规定，以免弄巧成拙。

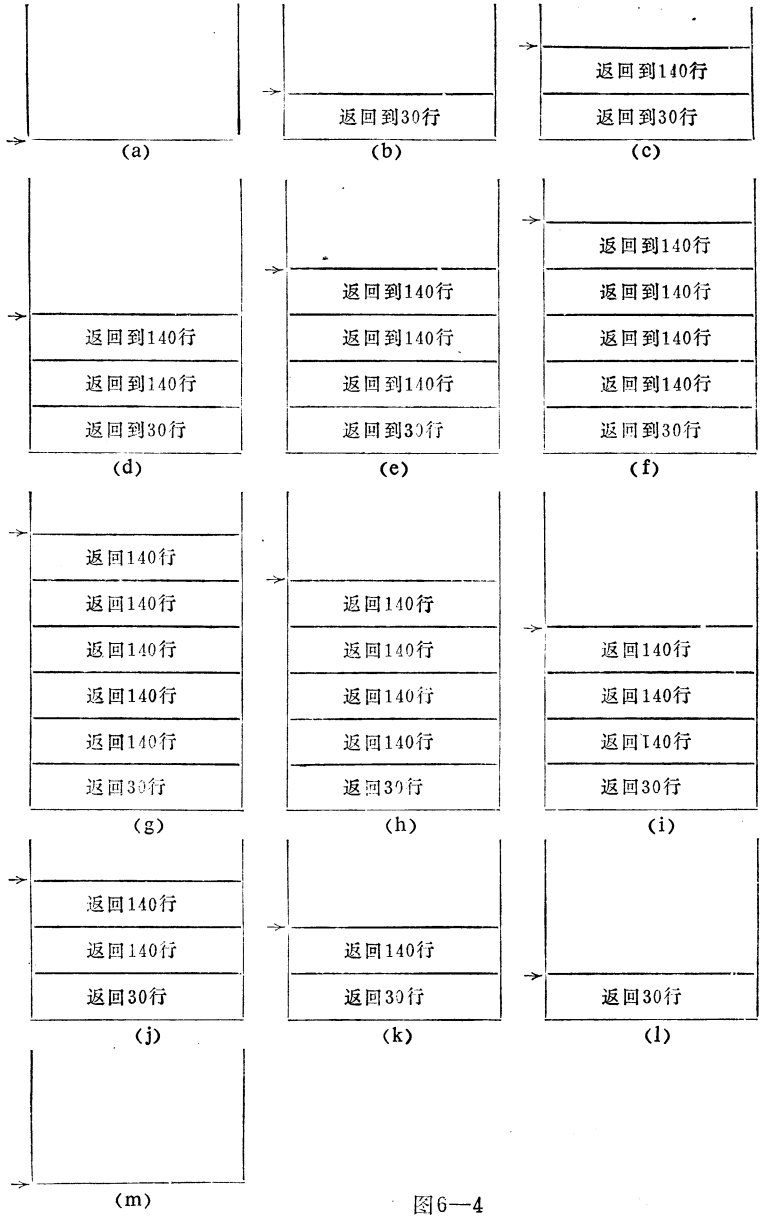

返回到30行
返回到140行
返回到30行
返回到140行
返回到140行
返回到30行
返回到140行
返回到140行
返回到140行
返回到30行
返回到140行
返回到140行
返回到140行
返回到140行
返回到140行
返回到140行
返回到140行
返回到140行
返回到140行
返回到140行
返回到30行
返回到140行
返回到140行
返回到30行
返回到140行
返回到30行
返回到30行
(a)
(b)
(c)
(d)
(e)
(f)
(g)
(h)
(i)
(j)
(k)
(l)
(m)

图6—4

3．不允许未经GOSUB 语句而进入子程序中；也不允许未经RETURN语句退出子程序。

因为只有经过 GOSUB 语句，返回地址的信息才能进栈，遇RETURN语句时，才知道返回什么地方。因此，没有经过GOSUB进入子程序时，子程序末的RETURN将不知道返回什么地方。同样，如果未经RETURN退出子程序，进入该子程序时那组GOSUB信息仍处于栈顶，还会被别的RETURN利用，容易造成逻辑上的混乱。和循环栈类似，没有经过RETURN退出子程序，实际上机器仍认为它还在子程序中。

不经GOSUB语句进入子程序的情况，在初学者编写的程序中经常发生。

例如程序：

```
10  GOSUB  100
20  PRINT  A
100  A = 3 * 5
110  RETURN
```

这个程序在执行完20行后，自动地进入了子程序。于是在执行110行时，将找不到返回地址而出错。

通常，在一个子程序的前面如果是主程序的话，要根据需要加上必要的 END (STOP)或 GOTO语句。比如上面那个程序，可加入：

```
30  END
```

如果从200行起还有主程序的后续部分，则加：

```
30  GOTO  200
```

4．POP语句的使用。

POP的格式是：

〔行号〕 POP

它的作用就是从子程序栈中弹走一组GOSUB信息（即栈指针

下移一组数据位置），从而使其后遇到的RETURN 不是返回调用它的那个GOSUB的后续语句，而是返回到相邻外层那个GOSUB的后续语句。它主要用于嵌套的GOSUB结构中。

注意：POP 和 RETURN 的相同作用是退栈。不同的是，RETURN要依据退出的那个栈数据返回程序的特定处，而POP不返回，它执行结束后，继续执行其后续语句。

比如程序：

```
10  GOSUB  30
20  END
30  GOSUB  60
40  PRINT  SQR (A)
50  RETURN
60  INPUT  A
70  IF  A>=0  THEN  100
80  PRINT  "A<0"
90  POP
100  RETURN
```

这个程序运行情况是：

```
RUN
? 4  (输入4)
2
RUN
? 9  (输入9)
3
RUN
-4
A<0
```

读者可用栈状态画出这个程序执行情况。另外，请读者考虑，在非嵌套的子程序结构中使用 POP，将是什么结果？在主程序中

使用POP又会怎样？

5.一般地，GOSUB和RETURN是成对地使用的。即通常情况下，有一个GOSUB，就有相应的RETURN。

但若是在子程序中使用IF语句，那么一个GOSUB，可对应两个或两个以上的RETURN。这时仍可以把这些RETURN看作是单一对应GOSUB的。因为每次进入这个子程序时，总是由其中一个RETURN退出。另外，一般 GOSUB 语句所跟的行号应为子程序的首行号，RETURN也应放在子程序之尾。但有时根据程序需要也可以在 GOSUB 之后跟一个子程序中间的行号。还有上面说过的，在子程序中未结束之前，也可用一个条件语句控制提前退出子程序。实际上，这时你可以认为：从GOSUB 指定的行号起直到需要执行的 RETURN，是一个子程序。可见子 程 序的使用，还是很灵活的。

6.循环体内可以使用转子语句，子程序中也可以使用循环语句，它们之间允许相互嵌套。但这种嵌套不得交叉。因为循环栈和子程序栈数据结构不同，交叉嵌套将会搞乱栈的控制权，导致出错。

§2. 开关语句与出错处置

上一节的 POP 和这一节将要介绍的语句，均 不 属 于 基 本 BASIC。但由于它们和子程序联系密切，放在这里 更适宜些。

一、开关语句

开关语句，好象是一个波段开关一样，可以根据程序的需要，拨到不同的档上，以实现不同的功能。

开关语句的格式是：

行号 ON 算术表达式 GOTO 行号 〔{，行号}〕

行号 ON 算术表达式 GOSUB 行号 〔{，行号}〕

GOTO(或GOSUB)之后，可以跟一个行号，也可跟若干个行号，而每两个行号之间要用逗号分隔开。这些行号可依顺序叫做第一个行号，第二个行号等等。ON之后的算术表达式，取其整数值使用(若为带小数点的数，舍掉小数部分)。语句的功能是：当算术表达式的值为几时，就执行第几个行号：对ON—GOTO，就转到那个行号去；对ON—GOSUB则转入那个行号开始的子程序。

比如程序段：

```
100  ON  A GOTO  400, 800, 1000
110  END
```

相当于程序：

```
100  IF  A=1  THEN  400
102  IF  A=2  THEN  800
104  IF  A=3  THEN  1000
110  END
```

类似地，程序：

```
100  ON  A  GOSUB  400, 800, 1000
110  END
```

相当于程序段：

```
100  IF  A=1  THEN  COSUB  400 : GOTO  110
102  IF  A=2  THEN  GOSUB  800 : GOTO  110
104  IF  A=3  THEN  GOSUB  1000
110  END
```

从上面的类比中，我们应看出下列两点：

1.当ON后的表达式的值大于GOTO(或GOSUB)后的行号个数或小于1时，则本开关语句不执行，自动执行后续语句(如上例中，当A>3或A<1时，100行什么也不做，直接转110行)。

2.开关语句是一个语句，每执行这个语句时，总是依算术表达式的值，执行GOTO(或GOSUB)一个行号。对于ON—GOTO

而言，一经转走，就不由ON—GOTO控制了。而对于GOSUB，转去执一子程序之后，还要返回来，它是返回本开关语句的下一个语句（上例为110行的END），而不是再GOSUB后一个行号。

以上两点，是初学者容易误会的。

开关语句可根据程序运行时情况的不同，控制做不同的后续处理。在解决实际问题中，很有用处。

二、出错处理

一个程序在运行时一旦发现错误，会打印出错信息，并中断程序的运行。

而运行中发现的错误是各种各样的。有些错误是致命性的，无法补救，只得修改程序重新运行。有些错误则是可以补救的。对于这类可补救的错误，我们往往不希望中断程序的运行，而愿意给出补救，使程序继续运行下去（特别是对一些运行时间较长的程序）。出错处置就是为此目的而提供的一个手段。

1．如何检出一个错误。

通常，程序中的错误，是由解释程序来发现的。这时根据错误的类型，在屏幕上给出相应的出错信息，然后动态停机，这就回到了BASIC初态，给出提示符]。

如果想让解释程序发现源程序中的错误后交用户程序处理，那就得用ONERR GOTO语句。

这个语句的格式是：

行号 ONERR GOTO 行号

这个语句应设置在可能出错的程序段之前。当机器运行程序时遇到了这个语句，以后在运行程序中如有错误，便自动转移到ONERR GOTO之后所指定的程序行去。因此应从那个行号开始，写出一段出错处理程序。

2．如何写出错处理程序

错误的性质是多种多样的，处理的方法也不同。这要根据你

所检查的那段程序来决定。可以作出一些提示，也可以改变一些数据。

出错处理结束后，要使程序返回出错语句继续运行。

从出错处理转回去，一般不用GOTO语句，而用RESUME（重做一次）命令。

其格式是：

行号 RESUME

它总是和ONERR GOTO 配合使用的。它写在出错处理程序之末，程序遇到这个语句（命令）时会返回出错处继续执行。

需要指出的是，如果没有出错就遇到RESUME，或在出错处理中又发生错误，将会出现混乱。所以不要乱用RESUME，在出错处理中更要防止出错。

现举两个例子：

例1.

```
10  ONERR GOTO 50
20  INPUT X
30  PRINT SQR (X)
40  END
50  PRINT "DATA LESS THAN ZERO"
60  INPUT X
70  RESUME
```

这个程序可以检查出：一旦20行输入一个小于0的数(负数)，30行就会出错。由10行的信息使程序转到50行。作出提示后，再由60行输入一个正确的X，经由70行控制返回30 行重做（求平方根)。40行上结束。

例2.

```
10  ONERR GOTO 80
20  INPUT "X = "; X
30  PRINT 1/X
```

```
40  ONERR GOTO 110
50  INPUT "Y="; Y
60  PRINT SQR (Y)
70  END
80  PRINT "DATA MUST NOT BE ZERO"
90  INPUT "X="; X
100 RESUME
110 PRINT "DATA LESS THAN ZERO"
120 INPUT "Y="; Y
130 RESUME
```

这个程序可以处理两种输入错误中的一个。第10行负责控制30行不能做除法时转移到80行，第40行负责控制60行不能开平方时转移到110行。100行负责处理完分母为零情况后返回30行，130行负责处理完负数不能开平方的情况后返回60行。

§3. 用户自定义函数

在BASIC中，除了机器内部定义的一批函数外，也允许用户自己定义某些函数。这类函数是在程序中定义的，也只能在本程序中使用。

一、用户如何定义一个函数

自己定义函数，要通过自定义函数语句。

该语句的格式是：

〔行号〕 DEF FN 名字（实型变量名）= 算术表达式

其中的名字，构成规则同变量名的构成规则，但不得带尾符（如%或$）。自定义函数的值为实型量。FN 名字合起来称为函数名。

括号内的实型变量名，又称形式参数。在定义时不考虑具体取值为多少。

假定某个程序中要反复用到下面的计算公式，这时就可以把这个式子定义成一个函数，以后每当需要按这个公式计算时，只要给出自变量，就能像一般算术函数那样来调用：

$100X^2+5X-205$

如果名字用AB，可以这样来定义这个函数：

```
100  DEF  FN AB (X) = 100 * X * X + 5 * X - 205
```

这个定义了的函数，函数名可视作 FNAB。可见自定义的函数名必须以FN打头，其后的字符可由用户自定。

二、自定义函数的调用

一个函数被定义之后，就可以在同一个程序中被调用了。

如象上面定义了的那个函数，下列各程序行都是合法的调用：

```
200 A = FN AB (3)
⋮
300 PRINT FN AB (3+5)
⋮
400 T = 35 : S = 4
410 U = FN AB (T+S) - 2 * FN AB (S * 3)
```

执行200行，相当于把数3作为X 的值代入那个公式中，将计算结果赋给变量A。

执行300行，相当于把3 + 5的结果8 作为X的值，代入那个公式中，把计算结果打印出来。

……

这些调用中用到的、括在圆括号中的3，3 + 5，T + S，S*3等都有具体的值，称为实在参数。

可见，自定义函数的调用格式是

〔行号〕 …FN 名字（算术表达式）

格式中的…表明那里需加适当的语句成分。因为其后的部分仅是一个函数，不是一个语句。

三、使用自定义函数应注意的几个问题

1．自定义函数必须先定义，然后才能调用，顺序不可倒置。

2．苹果机上自定义函数的参数，只允许一个。

3．形式参数一般应出现在自定义时的算术表达式中。如果不是这样，则该函数被调用时与参数无关。即使这样，调用时也得给出一个参数（这时称这个参数为哑元）。

如程序：

```
10  DEF FN A (X) = Y + 1
20  PRINT FN A (5)
30  Y = 3
40  PRINT FN A (Y)
50  PRINT FN A (Y + 2)
60  Y = 8
70  PRINT FN A (Y - 1)
80  PRINT FN A (4)
90  END
RUN
1
4
4
9
9
```

可见，对10行定义的函数，不管调用时实参为什么值，它总是取当时Y值加1作为函数的结果的。

4．自定义函数允许嵌套使用，且可调用它自己。

如：

```
10  DEF FN T (X) = X + 1
20  A = 1
30  PRINT FN T (FN T (FN T (A)))
40  END
```

```
RUN
4
```

§4. 应用程序举例

以下就本章所讲内容，再介绍几个应用程序的实例。

例1. 部分累计及总计某公司的销售额，并打印报表。

设某公司有若干推销员，它们各自找用户推销产品。每隔一定时间作一次统计：累计每个推销员之销售额及总公司的销售额，并打印出来。

要求表格有表头，接下去登记每个推销员对每个用户的销售额。一个推销员的项目登记完之后，用30个“·”连成一条横线，其下结算出该推销员之累计销售额。接下去是另一个推销员的统计结算。整个公司的推销员结算完后，也用30个“·”号连成一条横线，其下结算出全公司的总销售额。此外，除表头外，要求每印出50行要换一页（以便装订）。每页都得有表头。

下面列出了这个程序。程序中所设各变量名分别表示：

C——当前页码；

L——当前行号；

SU——每个推销员之销售额；

T——公司总销售额；

SN $——推销员号；

CN $——用户号；

A——某个推销员对用户的销售额；

N——公司共有推销员人数；

程序中700行开始的 DATA 语句，示意性列出了几个推销员号、用户号及该推销员向该用户的销售额。

程序如下：

```
10  L=50 : P=50 : C=1
20  D$="……"
30  READ N
40  SU=0 : T=0
50  READ SN$, CN$, A : T$=SN$
60  FOR I=1  TO N-1
70  REM  TITLE PRINTING
80  GOSUB 600
90  GOSUB 400
100 SU=SU+A
110 READ SN$, CN$, A
120 IF SN$=T$ THEN 170
130 GOSUB 500
140 PRINT "  SUBTOTAL:"; SU
150 PRINT : T$=SN$
160 T=T+SU : SU=0
170 NEXT
180 GOSUB 400
190 GOSUB 500
200 SU=SU+A
210 PRINT "          SUBTOTAL:"; SU
220 PRINT
230 T=T+SU
240 GOSUB 500
250 GOSUB 500
260 PRINT "          TOTAL:"; T
270 END
400 PRINT SN$; "    " ; CN$; "    "; A
410 L=L+1
```

```
420 RETURN
500 PRINT D$
510  L=L+1
520 RETURN
600 L=L+1
610 IF L<P  THEN 670
620 PRINT "          * *SALES REPORT  * *
          PAGE:", C
630 PRINT : PRINT
640 PRINT "SALE NO. CUST.NO. AM OUNT"
650 PRINT : PRINT
660 C=C+1 : L=1
670 RETURN
690 DATA 6
700 DATA "S001", "C001", 100
710 DATA "S001", "C002", 250
720 DATA "S001", "C010", 400
730 DATA "S002", "C005", 370
740 DATA "S002", "C008", 450
750 DATA "S003", "C021", 1000
 ⋮
 ⋮
```

程序中用了三个子程序：

400～420行，控制输出每个推销员对某用户的销售额；

500～520行，控制输出连续30个“.”号；

600～670行，控制输出表头。

程序运行后，输出如下报表：

```
          * * SALES REPORT  * *          PAGE:×××
SALE NO.  CUST.NO. AMOUNT
S001      C001     100
```

```
S001        C002        250
S001        C010        400
.....
            SUBTOTAL:750
S002        C005        370
S002        C008        450
.....
            SUBTOTAL:820
.....
.....
        TOTAL:×××
```

例2. 质因数分解。从键盘上输入一个自然数，把它分解成质因数。

如：

$12 = 2 \times 2 \times 3 = 2^2 \times 3$

$18 = 2 \times 3 \times 3 = 2 \times 3^2$

当键盘上输入的数为0时，结束。

程序如下：

```
10  REM *** PRIME FACTOR ***
20  REM THIS PROGRAM DETERM INES THE PRIME
    FACTORS OF AN INTEGER
30  REM MAIN
40  HOME
50  PRINT "PRIME FACTOR ROUTINE"
60  PRINT "ENTER A ZERO TO END"
70  GOSUB 1000
80  PRINT : PRINT
90  PRINT "GOOD BYE"
100 END
1000  REM NUM
```

```
1010  PRINT
1020  INPUT "NUMBER:"; Y
1030  IF Y=0 THEN 1060
1040  GOSUB 2000 : REM CALC
1050  GOTO 1010
1060  RETURN
2000  REM CALCULATE
2010  PRINT Y; " = ";
2020  X=Y : K=1
2030  FOR I=2  TO X
2040  S=0
2050  IF INT (Y/I)=Y/I THEN Y=Y/I : S=S+1 : GOTO
      2050
2060  GOSUB 3000 : REM DISPLAY
2070  NEXT
2080  PRINT
2090  RETURN
3000  REM DISPLAY
3010  IF S=0 THEN 3070
3020  IF K=0 THEN 3040
3030  K=0 : GOTO 3050
3040  PRINT" * ";
3050  IF S=1 THEN PRINT I; : GOTO 3070
3060  PRINT I; " ↑ "; S;
3070  RETURN

RUN
PRIME FACTOR ROUTINE
ENTER A ZERO TO END

NUMBER:12
```

```
12 = 2 ↑ 2 * 3

NUMBER:18
18 = 2 * 3 ↑ 2

NUMBER:0

GOOD BYE
```

这个程序的第10行是一个注解，可以认为是程序的名字。

第20行也是一个注释，注明程序的功能。

第30～100行是主程序。

第1000～1060行是第一个子程序。

第2000～2090行是第二个子程序。

第3000～3070行是第三个子程序。

程序结构用了嵌套方式。

第三个子程序主要功能是输出显示。

第二个子程序主要功能是求质因子。由于每次求完要输出显示，所以由它调用第三个子程序。

第一个子程序主要功能是输入自然数，因为每次输入自然数后要求它的质因子，所以由它调用第二个子程序。

例3. 求X在－2到7的范围内的超越方程 sin (X) = 0的所有根。

对于求超越方程和高次方程的根，一般可用对分区间套法。

设$f(x)=0$，在$[a,b]$区间内有根，且函数$f(x)$在此区间上是连续的。

我们可将$[a,b]$区间分割成若干等分(或以确定的步长间隔)，形成若干个小区间（如图6—4)。

从左到右，依次检查在每个小区间的两个端点上函数值是否异号。如果不是异号（是同号)，说明函数在这个小区间上无解，

向右顺推一个小区间，再用上法检查。如果所有小区间上均无解，则函数在整个区间上无解。

当发现某个小区间的两个端点上，函数值异号时(如$[a_1,b_1]$小区间)，表明根在其中。

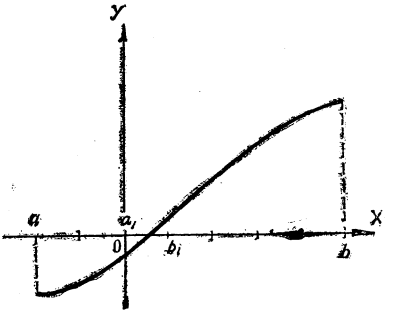


图6—4

这时再对小区间对分（一分为二)，用同样的方法确定根在小区间的左半部，还是在小区间的右半部（二者必具其一)。再将有根的这半个小区间对分，确定根在小小区间的左半部还是右半部。这样可以反复分下去。一直到对分到很小很小时，最后一个对分点即可当作近似解。

$[a_1,b_1]$区间找完后，还要扫描其右侧的各个等分区间。如发现有根，类似上述对分方法，求出近似解。一直扫描完最右一个等分区间，所有的根也就求出来了。

可见这种方法是通过等分区间查出一个根的范围，再用对分区间求出近似解。在对分过程中，一个个区间是套起来的：大的套小的，而根在这些套的最内层。也可以说这些区间套套着一个根。

这样，在求根时，我们可以将方程的左部定义为一个自定义函数(比如用 FNZ(X))，则对分求根的过程如图6—5所示。其中A为等分区间的左端点，B为等分区间的右端点。X0 为对分点；ε是用户给定的一个很小的数，指明当对分的小区间比ε还小时，就不再对分了（认为误差已够，X0即可作为近似解)。

下面是求X在－2到7区间上的$\sin(x)=0$的全部根的程序。程序中A，B表示区间的左右端点，N代表等分数，E代表根的精度(即ε)。

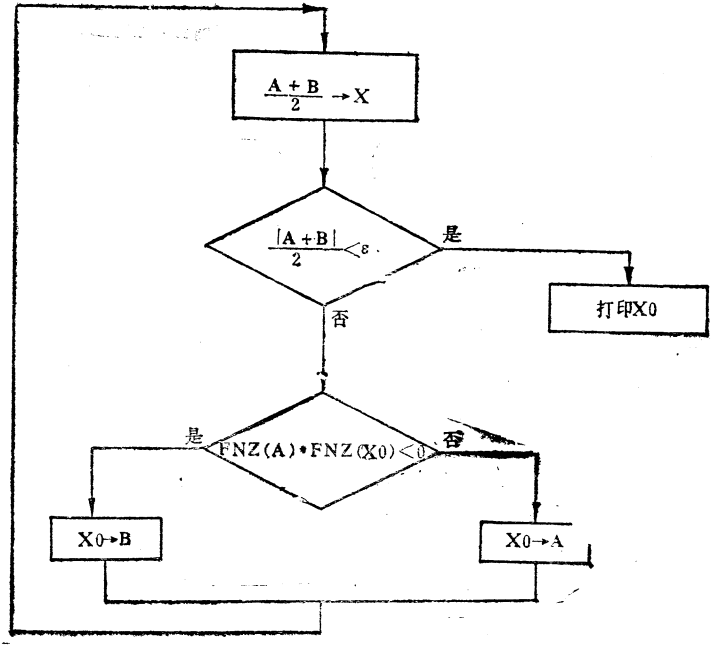


图6—5

```
10  DEF FN Z (X) = SIN (X)
20  DATA  -2, 7, 1E-07
30  GOSUB 100
40  END
100 REM QUJIANTAO
110 READ A, B, N, E
120 H = (B - A)/N : M = 0
130 FOR I = 1 TO N
140 A1 = A + (I - 1) * H : B1 = A + I * H
150 IF FN Z (A1) * FN Z (B1) >= 0 THEN 240
160 X0 = (A1 + B1)/2
170 IF ABS (A1 - B1)/2 < E THEN 220
```

```
180 ON SGN (FN Z (A1) * FN Z (X0)) + 2 GOTO  200,
    210, 210
200 B1 = X0 : GOTO 160
210 A1 = X0 : GOTO 160
220 M = M + 1
230 PRINT "X"; M; " = "; X0
240 NEXT
250 RETURN
```

本程序运行结果：

X1 = 4.15162908E－08

X2 = 3.14159261

X3 = 6.28318528

上述程序中，在一个小区间内对分时，并未使用FOR语句，而是经160行求出对分点后，再由200行或210行转160行，作下次对分。只有170行的条件满足，才停止在这个小区间上对分。这种对分是没有固定次数的，所以还是不用FOR语句为好。

程序中180行用了ON—GOTO 语句，也用了SGN 函数，是为了熟悉开关语句，并不是一种很好的程序结构。直接使用条件转移语句可能效率更高些。

对这类程序要注意的一个问题是，当开始等分的区间太大时(N 取值很小)，有可能漏掉一些解，因为这时函数的曲线可能两次穿过X轴而没找出来；把N取得大点，开始等分的区间就小了，这可以避免漏根，但运行效率要低。所以要取适当的N 值。

此外 E的大小，也严重影响运行效率。这要根据问题的需要而适当选取。

习　题

(一) 写出下列程序的执行结果，并写出运行时执行行号的顺序。

```
10  GOSUB 100
20  GOSUB 200
30  GOSUB 300
40  END
100 READ A, B
110 C=A+B
120 PRINT C
130 RETURN
200 READ A, B
210 C=A*B
220 PRINT C
230 RETURN
300 READ A, B
310 C=A/B
320 PRINT C
330 RETURN
400 DATA 15, 20, 25, 30, 35, 40
```

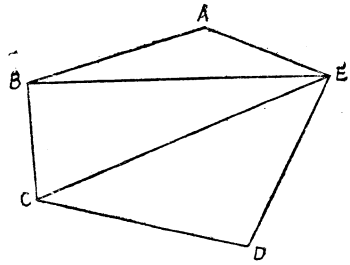


图6—6

（二）用子程序设计一个计算程序，用以计算图6—6五边形的面积。其中各线段长可用DATA语句给出（具体数字，由用户自己设定，但不能给出非法数据）。

（三）用子程序方法写出求3！+4！+5！的程序来。

（四）下列程序的运行结果是什么？

```
10  GOSUB 100
20  GOSUB 200
30  END
100 DEF FN A (X)=INT (X*RND (0)+1)
110 FOR X=1 TO 10
120 PRINT FN A (7)
130 NEXT
140 RETURN
```

```
200 DEF FN A (I)=X+1
210 FOR I=1 TO 10
220 PRINT FN A (I)
230 NEXT
240 RETURN
```

（五）利用开关语句，对从键盘上采集来的各种不同数据，控制程序作不同的处理：对读入的零，只统计次数(读入多少次0)；对读入的正数，求出均值；对读入的负数，只找其两个边界值。数据总数不会多于10000个。作为标志：当读到－9999999时，控制程序结束。还要求当键盘输入非数字数据时，能做出错处理。

（六）下列程序，会出现什么情况

```
10  GOSOB 100
20  END
100 PRINT A; "  ";
110 GOSUB 200
120 PRINT A, B
130 RETURN
200 B=5
210 PRINT B; "    ";
220 GOSUB 100
240 RETURN
```

（七）下列程序运行时会出现什么情况。

```
10  M=4
20  GOSUB 100
30  END
100 IF M=0 THEN P=1 : RETURN
110 M=M-1
120 GOSUB 100
130 M=M+1
140 P=P*M
```

```
150 RETURN
```

(八) 读懂下列程序，说明程序的功能是什么。

```
10  REM * * COMPLETE ARITHMETIC DRILL
20  INPUT "ENTER YOUR NAME"; N$
30  INPUT "WHICH KIND OF PROBLEMS : 1 (ADD),
    2(SUBTRACT), 3(MULTIPLY) OR 4(DIV IDE)?"; K
40  INPUT "HOW MANY PROBLEMS?"; N
50  INPUT "HOW MANY DIGITS IN EACH NUM BER
    IN A PROBLEM?"; D
60  R=0
100 REM * *SUBROUTINE SELECTION ROUTINE
110 ON K GOSUB 200, 400, 600, 800
120 REM * *PRINT RESULTS
130 PRINT "CONGRATULATIONS"; N$ "YOU HAVE
    COMPLETED?"; N; "PROBLEMS"
140 PRINT "WITH A SCORE OF"; R
150 INPUT "DO YOU WANT TO DO SOME MORE?"; K
160 IF K$="YES" THEN 30
170 END
200 REM * *SUBROUTINE FOR ADDITION
210 FOR I=1 TO N
220 GOSUB 1000
230 REM * * PRINT PROBLEM AND REQUE ST
    ANSWER
240 PRINT : PRINT A; "+"; B; "=";
250 INPUT C
260 REM* *TEST ANSWER
270 IF C < > A+B THEN PRINT "SORRY, WRONG
    ANSWER" : GOTO 300
280 GOSUB 2000
```

```
290 PRINT "RIGHT ON.GOOD"
300 NEXT
310 RETURN
400 REM * * SUBROUTINE FOR SUBTRACTION
    PROBLEMS
410 FOR I=1 TO N
420 GOSUB 1000
430 REM * * PRINT PROBLEM
440 PRINT : PRINT A; "-"; B; "=";
450 INPUT C
460 REM * * TEST ANSWER
470 IF C < > A-B THEN PRINT "SORRY, WRONG
    ANSWER" : GOTO 500
480 GOSUB 2000
490 PRINT "GOOD ANSWER"
500 NEXT
510 RETURN
600 REM * * SUBROUTINE FOR MULTIPLICATION
610 FOR I=1 TO N
620 GOSUB 1000
630 REM * * PRINT PROBLEM
640 PRINT : PRINT A; "*"; B; "=";
650 INPUT C
660 REM * * TEST ANSWER
670 IF C < > A*B THEN PRINT "SORRY, WRONG
    ANSWER" : GOTO 700
680 GOSUB 2000
690 PRINT "GOOD ANSWER"
700 NEXT
710 RETURN
```

```
800 REM * * SUBROUTINE FOR DIVISION
810 FOR I=1 TO N
820 GOSUB 1000
830 REM * * PRINT PROBLEM
840 PRINT:PRINT A; "/"; B; "=";
850 INPUT C
860 REM * * TEST ANSWER
870 IF C < > A/B THEN PRINT "NO GOOD" : GOTO
    900
880 GOSUB 2000
890 PRINT "NICE ANSWER"
900 NEXT
410 RETURN
1000  REM * * SUBROUTINE TO GENERATE
      NUMBERS
1010  A=INT (10↑D*RND (1)) : IF A<10 ↑D/10-1
      THEN 1010
1020  B=INT (10↑D*RND(1)) : IF B<10↑D/10-1
      THEN 1020
1030  RETURN
2000  REM SUBROUTINE TO KEEP SCORE
2010  R=R+1
2020  RETURN
```

第七章　数组与字符串函数

到现在为止，我们在前几章中所用到的各种变量(实型变量、整型变量、字符串型变量)，都有一个特点，就是在程序运行的任何时刻，每个变量名只能代表一个数据。这种变量名，我们叫做简单变量。

用简单变量固然可以编写出各种各样的程序。但是如果在程序中有成组地出现的数据时，程序的编写与处理，将是十分麻烦的。

假定我们要处理200个学生的考试成绩，每人考6门课，共有1200个分数。这1200个分数如果要反复使用，最好的办法是把这1200个数一次读入1200个变量，否则每次使用得从头读一遍。1200个变量在内存中，随时可以使用，这当然很方便。然而这是不可能的。因为所有可能组成的变量名总共还不到1000个。

这一章我们将引入下标变量和数组，对于处理成组地出现的同一数据类型的数据，将是一个有效的工具。

这一章还要介绍一组与字符串有关的函数。它们在处理字符串时是很有用处的。

本章将介绍的保留字除DIM(它属于基本BASIC)外，还有保留字STORE、RECALL、LEN、STR$、VAL、ASC、CHR$、LEFT$、RIGHT$和MID$等。

§1．下标变量与数组的概念

一、下标变量

对于200个学生的考分，若是每个学生的总分已算出，则有

200分数。在数学上，我们可以用

$$a_1, a_2, a_3, \cdots a_{199}, a_{200}$$

来分别表示它们。

如果用简单变量

A1，A2，A3，…A199，A200

来表示它们，显然是不行的。因为A1、A10、A11、A12、…A100，A101…A110，A111…A199都被机器视作同一变量名。

但如若用

A(1)，A(2)，…A(199)，A(200)

来表示，机器是接受的，并能区分它们。

类似地，200个学生，每人6门考分，在数学上也可表示成

$$\begin{array}{llll} b_{1,1} & b_{1,2} & b_{1,3} \cdots & b_{1,6} \\ b_{2,1} & b_{2,2} \cdots & \cdots & b_{2,6} \\ \cdots\cdots & & & \\ b_{199,1} & b_{199,2} & \cdots & b_{199,6} \\ b_{200,1} & b_{200,2} & \cdots & b_{200,6} \end{array}$$

在BASIC语言中，也可用

B(1,1)，B(1,2)，…，B(1,6)

B(2,1)，B(2,2)，…，B(2,6)

……

B(199,1)，B(199,2)，…，B(199,6)

B(200,1)，B(200,2)，…，B(200,6)

来表示。机器也接受并能区分它们。

上面引入的A(1)，A(2)，…，A(200)，B(1,1)，B(1,2)，…，B(1,6)，…，B(200,6)等，都称作下标变量。其中的A，B，可象简单变量名那样构成。

可见下标变量是这样一种变量：在一个简单变量名之后，跟着一个小括号对，括号对中可以是一个整数，也可以是多个整数，

但两个整数之间加一个逗号分隔开。这些整数称为下标。

事实上，下标不仅可以用常数、数型变量、算术表达式，甚至可以用下标变量来写出。

当下标值非整数时，机器自动截舍其小数部分。

当变量B的值为5时：A(B)=A(5)，A(B+6)=A(11)

当A(5)=3时：A(A(5))=A(3)

从我们引入下标变量的过程可知，下标实际上是一组数据类型相同的数据的序号。因此它不得为负，也不得是字符串型数据。任何这类数据，都被视为非法的。

下标变量的数据类型同简单变量一样，可以是实数型、整数型，也可以是字符串型。它们的标志跟在下标变量名之后、小括号之前。

如：

A$(5)　是字符串型下标变量

A(5)　是实数型下标变量

A%(5)　是整数型下标变量

下标变量在程序中的用法，也和简单变量一样，任何一个下标变量都可以通过适当的语句取得数据，也可以出现在表达式中为其它变量提供数据或输出。

二、数组

具有同一个下标变量名的所有下标变量的总合，称为一个数组。这时，下标变量名又称数组名，每个下标变量称为该数组的一个元素。

比如200个学生的200个分数，可用200个下标变量

A(1),A(2),…,A(200)

表示，这200个下标变量组成一个数组，数组名为A，共200个元素，都是实型量。

类似地，200个学生，每个学生6门考分，可用下标变量

B(1,1)，B(1,2)，…B(200,1)，…，B(200,6)

表示。这1200个下标变量组成B数组。

构成一个数组的所有下标变量，它们的下标个数是相同的。下标个数称为数组的维数。如上面的A数组维数为1，B数组维数为2。苹果机允许一个数组多达88维。

由于数组名就是下标变量的名字，一个数组元素就是一个下标变量，数组的维数就是下标变量的下标个数，因此，在以后的叙述中，经常提到数组而很少提下标变量，但读者要意识到它们的内在关系。特别是对任何一个数组元素的任何操作，实际上也应看作是对一个下标变量的操作。上边我们已经谈到，任何一个下标变量在程序中与简单变量有相同的使用方法。因此，任何一个数组元素都可通过适当的语句得到数据，任何一个数组元素的值也一样可以拿来参加运算或被打印出来。

下述各语句都是合法的：

```
INPUT A(1)
INPUT B$(I), C%(3*I), D(2, 3*I)
READ A(2+A), A$(5*5), C%(6)
PRINT A$(5), B(4*I), A%(7*I+5)
A(I)=6+A*B(I, I+1)
A(5)=A(3)+B(I)*8
```

读者不难想到，使用简单变量时，N个不同的变量，必须取N个不同的变量名；而对一个数组中的N个不同元素(即N个下标变量)，只需取N个不同的下标即可。而且下标又可以通过程序的运行予以改变，这就给编写程序提供了较大的方便。

让我们再看一个例子。假定要求从键盘上输入10个数，把它们分别读入10个变量名。如果用简单变量，我们可以设这10个变量名为A0、A1、A2、…、A9。这样在程序中必须把这10个变量名一一写出：

```
10 INPUT A0, A1, A2, A3, A4, A5, A6, A7, A8,
A9
20 END
```

如果使用数组，程序可以改写成

```
10 FOR I=0 TO 9
20 INPUT A(I)
30 NEXT
40 END
```

在运行这后一个程序时，数组元素（即下标变量）A(I)随着循环变量I的改变而改变，逐次读入A(0)，A(1)，…，A(9)十个元素的值。

如果增加一个DIM语句，后一个程序只需把循环的终值改为1000或10000，它将能读入1000个或10000个数（DIM语句，将在下节介绍）。

而对前一个程序来说，姑且不说无法写出1000个、10000变量名，即使数据量少一点，三、五百个变量名要送入内存中，也够你麻烦的。此外，在程序结构上，也无法使用循环的办法。

三、数组表简介

象变量表一样，在程序运行过程中也要建立一个数组表，以便实现对数组任一元素数据的按名存取。由于数组元素是一个个下标变量，因此，数组表也称下标变量表。

每个数组的维数不同，下标大小也不一样，数组元素的个数就不尽一致。在没有使用DIM语句时，机器总是自动给定一个下标的值允许从0到10，共11个。

这样，在没有使用DIM语句时：

一维数组可以有11个元素；

二维数组可以有$11\times11=121$个元素；

三维数组可以有$11\times11\times11=1331$个元素；

……

在这个前题下，我们可以形象地设想一下数组的结构。

一个一维数组，好象在一条纸上，从左到右顺序地写出11个数据。这十一个数据，依次叫作第0号元素，第1号元素，…，第10号元素。下标变量的下标就是指的元素的号码。

一个二维数组，可以想象成把11条如上结构的纸条从上到下依次平行地贴在一页白纸上，它们分别地称为第0行，第1行，…第10行。而对每行来说，从左到右为11个元素。这样二维数组的121个元素构成了11行11列的一个方阵。第一个下标，指的是行的号码；第二个下标，指的是该行中的第几号元素。因此，数组元素A(2，3)指的是A数组的第二行中的第三个元素（即第二行第三列交叉处的元素——以后简称第二行第三列元素）。

类似地，如上结构的页，可以装订成一本书，就象一个三维数组。把若干书包在一起就是四维数组等等。

数组表就是按上述结构造出的。它有一个表头，这里记有数组的名字、长度维数和各最大下标的值，接下去便排放各元素。顺序是先放0号元素，后放末号元素；先放0行元素，后放末行元素；先放0页元素，后放末页元素，…。这样，程序运行时，只要用到某一数组元素，就可以按名字和下标值找到这个元素存放的位置，实现对它的存数或取数操作。

整型数组元素比实型数组元素占用空间少，因而节省内存。

对于字符串型数组，同样建立在这张数组表中，只是在每个元素的位置上不是登录具体数据本身，而是一个索引，指明该具体串值在内存中的地址和长度。当然有了这些索引项，同样可以取得具体数据。

§2. 数组说明语句(DIM)

前文已提及，程序中没有DIM语句，机器自动认为下标变量

的每个下标值只能从0到10，换言之，最大下标值为10。这在实际应用中往往是不够用的。比如200个学生的200个总分，用一个数组就不行。而在另外一些时候，当着下标值用不到10时，机器仍按每个下标从0到10留出位置，也有些浪费。

语句DIM就可以解决这个问题。

DIM的语句格式是：

〔行号〕DIM 数组名 下标表〔{，数组名 下标表}〕

下标表的格式又是：

（最大下标〔{，最大下标}〕）

下标变量表中的第一个最大下标值，定义该下标变量第一个下标可取的最大值；…第N个最大下标值，定义该下标变量的第N个下标可取的最大值。

最大下标值的构成规则同下标的构成规则一样，只要不是负数或字符串型量就可以。最大下标值的取值范围没有明确的数值规定，只要内存中有富裕空间即可。

程序在运行时如果遇到DIM语句，则将按语句给出的数组名在数组表中建立一个表头，并以DIM指定的最大下标值留出一定的空间来（这时数组表中每行最后一个元素的号码，最后一行元素的号码，…，就不一定是10了，它们取决于DIM给出的最大下标值）。DIM还会将该数组所有元素初始化为0(或空串)。

使用数组说明语句DIM时，要注意几个问题：

1.任何一个数组，只允许被说明一次。这是解释程序规定的。目的是防止多次说明造成混乱。

2.任何一个数组，如果有说明的话，必须先说明，后使用。

事实上，对于任何一个数组，若是没有用DIM语句说明，实际上在第一次使用时已隐含地执行了一次DIM语句，只不过其每个最大下标都取10罢了。换言之，它也遵循先说明后使用的原则。

对这种第一次使用时已隐含地做了一次DIM说明，再以显式给出说明时，实际上等于二次说明，同样违背第一条规则。

3.由于最大下标值可以用算术表达式给出，一定要防止出现负下标值的情况。

§3. STORE与RECALL

STORE与RECALL是专用于数型数组与磁带交换信息的。它们的格式如下：

〔行号〕 STORE　　数组名

〔行号〕 RECALL　数组名

STORE命令，可以把指定的数组的所有元素存入磁带上。而RECALL则可以把磁带上的数组元素读入指定的数组中。

STORE要在指定的数组取得数据之后使用，否则写入磁带的全是0。

RECALL使用的数组名可以不同于STORE 使用的数组名。

在使用STORE或RECALL之前，应对各自的数组用DIM说明过。读入数组的元素总个数不得少于存于磁带时数组元素总个数。读入数组的最大下标值，除最后一个最大下标值可以增大外，其余最大下标值只要同存带时那个数组的最大下标值不同，读入的数组元素将产生混乱。读入数组的维数与写入数组的维数不同，也会产生混乱。换言之，RECALL用的数组可以比STORE 用的数组增大维数或增大最后一个最大下标值。在这种情况下执行 RECALL，读入的数组元素与STORE写到带上那个数组元素一样。读入的数组元素多于写带的数组元素时，多余的元素补0。这时可能出现提示ERR，但不影响程序的运行及对数组的使用。

§4. 应用程序举例

例1. 约瑟夫斯（Josephus）问题。有N个人拉成一个圆圈。从其中某人开始，按一个方向顺序编上号码。然后从第一个人开始数起，每数到M时，就把这个人从圆圈上拉出来，再从下一个人数起，再数到M时，那个人也被从圆圈上拉出来。如此继续下去，直到全部从圆圈上拉完。试写一个程序，按从圆圈上拉出来的顺序打印出他们的编号来。

解法1： 用数组A的各个下标来代表圆圈中的顺序位置，而数组各元素的值则代表相应位置上那个人的编号。变量N代表圆圈上的人数，变量M则代表数至第几个人时拉一个人出来。每当一个人被拉出时，圈中人数(N)减1，圆圈中出现的空位由编号较大的人递补。

程序如下：

```
10  INPUT  "N=?";  N
20  INPUT  "M=?";  M
30  DIM  A(N)
40  FOR  I=1  TO N
50  A(I)=I
60  NEXT
70  S=0 : R=0
80  S=S+1 : R=R+1
90  IF  R>N  THEN  R=1
100 IF  S<M  THEN  80
110 PRINT  A(R),
120 IF R=N  THEN  R=1 : GOTO  160
130 FOR  I=R  TO  N-1
140 A(I)=A(I+1)
```

```
150 NEXT
160 N=N-1
170 IF  N>1  THEN  S=1 : GOTO  80
180 PRINT  A(N)
190 END
```

当取N为16，M＝4时，输出为：

```
4               8               12
16              5               10
15              6               13
3               14              9
7               11              2
1
```

当N取13，M取5时，输出为：

```
5               10              2
8               1               9
4               13              12
3               7               11
6
```

程序的10和20行，分别从键盘上取得N和M的值。30行是数组说明语句。

40行至60行，为各位置上的人编号。

80行至100行控制从圆圈上某人数起，直到数至第M个人；110行把这个人的编号打印出来；90行用来保证圆圈上编号最大的那个人（第N个人）的下一个人为第1个位置上的人。

120行至170行控制把每次数到的第M个人拉出圆圈外。比这个人编号大的，依序向前错位。其中120行是对应一种特殊的情况：当被拉出去的人为圆圈上编号最大者时，不必错位。

180行指明当圆圈上只剩最后一个人时，就不要再从1数到M了，直接打印出来就行。

解法2：数组A的各下标，变量N、M均同上程序的意义。但各元素的值不代表该位置上那个人的编号，而代表下一个人的位置号。好象每个人都记着下一个人的位置号码一样。每当数到M时，将这个位置号输出，表示这个位号上的人被拉了出去。这时这个被拉出去的人应把他记着的那个位置号转交给前面那个没有被拉出的人。这又可以比作：若干人面向中间拉成一个圈。当其中第3个人被拉出圈时，他应把左手拉着的第4个人的右手转交给右手拉着的第2人的左手。

程序中用变量U来统计被拉去的人数；变量S用来数数；R表示被拉出去的位置号，变量T记改变前的R。

程序如下：

```
10  INPUT "N=?";  N
20  INPUT "M=?";  M
30  DIMA (N)
40  FOR  I=1  TO  N-1
50  A(I)=I+1
60  NEXT
70  A(N)=1
80  S=0 : U=0 : R=N
90  S=S+1 : T=R : R=A(R)
100 IF  S<M  THEN  90
110 PRINT  R,
120 U=U+1 : IF U=N  THEN  END
130 A(T)=A(R)
140 S=0 : GOTO  90
```

这个程序和上一个程序执行结果是一样的。

例2．学生成绩统计

设某班有学生N名参加考试，要求写出一个程序，统计各分数段的人数，并求全班考分的平均值、变异数与标准差。

设学生的考分依次存于数组A中：那么

$$\text{平均差 } M=\frac{1}{N}\left[A(1)+A(2)+\cdots+A(N)\right]$$

$$=\frac{1}{N}\sum_{I=1}^{N}A(I)$$

$$\text{变异数 } V=\frac{1}{N-1}\left[\sum_{I=1}^{N}A(I)^2-2M\sum_{I=1}^{N}A(I)+NM^2\right]$$

$$=\frac{1}{N-1}\left[\sum_{I=1}^{N}A(I)^2-NM^2\right]$$

$$\text{标准差 } S=\sqrt{V}$$

程序如下（设该班学生为20名。学生人数及考分都列在DATA中。要求分成6个分数段）：

```
10 READ N
20 DIM A(N), A$(5), B(5)
30 FOR I=1 TO N
40 READ A(I)
50 K=INT (A(I)/10)+1
60 ON K GOSUB 90,90,90,90,90,100,110,120,130, 140,
   140
70 IF K<1 OR K>11 THEN PRINT "DATA ERROR"
   : STOP
80 GOTO 150
90 B(0)=B(0)+1 : RETURN
100 B(1)=B(1)+1 : RETURN
110 B(2)=B(2)+1 : RETURN
120 B(3)=B(3)+1 : RETURN
130 B(4)=B(4)+1 : RETURN
140 B(5)=B(5)+1 : RETURN
150 NEXT
160 FOR I=0 TO 5
```

```
170 READ A$(I)
180 NEXT
190 PRINT "**SCORE  INTERVAL  LIST**"
200 FOR  I=5  TO  0  STEP  -1
210 PRINT  A$(I); ":";  B(I)
220 NEXT
230 M=0 : A=0
240 FOR  I=1  TO  N
250 M=M+A(I)
260 A=A+A(I)*A(I)
270 NEXT
280 M=M/N
290 V=(A-N*M↑2)/(N-1)
300 S=SQR  (V)
310 PRINT : PRINT
320 PRINT "**  STATISTIC**"
330 PRINT "NUMBER  OF  SAMPLE  :"; N
340 PRINT "                MEAN  :"; M
350 PRINT "                      VARIANCE  :"; V
360 PRINT "STANDARD          DEVIATION  :"; S
370 DATA  20
380 DATA  80, 90, 85, 70, 100, 0, 10, 75, 95, 70
390 DATA  75, 79, 89, 60, 49, 5, 85, 80, 55, 60
400 DATA "0-49", "50-59", "60-69","70-79", "80-89",
    "90-100"
410 END
```

这个程序运行输出是：

```
 **SCORE  INTERVAL  LIST**
90-100:3
80-89:5
```

```
70－79:5
60－69:2
50－59:1
0－49:4
＊＊ STATISTIC ＊＊
NUMBER OF SAMPLE :20
MEAN :65. 6
VARIANCE :856. 35789
STANDARD DEVIATION :29. 2635933
```

这个程序大体上分两段：

220行前为一段，读入分数，划分分数段，并打印出来。

230行以后为求均值、变异数与标准误差，并打印出来。

例3. 排序问题

排序问题在数据（信息）处理中，占有很重要的地位。它的目的主要是根据某种预先规定的准则，把数据按一定的顺序排列起来，以便将来使用时能尽快地找到它。

在日常遇到的问题中，也常有排序问题。比如学生的名次，要通过考分的大小顺序，才能排出；运动员名次，也要通过得分的大小顺序排出；一本字典，无论是四角号码法，也无论是部首偏旁法，或者是音序法，总是按一定顺序排出的。

排序的方法很多。比较常用的有冒泡法，最小元素法等。

所谓冒泡法，是这样一种方法。对于一组数据，我们可以设想它们从上到下排成一个竖直串（象冰糖葫芦一样穿在一起），第一个数据在最上面，最后一个数据在最下面。先拿第一个数据与第二个数据比较，如果小的在上而大的在下，就拿第二个数据和第三个数据相比较。否则，当第二个数据小于第一个数据时，这两个数据交换位置（第二个数据变成第一个数据，第一个数据变成第二个数据），这时再作第二个数与第三个数据的比较。如

果小的在上而大的在下，则取第三个数据和第四个数据作比较。当着第三个数据比第二个数据小时，这两个数据交换位置。也就是使第三个数据变成了第二个数据。由于第二个数据是刚刚调过来的，它还得与第一个数据比较，当着第二个数据又小于第一个数据时，这两个数据还得交换。这样一次次比较下去，一旦发现相邻两个数据大的在上而小的在下时，就两两换位。这样好象水中的气泡一样，一个最小的数据总会冒到顶端。最后这组数据中任何相邻两个总是小的在上而大的在下，这时排序就告结束。

最小元素法和冒泡法实质上很类似，它是这样完成的：先取第一个数据作基准，依序逐个检查其后的每个数据是否有比它小的，一旦发现，则将此小的数据调换到第一个数据的位置上作为新的基准，原来那个基准（即原第一个数据）调换到刚发现的那个小数据原来占据的位置（也就是新基准与旧基准对调位置）。然后继续向后找有无与比新基准小的数据，如有，再把它当作新基准与刚才的新基准对换位置。如此直到比较完所有数据。这时，所有数据中最小的数据便调换到第一个数据的位置上了。然后再以第二个数据作为基准，用上面的方法把次小的数据调换到第二个数据的位置上。如此，第三，第四…个数据都做这样的处理，最后这组数据就从小到大顺序地排好了。

这里我们想介绍一下对最小元素法的改进方案。从刚才的叙述中可以看到，在把所有数据中最小的一个调到第一个数据位置的过程中，有可能要做多次对调，这是要花时间的。改进的办法是：先仍以第一个数据作基准，当发现有一个数据比基准小时，不作数据的对调，而改用那个小数据作基准。后边再发现比基准小的数据，再用这个新的小数据作基准，直到比较完最后一个数据。这时这个最后的基准便是所有数据中的最小者。这样将第一个数据与它对调就行了。找次小数，次次小数据、均可如此处理。

上述的描述中，我们都用数据这一概念，它表明数据可以是

语言允许的各种类型数据。

还需指出的是，我们上边排好的数据，都是小的在前，大的在后。如果希望大的在前，小的在后，怎么办？完全可以用上面介绍的方法，把叙述中的“小”字和“大”字对调一下就行了。

最后还得说明一点，关于两个数据的对调问题，要采取一点点措施。比如，希望变量A（假定其值为5)与变量 B（假定其值为10）对调。如果你使用两个赋值语句A＝B和 B＝A，那是错误的。因为你执行完第一个赋值语句时A的值变为10，而B的值还是10。再执行第二个赋值语句，使B 得到的还是10。这样的结果不是交换A、B的取值，而是使变量A 和变量B均得到了10。这是与原意违背的。

要想使两个变量的取值对调，必须通过第三个变量的辅助，它实际上起暂存的作用。如果还是做上面的 A、B 对换，这时设一个变量C，然后通过下面三个赋值语句：

```
C＝A ： A＝B ： B＝C
```

执行第一个语句时，C得到5；执行第二个语句时，A得了10；执行第三个语句时，B得到了5。从而实现了变量A与B 的对调。

现在可以用排序的方法来编写程序了。

假定有4个学生，考5门课，共20 个分数（列在下面程序 的 DATA 语句中，学生的姓名也被列入)。要求按平均分排出他们的名次。

程序如下：

```
10  READ  N,M
20  DIM  A$(N),A(N,M),B(N)
30  FOR  I＝1  TO  N
40  READ  A$(I)
50  FOR  J＝1  TO  M
60  READ  A(I,J)
```

```
70  A(I,0)=A(I,0)+A(I,J)
80  NEXT  J
90  B(I)=A(I,0)/M
100 NEXT  I
110 FOR  I=1  TO  N-1
120 K=I
130 FOR  J=I  TO  N-1
140 IF  B(J+1)>B(K)  THEN  K=J+1
150 NEXT  J
160 T$=A$(I) : A$(I)=A$(K) : A$(K)=T$
170 T=B(I) : B(I)=B(K) : B(K)=T
180 NEXT  I
190 FOR  I=1  TO  N
200 PRINT  A$(I),B(I)
210 NEXT
300 DATA  4,5
310 DATA "ZHANG  SAN", 78, 86, 84, 90, 68
320 DATA "LI  SI", 57, 76, 69, 47, 63
330 DATA "WAZG  WU",100, 98, 89, 92, 95
340 DATA "ZHAO  LU", 88, 93, 79, 84, 85
350 END
```

这个程序运行时输出

```
WAZG  WU          94.8
ZHAO  LU          85.8
ZHANG  SAN        79.2
LI  SI            62.4
```

程序中A数组的第0行元素为某生的总分。平均分放在B数组中。

例4. 数据（信息）的检索（查找）

数据检索在计算机应用中的作用更为重要。它可以从计算机

存贮的大量数据中迅速而准确地查找出某个需要的信息。

检索的方法也很多，一个好的检索程序可以大大节省检索时间。

常用的比较容易理解的一种检索方法叫线性查找法。它是把信息从头到尾全部扫描一遍，从中找出所要查找的内容。这种方法的优点在于适应性强，无论信息按什么顺序排放，都可以查找出来。缺点是查找速度慢。不过，由于计算机本身的运算速度越来越快，这个缺点也就不那么重要了。

对于已经排好大小顺序的信息，通常也常用对分查找法。所谓对分查找，很类似于对分区间套求高次方程的根的那种对分方法。由于数据(信息)已按大小顺序排好，那么处在全部数据区端点的两个数据，必有一个最大的，另一个是最小的。此时将全部数据从中间对分，即找出最中间一个数据，检查它比你要找的数据大还是小，从而确定你所查找的信息是在比中间数据大的那半组数据里，还是在比中间数据小的那半组数据里。这时再把你的数据所在的那半组数据作对分比较。不断的对分比较就可以找到一个可能包括你所查找的信息的更小的范围。注意，这种对分过程得有一个终点，因为数据是一个个独立的实体，不能把一个数据再对分开来（这就不能象对分区间套那样，对分到最小的区间小于给定的ε)。当对分到一定时候，最后那个小范围内还可能有一批数据（数据要量尽可能少)，这时再在这个小范围内用线性查找方法。

需要指出的是，有时你所查找的信息可能找不到。通常对这个情况应给出提示说明。

现在我们用线性查找法来编写一个应用程序。

设有学生若干人（可以是一个学校的全体学生——只要内存放得下的话，也可以是一个年级的、一个班的或一个小组的学生)，用DATA语句列出他们的姓名、性别、籍贯和考试分数，通过一段

程序的运行，可以把这些数据存放在一个或几个数组中。检索的任务就是有目的地找出其中某个人或某些人。比如可以给出一个姓名，从中查出这个人的情况；也可以查找所有男同学（或女同学）的情况；还可以查找某个分数段范围的人；当然也应能找出籍贯为某地的人等等。程序写的再复杂点，甚至可以找出符合某几种情况的人来（比如考分在90分以上的、籍贯为北京的所有男同学等）。

下面列出了这个程序。在DATA语句中只示意性的列出了几个人的情况。程序中变量N表示人数，变量J表示你要查找那一项（查姓名或性别或考分等），给出0时表明查找项结束（好象说：就查找符合刚刚选择的那几项条件的人），N和J均由键盘输入。数组A$存放所有人的可以用串型量表示的数据（如姓名、籍贯等），数组B则存放所有人的可以用数型量来表示的数据。变量K用来记录适合某种情况的人数，数组C则记录了这些人的序号。

```
10  INPUT "N=?";  N
20  DIM  A$(N,3),B(N),C(N)
30  FOR  I=1  TO  N
40  FOR  J=1  TO  3
50  READ  A$(I,J)
60  NEXT  J
70  READ  B(I)
80  NEXT  I
90  FOR  I=1  TO  N
100 C(I)=I
110 NEXT
120 K=N : L=0
130 INPUT "0:END,1:NAME,2:SEX,3:NATIVE PLACE,
    4:TESTING  SCORE";  J
140 ON  J  GOSUB  300,400,500,600
```

```
150 IF J < > 0 THEN 130
160 FOR I=1 TO K
170 FOR J=1 TO 3
180 PRINT A$(C(I),J), " ";
190 NEXT J
200 PRINT B(C(I))
210 NEXT I
220 INPUT "YES OR NO?"; A$
230 IF A$= "YES" THEN 90
240 PRINT "BYE! "
250 END
300 INPUT "NAME:"; A$
310 GOSUB 700
320 RETURN
400 INPUT "SEX:"; A$
410 GOSUB 700
420 RETURN
500 INPUT "NATIVE PLACE"; A$
510 GOSUB 700
520 RETURN
600 INPUT "TESTING SCORE"; A$
610 IF A>B THEN T=A : A=B : B=T
620 FOR I=1 TO K
630 IF B(C(I))<A OR B(C(I))>B THEN 660
640 C(L+1)=C(I)
650 L=L+1
660 NEXT
670 K=L : L=0
680 RETURN
700 FOR I=1 TO K
```

```
710 IF A$(C(I),J)< > A$ THEN 740
720 C(L+1)=C(I)
730 L=L+1
740 NEXT I
750 K=L : L=0
760 RETURN
800 DATA "CHANG SHENG", "MALE","BEIJI NG",98.2
810 DATA "LI YUE MING", "MALE", "TIAN JIN",
    89.5
820 DATA "LIN TAI XING", "FEMALE",. "BEIJING",
    85.3
830 DATA "WEN WEI", "FEMALE", "SHANGHAI", 95.9
```

程序中对数组C的使用，是一个小小窍门，读者可以细心体察它。

此外，由于每个学生的考分在这个程序中没有任何算术运算过程，只是用来作比较运算，因此，把考分当作字符串型量也是允许的。这样，可以把学生的所有数据存入一个数组中（如仍用A$数组），不另设B数组也行。

例5. 查字典程序

由于字典编排是按顺序的，所有字可以看作已排好序的数据，因而可使用对分查找法查找。

我们拿查《新英汉大词典》为例。把每页的头一个词依序存入某一数组，它的顺序号也就是页号。下面的程序可以用于查找出指定的单词应在那一页上。

由于词典共有1648页，我们对分的次数，再多也无需超过10次。因为$2^{10}=1024$，而$2^{11}=2048$，对分11次，将没有任何意义。

程序如下（对分9次）：

```
10 DIM A$(1648)
20 FOR I=1 TO 1648
```

```
30  READ  A$(I)
40  NEXT
50  H=1648/2 : K=H
60  INPUT  B$
70  FOR  I=1  TO 9
80  K=INT (K/2)+1
90  IF  B$=A$(H)  THEN  PRINT  H : END
100 IF  B$>A$(H)  THEN  H=H+K:GOTO 120
110 H=H-K
120 NEXT  I
130 FOR  I=H-K  TO  H+K
140 IF  B$=A$(I)  THEN  PRINT  I : END
150 IF  B$>A$(I)  AND  B$<A$(I+1)  THEN  170
160 NEXT
170 PRINT  I
180 END
200 DATA  A, ABASE, …
```

程序的10—40行，用来将每页的头一个词读入数组A$ 中。50行为H和K取初值。60行输入欲查找的词。70—120行为对分查找，130—160行为最后一个小范围内的线性查找。K为对分后区间的长度，H为对分点（即页码）。80行取整动作，是因为页码总是整数。由于取整对正数而言是截舍小数部分，容易使130行线性查找时范围变小产生漏页，故80行的算术表达式增加一项加1处理。

DTAT 语句中的数据也只列了两个作为示意。

例6. 线性方程组求解

设有线性方程组

$$\begin{cases} 3X+2Y-7Z=3 & ① \\ 2X-Y+2Z=6 & ② \\ 5X-3Y-3Z=-3 & ③ \end{cases}$$

可用下列步骤求出X，Y，Z的解：

(1) ①÷3 得：

$$X+\frac{2}{3}Y-\frac{7}{3}Z=1 \qquad ④$$

(2) ②÷2 得：

$$X-\frac{1}{2}Y+Z=3 \qquad ⑤$$

于是，④－⑤ 得：

$$\frac{7}{6}Y-\frac{10}{3}Z=-2 \qquad ⑥$$

(3) ③÷5得：

$$X-\frac{3}{5}Y-\frac{3}{5}Z=-\frac{3}{5} \qquad ⑦$$

这样，④－⑦得：

$$\frac{19}{15}Y-\frac{26}{15}Z=\frac{24}{15} \qquad ⑧$$

(4) ⑥×$\frac{6}{7}$得：

$$Y-\frac{20}{7}Z=-\frac{12}{7} \qquad ⑨$$

而 ⑧×$\frac{15}{19}$得：

$$Y-\frac{26}{19}Z=\frac{24}{19} \qquad ⑩$$

(5) ⑩－⑨得：

$$\frac{198}{7\times19}Z=\frac{396}{7\times19} \qquad ⑪$$

(6) ⑪÷$\frac{198}{7\times19}$得：

$Z=2$ ⑫

(7) 将⑫代入⑨：

$Y=4$ ⑬

(8) 将⑫、⑬代入④：

$X=3$ ⑭

于是得解：$X=3$，$Y=4$，$Z=2$

上面第(1)至(6)步为消元过程，⑦、⑧步则为回代过程。

我们把①、②、③式的X、Y、Z的系数置于数组A中，把等号右端的常数项置于数组B中，可构成如下两个数组：

	X系数	Y系数	Z系数	常数
	↓	↓	↓	↓
①式→	3	2	−7	3
②式→	2	−1	2	6
③式→	5	−3	−3	−3
		数组A		数组B

在消元过程中，第(1)步做完后，两个数组变成了：

④式→ $\begin{pmatrix} 1 & \frac{2}{3} & -\frac{7}{3} \\ 2 & -1 & 2 \\ 5 & -3 & -3 \end{pmatrix} \quad \begin{pmatrix} 1 \\ 6 \\ -3 \end{pmatrix}$

第(2)步做完后：

⑥式→ $\begin{pmatrix} 1 & \frac{2}{3} & -\frac{7}{3} \\ 0 & \frac{7}{6} & -\frac{10}{3} \\ 5 & -3 & -3 \end{pmatrix} \quad \begin{pmatrix} 1 \\ -2 \\ -3 \end{pmatrix}$

第(3)步做完后：

⑧式→
$$\left[\begin{array}{ccc|c} 1 & \frac{2}{3} & -\frac{7}{3} & 1 \\ 0 & \frac{7}{6} & -\frac{10}{3} & -2 \\ 0 & \frac{19}{15} & -\frac{26}{15} & \frac{24}{15} \end{array}\right]$$

第(4)步做完后：

⑨式→
$$\left[\begin{array}{ccc|c} 1 & \frac{2}{3} & -\frac{7}{3} & 1 \\ 0 & 1 & -\frac{20}{7} & -\frac{12}{7} \\ 0 & \frac{19}{15} & -\frac{26}{15} & \frac{24}{15} \end{array}\right]$$

第(5)步做完后：

⑪式→
$$\left[\begin{array}{ccc|c} 1 & \frac{2}{3} & -\frac{7}{3} & 1 \\ 0 & 1 & -\frac{20}{7} & -\frac{12}{7} \\ 0 & 0 & \frac{198}{7\times 19} & \frac{396}{7\times 19} \end{array}\right]$$

第(6)步做完后：

⑫式→
$$\left[\begin{array}{ccc|c} 1 & \frac{2}{3} & -\frac{7}{3} & 1 \\ 0 & 1 & -\frac{20}{7} & -\frac{12}{7} \\ 0 & 0 & 1 & 2 \end{array}\right]$$

第(7)步做完后：

⑬式→
$$\left[\begin{array}{ccc|c} 1 & \frac{2}{3} & -\frac{7}{3} & 1 \\ 0 & 1 & 0 & 4 \\ 0 & 0 & 1 & 2 \end{array}\right]$$

做完第(8)步：

⑭步→ $\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 3 \\ 4 \\ 2 \end{pmatrix}$

可见，消元过程实际上就是使由方程组系数构成的方阵从左上角到右下角这条对角线上的元素变为1，并使左下三角阵中各元素为0。而回代过程又使右上三角阵各元素为0。

在上述消元过程中，任何一次使A(I, I)元素变为1时，总是通过A(I, I) 除以A(I, I) 实现的。如第(1)、(4)、(6)步。这就出现了一个问题：当着A(I, I)很小（甚至为0）时，用A(I, I) 作为除数是很危险的，因为可能出现溢出（或不能除），这是应予避免的。方法之一是：当着想使A(I, I) 变为1时，从I行开始到最后一行，找出I列上绝对值最大的元素所在的行，把这一行与I行对调，同时对常数项数组也作相应的对调（这实际上相当于把方程组的两个方程调换位置）。这时处在A(I, I) 位置上的元素就可以大胆地作为除数来使用。如果找到的绝对值最大的元素仍然为0或接近于0，那只能说明方程无解或具有不定解。这种方法是在某一列上找绝对值最大元素的，故称为列主元消去法。

类似地还有行主元消去法和全主元消去法。

行主元消去法是在使A(I, I) 元素变为1时，从I列至最后一列，找出I行上绝对值最大的元素所在的列，把这一列的所有元素与I列元素对调。这样做实际上相当于把方程组中某一元所在的位置与另一元所在的位置对调，因此，要用一个数组记载这种对调情况，以免得到的解张冠李戴。

全主元消去法是综合了上述两种方法得出的。它是在使A(I, I) 变1时，从I行到最后一行，从I列到最后一列这样一个方阵中，找出绝对值最大的元素，然后作行与行对调（别忘记常数项数组也对调）、列与列对调（别忘了记载对调情况），使这个绝对值最大的元素能换到A(I, I) 位置上来。

下面一个程序是列主元消去法解线性方程组的程序。变量N表示方程的个数，变量E表明任何一个绝对值最大的元素比它小时，认为方程无解。数组A存放各方程的系数，数组B开始存放各方程的常数项，最后放方程的解。

作为一个例子，求下列线性方程组的解：

$$X_1 - 4X_2 + 6X_3 - 5X_4 = 8$$
$$-5X_1 + 21X_2 - 33X_3 + 32X_4 = -32$$
$$6X_1 - 26X_2 + 43X_3 - 48X_4 = 25$$
$$5X_1 - 24X_2 + 45X_3 - 64X_4 = -10$$

程序如下：

```
10 INPUT "N=?"; N
20 INPUT "E=?"; E
30 DIM A(N,N),B(N)
40 FOR I=1 TO N
50 FOR J=1 TO N
60 READ A(I,J)
70 NEXT J,I
80 FOR I=1 TO N
90 READ B(I)
100 NEXT
110 FOR K=1 TO N
120 C=A(K,K):I=K
130 IF K=N THEN 170
140 FOR J=K+1 TO N
150 IF ABS (A(J,K))>ABS (C) THEN C=A(J,K):I=J
160 NEXT J
170 IF ABS (C)<E THEN STOP
180 IF I=K THEN 230
190 FOR J=K TO N
```

```
200 T=A(K,J) : A(K,J)=A(I,J) : A(I,J)=T
210 NEXT J
220 T=B(K) : B(K)=B(I) : B(I)=T
230 C=1/C
240 FOR J=K TO N
250 A(K, J)=A(K,J)*C
260 NEXT J
270 B(K)=B(K)*C
280 IF K=N THEN 370
290 FOR I=K+1 TO N
300 T=A(I,K)
310 FOR J=K TO N
320 A(I,J)=A(I,J)-T*A(K,J)
330 NEXT J
340 B(I)=B(I)-T*B(K)
350 NEXT I
360 NEXT K
370 FOR I=N-1 TO 1 STEP -1
380 FOR J=I+1 TO N
390 B(I)=B(I)-A(I, J)*B(J)
400 NEXT J, I
410 FOR I=1 TO N
420 PRINT "X("; I; ")="; B(I)
430 NEXT
500 DATA 1, -4, 6, -5, -5, 21, -33, 32
510 DATA 6, -26, 43, -48, 5, -24, 45, -64
520 DATA 8, -32, 25, -10
530 END
```

运行情况：

```
RUN
```

```
N=? 4
E=? 1E-10
X(1)= .999999214
X(2)= 1.99999978
X(3)= 5
X(4)= 3.00000004
```

程序的10—100行用来读入数据（系数与常数项）；120—160行用于找列主元素；190—220行作行交换；230—270行变对角上的元素为 1;290—350 行置左下三角阵为0;370—400行为回代过程；410—440行打印输出。

这个程序实际上是个通用程序，只要改变DATA中的数据，就可以解任何线性方程组。

例7. 矩阵运算

二维数组用于矩阵运算是很方便的。

这里举出两个矩阵相乘的例子。

设A为N行L列矩阵，B为L行M列矩阵，A矩阵乘以 B矩 阵所得的C矩阵为N行M列，其I行J列元素的值为A矩阵I行各元素与 B矩阵J列各元素分别相乘之和。

设

$$A=\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1l} \\ a_{21} & a_{22} & \cdots & a_{2l} \\ \cdots & \cdots & \cdots & \cdots \\ a_{n1} & a_{n2} & \cdots & a_{nl} \end{pmatrix} \quad B=\begin{pmatrix} b_{11} & b_{12} & \cdots & b_{1m} \\ b_{21} & b_{22} & \cdots & b_{2m} \\ \cdots & \cdots & \cdots & \cdots \\ b_{l1} & b_{l2} & \cdots & b_{lm} \end{pmatrix}$$

令 $C=A\times B$

则

$$C=\begin{pmatrix} c_{11} & c_{12} & \cdots & c_{1m} \\ c_{21} & c_{22} & \cdots & c_{2m} \\ \cdots & \cdots & \cdots & \cdots \\ c_{n1} & c_{n2} & \cdots & c_{nm} \end{pmatrix}$$

其中$c_i = a_{i1} \cdot b_{1j} + a_{i2} \cdot b_{2j} + \cdots + a_{il} \cdot b_{lj}$

作为例子我们设：

$$A = \begin{pmatrix} 1 & 0 & 3 & -1 \\ 2 & 1 & 0 & 2 \end{pmatrix}$$

$$B = \begin{pmatrix} 4 & 1 & 0 \\ -1 & 1 & 3 \\ 2 & 0 & 1 \\ 1 & 3 & 4 \end{pmatrix}$$

求C = A * B

程序如下：

```
10  INPUT N, L, M
20  DIM A(N, L), B(L, M), C(N, M)
30  FOR I=1 TO N
40  FOR J=1 TO L
50  READ A(I, J)
60  NEXT J, I
70  FOR I=1 TO L
80  FOR J=1 TO M
90  READ B(I, J)
100 NEXT J, I
110 FOR I=1 TO N
120 FOR K=1 TO M
130 C(I, K)=0
140 FOR J=1 TO L
150 C(I, K)=C(I, K)+A(I, J)*B(J, K)
160 NEXT J, K, I
170 FOR I=1 TO N
180 FOR J=1 TO M
190 PRINT C(I, J),
200 NEXT J
```

```
210 PRINT
220 NEXT I
300 DATA 1, 0, 3, -1, 2, 1, 0, 2
310 DATA 4, 1, 0, -1, 1, 3, 2, 0, 1, 1, 3, 4
320 END
```

运行后输出

```
 9        -2        -1
 9         9         11
```

这也是一个通用程序，只要修改DATA中的数据，可以求任意两个矩阵相乘积。

§5. 与字符串有关的几个函数

在上节例4中，我们曾提到，可以把学生的考分与姓名、籍贯等放在同一数组中，这时原来作为数型量的考分被当作字符串型量来处理，从而失去了算术运算的功能。因为“12”+“3”的结果是123，而不是15。

此外，还用那程序例来说，如果我们要找出所有某姓的学生，或者我们想查找只知其名不知其姓的学生，查找起来就困难了。

下面我们介绍的有关字符串的一些函数，对于解决上述一些问题提供了有效的手段。当然这些函数的用途远远不止这些。

一、测字符串长度的函数(LEN函数)

LEN函数格式：

LEN（字符串表达式）

这个函数将给出指定字符串的长度（即字符个数）。其数值范围只能在0～255的范围内（因为键盘缓冲区的限制，我们无法敲入一个大于255个字符的字符串）。

例如程序

```
10 A$ = "ABC"
20 B$ = "1234"
30 C$ = A$ + B$
40 PRINT LEN (A$),
50 PRINT LEN ("ABCDEF"),
60 PRINT LEN (C$),
70 PRINT LEN (A$ + "D")
80 END
RUN
3             6             7
4
```

二、STR$函数

STR$函数格式是：

STR$(算术表达式)

这个函数的功能是把给定的算术表达式的值转换成相应的字符串。

例如程序：

```
10 A = 2354
20 B = 46
30 PRINT STR$ (A)
40 PRINT STR$ (A + B)
50 PRINT STR$ (A) + STR$(B)
60 PRINT STR$ (10000000000)
70 END
RUN
2354
2400
235446
1E + 10
```

注意，STR$函数的值为字符串型量。从50行可明显地看出。

三、VAL函数

VAL函数格式是：

VAL（字符串表达式）

这一函数的功能是把给定的字符串型量转换为相应的数型量。

这个函数的本来用意是想把用引号对括起来的数型量再恢复为原来的数型量。因此，如果引号对中不是数型量时，就不应进行这种转换。因此特规定：字符串的第一个非空字符不是数字时，函数取0值，否则从左到右逐字符扫描给定的字符串，直到扫描完为止，或遇到数型量不允许的字符（注意：字母E是数型量允许的）为止。然后把扫描过的字符串转换为相应的数型量。

例如程序：

```
10 A$ = "123"
20 PRINT VAL (A$)
30 PRINT VAL (A$ + "55")
40 PRINT VAL ("ABC 12")
50 PRINT VAL ("12AB")
60 PRINT VAL (A$) + VAL ("55")
70 END
RUN
123
12355
0
12
178
```

四、ASC码函数(ASC函数)

ASC函数的格式是：

ASC（字符串表达式）

这个函数用来求出给定字符串中首字符的ASCII 码（用十进

制数表示)。

例如程序:

```
10 A$ = "AB"
20 PRINT ASC (A$)
30 PRINT ASC ("123")
40 END
RUN
65
49
```

五、CHR $ 函数

CHR $ 函数的格式是:

CHR $ (算术表达式)

这一函数的结果是产生一个以给定的数值为其ASCII 编码的字符。

例如程序:

```
10 PRINT CHR$ (65)
20 PRINT CHR$ (49)
30 END
RUN
A
1
```

因此,下列程序可以产生出全部ASCII 字符(其中有一部分是不可显示字符):

```
10 FOR I = 0 TO 255
20 PRINT CHR$ (I);
30 NEXT
40 END
```

运行这个程序会听到"嘟"声,是PRINT CHR $ (7) 产生的。在程序中适当使用这一语句会发出"嘟"声。CHS $ (7) 实

际上是字符CTRL—G。

建议，运行这一程序时请不要打开打印机。因为有几个ASCII字符会使打印机产生一些不必要的动作。

六、LEFT $ 函数

LEFT $ 函数的格式是：

LEFT $ (字符串表达式，算术表达式)

设算术表达式的值为N，这个函数产生这样一个子字符串：它由给定字符串的最左边的N个字符所组成。这时给定的字符串又称为源字符串。

算术表达式的值N，应在1—255的范围内。当N的值大于源字符串长度时，函数的值就是源字符串。

例如程序：

```
10 A$ = "ABCDEFGH"
20 PRINT LEFT$ (A$, 5)
30 PRINT LEFT$ (A$ + "1234", 10)
40 PRINT LEFT$ (A$, 10)
50 END
RUN
ABCDE
ABCDEFGH12
ABCDEFGH
```

七、RIGHT $ 函数

RIGHT $ 函数的格式是：

RIGHT $ (字符串表达式，算术表达式)

设算术表达式的值为N，这个函数将产生这样一个子字符串：它由源字符串的最右边N个字符所组成。

算术表达式的值应在1—255的范围内。如果这个值大于源字符串长度时，函数的结果就是源字符串。

例如程序：

```
10 A$ = "ABCDEFGH"
20 PRINT RIGHT$ (A$, 5)
30 PRINT RIGHT$ (A$ + "1234", 10)
40 PRINT RIGHT$ (A$, 10)
50 END
RUN
DEFGH
CDEFGH1234
ABCDEFGH
```

八、MID＄函数

MID＄函数格式是：

MID＄（字符串表达式，算术表达式1，算术表达式2）

设算术表达式1的值为N，算术表达式2的值为M。这个函数将产生这样一个子字符串：它由源字符串中的第N个字符开始取M个字符所组成。

N和M的值，必须在1—255的范围内。当N大于源字符串的长度时，本函数将产生一个空串；当N＋M大于源字符串长度或大于255时，所产生的子串由源字符串的第N个字符开始到源字符串的最后一个字符所组成。

例如程序：

```
10 A$ = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
20 PRINT MID$ (A$, 5, 10)
30 PRINT MID$ (A$ + "12345", 20, 25)
40 END
RUN
EFGHIJKLMN
TUVWXYZ12345
```

以上我们只介绍这些函数的意义。它们的使用并不困难，但

你要经常意识到它们的存在，并熟习它们的用途，以便顺手拿来帮助你解决问题。

比如，就上节例4来说，假定我们要查找所有姓张的同学，那么通过300行的INPUT语句应使A$得到“ZHANG”的值。这时若用A$数组的第一列元素一个个与A$比较，将不会找到相同的数据。因为数组A$的第一列是各位学生的全名，而此处指明要查找的只是姓。但我们通过LEN函数可求出A$的长度，按这个长度通过LEFT$函数可以取数组A$第一列的子串，用子串与A$比较即可达到目的。

习　题

（一）写一个方阵转置程序。例如将方阵

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$$

转置成一个新方阵

$$\begin{pmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{pmatrix}$$

要求转换过程中不要改变方阵名，也不要借别的矩阵的协助。最后要把新方阵打印出来。

（二）写一个矩阵加法程序。并将新矩阵的元素打印出来。例如

$$A=\begin{pmatrix} 12 & 18 & -50 \\ -7 & 325 & 60 \end{pmatrix}$$

$$B=\begin{pmatrix} 58 & -30 & 6 \\ 18 & 72 & -9 \end{pmatrix}$$

求C=A+B。C的结果应是

$$C=\begin{pmatrix} 70 & -12 & -44 \\ 11 & 397 & 51 \end{pmatrix}$$

（三）有若干个整数，要求分别按奇数和偶数由小到大的顺序打印出来，并打印出奇数之和与偶数之和。请编写此程序。具体整数个数及数据，可自行设定，写在DATA语句中。

（四）现有以下一个程序，请读懂它，并指明当输入为“ZHANG SI HAI”，“ZHAO YAN”，“JIA KUN”，数字，标点符号等时，程序运行的结果是什么？

```
10 N=6
20 FOR I=1 TO N
30 READ A$ (I, 1) : READ A$ (I, 2)
40 NEXT
50 INPUT "TYPE NAME TO FIND PAY"; B$
60 IF ASC (B$)>90 OR ASC (B$)<48 OR ASC (B$)>49
     AND ASC (B$)<65 THEN 50
70 IF LEFT$ (B$, 1)>"1" THEN 160
80  ON VAL (LEFT$ (B$, 1)) GOTO 100
90  PRINT "BYE!" : END
100 A=0
110 FOR I=1 TO N
120 A=A+VAL (A$(I, 2))
130 NEXT
140 PRINT "TOTAL:"; A, "MEAN:"; A/N
150 GOTO 50
160 FOR I=1 TO N
170 IF B$=A$(I, 1) THEN 200
180 NEXT
190 PRINT "CAN'T FIND NAME" : GOTO 50
200 PRINT A$(I, 1); "'S PAY IS"; A$ (I, 2); "YUN"
210 GOTO 50
300 DATA "YANG HUAN", 79
310 DATA "ZHAO YAN", 98
```

```
320 DATA "ZHANG SI HAI", 66
330 DATA "LIU SU YUN", 107
340 DATA "SHE HAI JUN", 99
350 DATA "CHEN NAN", 136
```

（五）请对例4的程序作适当修改，以便适应如下查找需要：

1.给出某人全名时，可仍按原来的方法查找；

2.只给出姓时，应将该姓的全部成员作为查找到的对象；

3.只给出名（不知姓）时，应将同名（不管姓什么）成员作为查找到的对象。

要求不改变原程序的其它功能，只需写出要修改（包括增删）的段落即可。

（六）编写一个行主元消去法解线性方程组解的程序。行数、列数及各元素数据，上机时自定。

第八章　图形绘制

前面几章中，我们着重介绍了基本BASIC。从这一章起，主要介绍苹果Ⅱ BASIC的扩展部分。由于这些扩展（当然包括前面已提到的扩展部分）大大丰富了该机的功能。

这一章介绍的保留字较多，计有NORMAL、INVERSE、FLASH、VTAB、HTAB、TAB、POS、SPC、HOME、SPEED=、TEXT、GR、COLOR=、PLOT、VLIN、HLIN、SCRN(、HGR、HGR2、HCOLOR=、HPLOT、DRAW、XDRAW、ROT=、SCALE=、POKE、PEEK等。

§1. 文本方式和文本方式下的图形绘制

一、文本方式

在屏幕上显示字母、数字和其它符号的方式，称为文本方式。所有计算机的显示终端，都有这种方式。苹果机开机时建立的BASIC系统，即处于这种方式之下。我们前些章介绍过的那些程序，都可以在这种方式下输入并运行。

以后我们会看到，苹果机的屏幕也可以显示图形，那时叫作图形方式。从图形方式转回文本方式，可用命令TEXT。

TEXT 命令的格式是：

〔行号〕TEXT

在文本方式下，屏幕从上到下为24行，每行可显示40个字符。因此整幅屏幕的容量为960个字符。每个字符由7(高)×5(宽) 个点阵组成，但实际占用8×7的点阵面积，相当于与邻字留出了间

隔。字符在机内存放采用ASCII码，可正常显示字符为64个。

字符在屏幕上显示时，又有三种不同的显示方式：正常显示方式(黑底白字)、反显示方式（白底黑字）和闪烁显示方式（上述两种显示方式交替变换)。

三种显示方式可分别通过命令NORMAL、INVERSE和FLASH来实现。

这三个命令的格式是：

〔行号〕NORMAL

〔行号〕INVERSE

〔行号〕FLASH

程序运行中凡遇到（或事先设定）上述命令之一时，所有输出到屏幕上的字符，将以命令指定的显示方式显示（不言而喻，开机时机器自行设定为正常显示方式)。

BASIC解释程序执行这三个命令时，仅仅是在“显示方式专用单元”中登记一个显示方式标记，以便需要输出显示时，根据这个标记去实现不同的显示方式。这个专用单元为内存中的第50号单元。正常显示方式的标记为255，反显示方式标记为63，闪烁显示方式标记为127。

在BASIC程序中直接将某一数送入某个单元是可以的。这需要通过POKE语句来完成。

POKE语句的格式是：

〔行号〕POKE　算术表达式1，算术表达式2

算术表达式1是内存单元号码(它必须是内存中可以读写的区域，即RAM中)，算术表达式2的值应是不大于255的正整数（若带小数则截舍小数部分)。语句的功能是把给定的正整数写入(存入）内存的指定单元中。

于是，语句POKE 50，255等同于命令NORMAL；语句POKE 50，63等同于命令INVERSE；而语句POKE 50，127则等同

于命令FLASH。

POKE 语句可以把一个数存入内存指定单元。反过来，也还有一个函数，能把指定单元中的数读出来。这就是PEEK 函数。

PEEK函数的格式是：

PEEK（算术表达式）

算术表达式的值应是一个整数（若为浮点数截舍小数部分），它指明一个内存单元的号码。函数得到的值是指定单元中存放的数。

比如在正常显示方式下，命令：

```
PRINT PEEK(50)
```

将输出255。

POKE和PEEK是很有用的，以后会经常用到。

二、文本窗口

文本窗口指的是文本方式下显示字符的区域。正常情况下，文本窗口为全屏幕。也可以说，文本窗口的左端从第0号字符位置开始，行宽为40个字符，顶端从0行字符开始，上下共24行。这四个数据分别记录在第32、33、34、35号内存单元中。这四个单元称作文本窗口专用单元。

通过POKE命令，可以改变窗口的大小和位置。

比如程序：

```
10 HOME
20 POKE 33, 5 : POKE 35, 15
30 PRINT "ABCDEFGHIJKL"
40 END
```

程序的20行规定窗口宽为5，因此运行结果为：

```
ABCDE
FGHIJ
KL
```

需指出的是，改变窗口时，应先将光标移至窗口内方可。否则它先在窗口外按原窗口显示，进入指定窗口后才能按新窗口的范围实施。

三、光标位置的控制

前面我们用过的PRINT语句，除能输出数据外，实际上也能控制光标的移动。在PRINT的输出项中，如果有一个由若干空格组成的空串，会使光标向右移动若干个位置（它是由输出空格实现的）；一个没有输出项的PRINT语句，可以使光标下移一行等等。但它的基本原则是顺序前进的：一个个右推，或一行行下推。光标不会向左移，也不会向上移，还不能跳过若干位置。

下面介绍一组语句与函数，将可使光标自由转换位置。

1.HOME

这个命令以前介绍过。格式是

〔行号〕 HOME

功能是清除整个屏幕，将光标移至文本窗口的左上角。

2.VTAB

VTAB语句的格式是：

〔行号〕 VTAB 算术表达式

其中算术表达式的值应为1—24的正整数(若有小数部分截舍去)。

语句VTAB的功能是，将光标移动到指定行上。

3.HTAB

HTAB语句的格式是：

〔行号〕 HTAB 算术表达式

其中算术表达式的值应为0—255。

HTAB的功能是，从光标当前所在行（当前行）的最左端(在此可称为1号位置）向后移动到给定号位置(当前行右端为40号位置，下一行第一字符位为41号位置，…)。

注意：算术表达式的值为0时，按256处理。

4.TAB函数

TAB函数格式为：

TAB（算术表达式）

算术表达式的值应在0—255范围内，有小数分部时截舍去。其值为0时按256处理。

TAB函数称作打印格式函数，必须用在PRINT语句中。它与HTAB 语句有点类似，功能是使光标从当前行的最左边向后移动给定的那么多个字符位置。但它不能使光标后退。因此，当算术表达式的值小于光标当前所在列号时，TAB函数不起作用。

5.POS函数

POS函数的格式是

POS（哑元）

所谓哑元，表明它没什么实际意义。但不能不给出。这里的哑元，可以是算术表达式，也可以是字符串常量(用引号对括住)。

这个函数给出一个正整数，表明光标当前位置为当前行的第几个字符位置（左端以0算起)。

6.SPC函数

SPC 函数的格式是：

SPC（算术表达式）

算术表达式的值应为0—255。当有小数部分时舍去。

这个函数也必须用在PRINT语句中。它的功能是在下一个输出项之前插入指定的空格符（虽然它也可以理解为向后移动指定的字符位置，但它一边移动，一边擦除扫描过的位置上的内容。而前面所说的光标移动都不含这种擦除功能。这点要特别注意)。

几个SPC函数可以连用。

四、文本方式下的图形绘制

用上面介绍的各种使光标移动的语句、函数，画出一幅由各

种字符组成的图形，不是十分困难的。比如下面最简单的程序，可以在屏上显示用“＊”号组成的中文“二”字。

```
10 HOME : VTAB 5
20 PRINT TAB(15); "* * * * * *"
30 VTAB 8
40 PRINT TAB (13); "* * * * * * * * * *"
50 END
```

这里需注意两点：

1.用立即执行方式作图不易实现。因为当光在标屏幕底部时，任何一个 RETURN 动作都会使输出行上推。而每个命令(语句)敲入后，总少不了敲 RETURN 键。除非你的图形很简单。

2.虽然通过适当的语句可以使光标在屏幕上作上下左右移动，然后将某些字符输出到不同的位置上，构成一幅图形。但若是要将图形打印输出，最好不要使光标上移或左移。因为打印机的打印头实现不了这种回退，从而使打印出的图形走样。

下面是一个文本方式下用“＊”组成的正弦曲线的程序例：

```
10 FOR X= -3.1416 TO
     3.1416 STEP 3.1416/18
20 PRINT INT (X * 180/3.1416);
30 PRINT TAB (22 + 18 *
   SIN (X)); "*"
40 NEXT
```

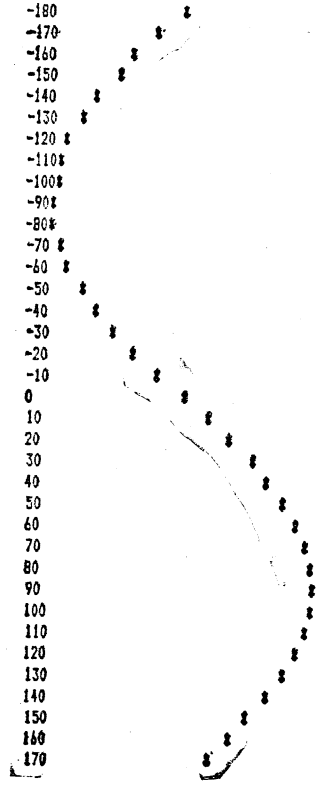


图8—1

```
50 END
```

这个程序运行时，由于屏幕只有24行，所以只能把最后一部分停在屏幕上。但打印机可以全部打印出来。打印的图形如图8—1所示。

§2. 低清晰度图形

图形方式不同于文本方式。它不是由可显示字符在屏幕上构成图形，而是由不同的色块（或色点）在屏幕上连接成图形。

低清晰度，是指这种图形的清晰度（也称分辨度、分辨率）较低。换言之，这类图形由较大的色块来组成，在整幅屏幕上的图素不多，不能画出较细密的图形。也正是这个原因，画图速度较快。它的另一个优点是不增加内存的开销。

一、低清晰度图形屏幕

用低清晰度图形方式绘制图形时，把屏幕当作一个直角坐标系下的平面来看待。它的原点在屏幕的左上角，X 轴沿着顶端一行向右，取值为0到39；Y轴沿着左边一列向下，取值为0到47。因此整幅屏幕可显示$40\times 48=1920$个色块。

屏幕之所以被划成为40列、48行，是由于低清晰度图形在内存中的存贮区借用了文本方式下的屏幕存贮区造成的(这一区域，是内存的1024号到2047号单元)。

前节介绍过，在文本方式下，屏幕被分成40×24个字符位置。可见在一个字符位置上正好是上下两个重叠的色块。

我们以前还介绍过，字符在内存中是以其ASCII编码存放的，而一个ASCII 码占用一个字节，即八位二进制数。在低清晰度图形方式下，把内存图形存贮区的任一字节的八位分成右四位、左四位，分别表示上、下两个图素。

我们还知道，四位二进制数可以表示十进制的0到15这十六个

数。苹果机就利用这十六个不同码来代表十六种不同的颜色，见表8—1：

表8—1

号码	颜色	号码	颜色	号码	颜色	号码	颜色
0	黑	4	深绿	8	棕	12	浅绿
1	深红	5	灰	9	橙	13	黄
2	深蓝	6	中蓝	10	灰	14	海蓝
3	紫红	7	浅蓝	11	粉红	15	白

表中的号码就是四位二进制数用十进制写出的。它代表了颜色的号码，我们以后直接叫作色号。

归纳上面的叙述，可以这样理解：内存的1024到2047这一个区域，是一个屏幕区。其中，一个字节中的数据，在文本方式下是一个字符的 ASCII编码；而在低清晰度图形下为上下相接的两个色块的色号（后四位为上色块色号，前四位为下色块色号）。

二、进入低清晰度图形方式的命令

要想进入低清晰度图形方式，应使用GR命令。

GR命令的格式是：

〔行号〕 GR

这个命令的功能有二：一是清除屏幕（也就是把内存中的屏幕区的内容清除），二是进入低清晰度图形方式。

这个命令是进行低清晰度图形绘制时所要使用的第一个命令。

不过这个命令不是把整幅屏幕都设置成图形方式，它设定的图形窗口是，从顶部一行开始向下只有40行图素位置。也就是说，Y轴上的取值只能从0到39。在图形的下面留出四行字符位置仍属文本方式。这种方式，实际上又称为低清晰度图形和文本混合方式。

GR对屏幕的这种混合设置方法，一方面可以使图形窗口呈正方形，另一方面可使低清晰度图形方式下程序输出的文字，能在下面四个文本行上显示出来。

三、低清晰度图形方式下的PRINT绘图

我们已经知道，PRINT语句可以把其后的输出项显示在屏幕上。实际上也就是把要输出的字符送入屏幕存贮区。象上节说的，在文本方式下我们可以在屏幕上看到一个个字符，而在低清晰度图形方式下，只要把光标置于图形窗口内，我们在屏幕上将看到一部分图形：在每个相应的字符位置上，上边出现一个以该字符ASCII码的八位二进制中后四位为色号的色块，下边出现其前四位为色号的色块。

因此，在低清晰度图形方式下，用适当的PRINT语句写的程序段，运行时可以在屏幕上出现彩色图形。

例如程序

```
 5 HOME
10 FOR I=1 TO 40
20 FOR J=1 TO 40
30 PRINT "L";
40 NEXT J
50 PRINT
60 NEXT I
70 END
```

这个程序运行的结果是使屏幕上显示 $40\times40=1600$ 个字符L。

但若是在程序运行前先执行GR命令，或者增加一个程序行：

```
3 GR
```

那么运行后将会使图形区变成一片浅绿色。

这什么原因呢？

从附录中我们可以查出：字符L的正常显示ASCII码，用十进制表示为204，用十六进制表示为CC，用二进制表示则为11001100。上述程序不管在文本方式还是低清晰度图形方式下，经由30行送入屏幕存贮区的都是一个个这样的编码。在文本方式下被显示为字符L，在低清晰度图形方式按前后两个四位的编码显示相应的色块。这里后四位编码为二进制的1100，十六进为C，十进为12。12号颜色为浅绿。前四位也一样。这样就出现了浅绿色图形屏幕。

类似地，运行下列程序：

```
10 HOME
20 GR
30 PRINT "LLLL::LL::LLLL"
40 PRINN "LLLL::LL::LLLL"
50 PRINT "::::::LL::::::"
60 PRINT "LLLL::LL::LLLL"
70 PRINT "LLLL::LL::LLLL"
80 PRINT "::::::LL::::::"
90 END
```

可以在屏幕的左上角出现一个浅绿底粉红色"北"字。

浅绿底色由所有"L"产生，"北"字则由":"产生。因为":"的ASCII码用十六进制写出是BB，前后四位都是十六进制的B，也就是十进制的11，而11号颜色是粉红色，所以所生成的"北"字就是粉红色。

而下列程序：

```
10 HOME
20 GR
30 PRINT "AAAA"
40 END
```

运行后显示上下并列的两小段色带。因"A"的ASCII码为十六

进制C1，因而上面一条色带为深红色（1号颜色），下面一条色带为浅绿色(12号颜色)。

上述三个程序中，都用了语句HOME，目的是把光标移到图形区中。否则光标只能在下面四行文本区里，那时PRINT的输出，只能以字符的形式出现在图形下方的文本区内。

虽然用PRINT语句在低清晰度图形方式下可以绘制图形，使用前面介绍的有关语句和函数来移动光标当前位置，对于这种绘制还能提供一些帮助。但这种方法也还是麻烦的，特别是选定颜色常常会遇到一些困难，因此这种方法用得不多。用的较多的还是下面介绍的方法：使用绘图语句。

四、低清晰度绘图语句

在使用命令GR进入低清晰度图形和文本混合方式后，下列语句会使你得心应手地绘制各种图形。

1.COLOR = 语句

它的格式是：

〔行号〕COLOR = 算术表达式

这个语句的功能是确定下面画图时用什么颜色。因此算术表达式的值应该是一个色号。这个算术表达式的值允许在0到255的范围内，而真正起作用的还是模16，即被16除后的余数。当有小数部分时舍去。

在使用命令GR时，我们说它的一个作用是清屏幕，这实际上相当于执行了一次COLOR = 0。因此，如果不给出颜色的设定，则认为是用黑色(0号颜色)来作图。那时由于屏幕本身就是黑色，你将什么也看不到。

2.PLOT语句

PLOT语句的格式是：

〔行号〕PLOT 算术表达式1，算术表达式2

其中算术表达式1指明X的坐标值，算术表达式2指明Y的坐标

值。它们的范围应在0到39的范围内（虽然Y的值可大到47，但超过39以后无意义）。如带小数则截舍去。

这个语句的功能是：用最近一次设定过的颜色，在给定的坐标(X，Y)点上画出一个色块。

例如，下列程序可以在屏幕的中间显示红蓝两个色块：

```
20 GR : COLOR = 1
30 PLOT 19, 20
40 COLOR = 6
50 PLOT 20, 20
60 END
```

30行画的点坐标为（19，20），颜色为20行指定的红（1号）色。50行画的点坐标为(20，20)，颜色为40行指定的蓝(6号)色。

3.HLIN语句

HLIN语句的格式是：

〔行号〕 HLIN 算术表达式1，算术表达式2 AT算术表达式3

格式中，前两个算术表达式分别给出两个X的坐标值（设为X1和X2），后一个算术表达式给出Y的坐标值（设为Y）。它们的取值范围同前。

这个语句的作用是用最近一次给定的颜色，从坐标点（X1，Y）到坐标点（X2，Y）画一条水平色条。

比如，下列程序可以在屏幕的顶部显示红黄两条平行水平条带，红的在上，黄的在下。

```
10 GR
20 COLOR = 1
30 HLIN 0, 39 AT 0
40 COLOR = 13
50 HLIN 0, 39 AT 1
60 END
```

4.VLIN语句

VLIN语句格式是：

〔行号〕VLIN 算术表达式1，算术表达式2 AT 算术表达式3

其中前两个算术表达式分别为给定两个Y轴坐标值（设为Y1和Y2），后一个算术表达式为给定的X轴坐标值（设为X）。那么，这个语句的功能就是，用最近一次设定的颜色，从坐标点（X，Y1）到坐标点（X，Y2）画一条垂直色带（算术表达式的取值范围同前）。

例如，下面的程序，可以在屏幕的左边画出红、黄两条垂直色带：

```
10 GR
20 COLOR = 1
30 VLIN 39, 0 AT 1
40 COLOR = 13
50 VLIN 0, 39 AT 0
60 END
```

在这个程序运行时，先在第1列上从下到上画红色条带，再在第0列上从上到下画黄色条带。

5.SCRN函数

SCRN函数的格式是：

SCRN（算术表达式1，算术表达式2）

其中两个算术表达式分别为屏幕上某一个点的X坐标值和Y坐标值。这个函数的功能是给出指定点的颜色号码。可见这两个算术表达式的值也只能在0到39的范围内。

比如对上面最后一个程序例子，运行后将在第1列上画了一条垂直红带，在第0列上画了一条垂直黄带。那么下列程序运行后将输出1和13：

```
10 PRINT SCRN (1, 5), SCRN (0, 10)
```

这个函数在编写游戏程序时较有用。

五、用低清晰度绘图语句绘制图形的几个例子

例1. 彩色方阵

```
10 GR
20 FOR I=0 TO 39
30 FOR J=0 TO 39
40 COLOR=RND (I)
50 PLOT I, J
60 NEXT J, I
70 END
```

这个程序运行后，可以在屏幕上出现$40\times40=1600$个色块组成的方阵。它先在第0列上从上到下画出40个色块来(色号是随机的)，再转第1列上从上到下画出40个色块，…，最后在第39列上从上到下画出40个色块。

例2. 彩条

```
10  GR
20  FOR I=0 TO 31
30  COLOR=I/2
40  VLIN 0, 39 AT I+3
50  NEXT
60  COLOR=5
70  VLIN 0, 39 AT 2
80  VLIN 0, 39 AT 35
90  HLIN 2, 35 AT 0
100 HLIN 2, 35 AT 39
110 FOR I=0 TO 14 STEP 2
120 HTAB 2*I+4 : PRINT I,
130 NEXT I
```

```
140 PRINT
150 END
```

这个程序运行后会先在屏幕上画出32条垂直彩条，由于30行的作用，每两条共用一种颜色，故在屏上为16条垂直色带。随后用灰色在这些彩条外沿画了一个边框，并在文本区由PRINT语句输出0，2，4，…，14八个数，正好标明它上方彩条的颜色号码。

例3．彩色框

```
10  GR
20  FOR I=0 TO 19 STEP 2
30  COLOR=1+I/2
40  HLIN I, 39-I AT I
50  HLIN I, 39-I AT 39-I
60  VLIN I, 39-I AT I
70  VLIN I, 39-I AT 39-I
80  NEXT
90  FOR I=0 TO 1000 : NEXT I
100 GOTO 10
```

这个程序运行时，先在屏幕上画一个小方框，逐次向内画出一个较小的方框，这样画出10个从大到小的方框后，在屏上停顿一会儿，清除屏幕，再从头画起，反复无穷。

例4．一个跑动点

```
10 GR
20 FOR I=0 TO 39
30 COLOR=0
40 PLOT X, X
50 COLOR=13
60 PLOT I, I
70 X=I
80 NEXT I
90 END
```

这个程序运行时会使一个黄色的点从屏幕的左上角跑到右下角。40行的作用是清除屏幕上的一个色点，在这里指明其坐标是(X, X)。60行是画一个黄点，坐标为(I, I)。70行使X取得I的值，从而返回时在30行上又清除它。这样先画后清，位置逐次向右下方移动，看起来像这个点在跑动。

如果嫌这个点跑得太快看不清楚，则可增加下面一个程序行：

```
25 FOR J=1 TO 100 : NEXT J
```

例5. 弹球运动

```
10 GR
20 X=0 : Y=5
30 XU=2 : YU=1
40 NX=X+XU : NY=Y+YU
50 IF NX>39 THEN NX=39 : XU=-XU
60 IF NX<0 THEN NX=0 : XU=-XU
70 IF NY>39 THEN NY=39 : YU=-YU
80 IF NY<0 THEN NY=0 : YU=-YU
90 COLOR=5 : PLOT NX,NY
100 COLOR=0 : PLOT X, Y
110 X=NX : Y=NY
120 GOTO 40
```

这个程序中的变量XU和YU分别表示每次画点时X坐和Y坐标值的改变量。程序的50行至80行用来检查下次画点的坐标位置是否会跑出图形区。如果会跑出去，则使其向反方向跑去。好象一个弹球，碰到边框时将其弹回去。

例6. 写汉字

下面的程序，可以在屏幕上画出红色横标，中间有黄色“中国”二字：

```
10 GR
20 COLOR=1
```

```
30  FOR I=11 TO 31
40  HLIN 2, 38 AT I
50  NEXT I
55  COLOR=13
60  VLIN 18, 23 AT  6
70  HLIN 7, 17  AT 18
80  VLIN 18, 23 AT 18
90  HLIN 7, 17  AT 23
100 VLIN 15, 27 AT 12
110 VLIN 15, 27 AT 22
120 HLIN 23, 33 AT 15
130 VLIN 15, 27 AT 34
140 HLIN 24, 32 AT 18
150 HLIN 24, 32 AT 21
160 VLIN 19, 24 AT 28 : PLOT 30, 23
170 HLIN 24, 32 AT 25
180 HLIN 23, 33 AT 27
190 END
```

这个程序的20至50行用来画一条红色横标。以后的程序行用来写“中国”二字。

写这类程序时，最好先找一张（或画一张）方格纸，在40×40的范围内写好要写的字，然后确定什么地方需画垂线、什么地方需画水平线、什么地方需画点，并确定出有关的坐标值来，据此来写你的程序。

汉字可以写出，一个动物形象或别的形象也可类似地画出。由于可以在不同地方用不同颜色，形象会更逼真。如果让图形的某些点作一些颜色变换，画出的动物眼睛或嘴巴就能动起来，更具有吸引力。

例7. 直方图

假定某工厂从76年以来到83年为止，逐年的产值是：15.5万元、17.34万元、16.2万元、17.3万元、20.45万元、25.3万元、32.94万元、45.8万元。请画出直方图来。

由于图形屏幕从上到下只有40个点，这就要选取一个适当的比例因子，使处理后的数据能在屏幕内放的下，同时也不能只使用屏幕的很小区域。此外，由于图形屏幕的Y轴和正常的坐标系的Y轴反向，要使屏幕显示符合常规才行。这些在程序中都应作处理。

程序如下：

```
10  FOR I=76 TO 83
20  READ A(I-76)
30  NEXT I
40  FOR I=0 TO 83-76
50  IF A<A(I) THEN A=A(I)
60  NEXT I
70  A=A/40
80  GR
90  COLOR=1
100 FOR I=0 TO 83-76
110 VLIN 39, 40-A(I)/A AT I*4+6
120 NEXT I
130 FOR I=76 TO 83
140 PRINT TAB ((I-76)*4+6); I;
150 NEXT I
160 DATA 15.5, 17.34, 16.2, 17.3, 20.45, 25.3, 32.94,45.8
200 END
```

从上面这些例子可以看出，用绘图语句来绘制图形要方便灵活得多，而且可以是彩色的，很吸引人。但低清晰度图形毕竟绘制不出更精细的图形来，而且低清晰度图形无法从打印机上打印

出来，这是它的缺点。

§3. 高清晰度图形

高清晰度图形方式克服了低清晰度图形方式的缺点，它是以增加内存的开销实现的。

一、高清晰度图形屏幕

在本章开始介绍文本方式时，我们提到过，每行可显示40个字符，而每个字符实际占用7个点的宽度，可见屏宽为40×7=280个点。一幅屏幕可显示24行字符，每个字符实际占用8个点的位置，所以屏高应为24×8=192个点。这就是说，屏幕的最大分辨度也只能把屏幕区分为280×192=53760个点阵。高清晰度图形屏幕也是把屏幕当作一个直角坐标系下的平面，原点（0,0)在左上角，X轴方向朝右，取值范围为0到279；Y轴方向朝下，取值范围为0到191。整幅屏幕为53760个点素构成。

在低清晰度图形方式下，图形存贮区的一个字节表示上下相连的两个色块，在高清晰度图形方式下，用后七位的状态是1还是0分别表明在七个点的位置上有点还是无点。这样从左到右正好由40个字节表示高清晰度图形的一行信息。由于整幅屏幕有192行，因而需要有40×192=7680个字节才能放下一幅高清晰度图形的信息。显然文本屏幕存贮区（只有1024个字节）是远远不够使用的。为此，干脆在内存中适当的地方开辟了两个高清晰度图形存贮区，以便能存放（或显示）两幅图形，并分别地称这两个存贮区为高清晰度图形第一页和高清晰度图形第二页。它们在内存中的地址分别是8192到16383和16384到24575。每个区域长度为8192个字节（它比7680稍大一点，是十六进数的2000，计算地址时方便些)。

高清晰度图形存贮区的一个字节中的后七位用以表示有点无

点，只剩下一位可以来控制颜色，那就只有两种颜色了（一位只能有两种状态），这显然是不好的。为此，对颜色的控制采用了如下一些补救办法：

同低清晰度图形类似，用语句设定色号。对应关系如下表：

色　　号	颜　　色	色　　号	颜　　色
0	黑	4	黑
1	绿*	5	橙*
2	紫*	6	蓝*
3	白	7	白

带*号者，要依所用电视机而定。

点的颜色还由下列因素决定：

当点的相应位为0时，屏上是一个黑点（同屏幕的低色，看不到），点的相应位为1时，则要看点在屏幕上的相对位置：处在偶数列时，可能显示紫色；处在奇数列时，可能显示绿色；若要显示两个并排挨着的点，这两个点均呈白色。当点的相应位为1，色选位（即一个字节中的最高1位）也是1时，则紫变兰、绿变橙等。

在这种情形下，高清晰度图形每个点的颜色，不能象低清晰度图形那样准确，有些点自动改变了颜色，甚至有时变成黑色而出现漏点现象。但对3号、7号颜色，不会出现漏点；1号、4号颜色也不会冒点。

二、进入高清晰度图形方式的命令

从文本方式或低清晰度图形方式进入高清晰度图形方式，要使用相应的命令：

HGR或HGR2。它们的格式是：

〔行号〕 HGR

〔行号〕 HGR2

HGR 命令的功能是：清除屏幕，进入高清晰度图形第一页和文本混合方式。这就是说，这个命令将使屏幕与内存的8192到16383对应起来，同时在下部留出四行文本方式供文本输出用。这时，图形的Y轴坐标值最大为159。

HGR2命令的功能是清屏幕，进入高清晰度图形第二页全屏幕方式。换言之，它使屏幕与内存的16384到24575对应起来，屏幕上不留文本输出行，全屏幕都用于图形。

三、高清晰度绘图语句

在进入高清晰度图形方式后，下列语句可供绘图使用。

1.HCOLOR＝语句

这个语句的格式是：

〔行号〕 HCOLOR＝算术表达式

其中算术表达式的值，只允许0～7范围内的数。带小数点时截舍去。

这个语句用来设定其后画图时使用的颜色号码。

在执行HGR或HGR2语句时，实际上执行了一次HCOLOR＝0的操作。所以不给出这个语句时，其后画图时在屏幕上将什么也看不到。

2.HPLOT语句

给出一点的坐标值，如为(X,Y)，那么语句

HPLOT X,Y

可以在 (X,Y) 处以最近一次设定的颜色画出一个点。

给出两个点的坐标值，如(X1,Y1,)，(X2,Y2)，那么,语句

HPLOT X1, Y1 TO X2,Y2

可以从 (X1,Y1) 到 (X2,Y2) 之间用最近一次给定的颜色连一条直线。

进一步，给出三个点的坐标值，如 (X1,Y1)，(X2, Y2)，(X3,Y3)。那么，语句

HPLOT X1, Y1 TO X2, Y2 TO X3, Y3

可以用最近一次设定的颜色，从 (X1,Y1) 到 (X2, Y2) 连一条直线，接着从 (X2,Y2) 到 (X3,Y3) 再连一条直线。

……

可见，HPLOT的语句格式是：

〔行号〕 HPLOT 算术表达式1，算术表达式 2〔{TO 算术表达式N，算术表达式M}〕

其中算术表达式必须成对出现，每对中的第一个代表某一点的X坐标值，第二个代表同一点的 Y 坐标值，它们之间用逗号分隔。它们的取值范围不能在屏幕之外。每对坐标值之间要有保留字 TO。这个语句不限定坐标点的个数，只要一个程序行放得下就行。

这个语句的功能，可以画点（只跟一个坐标对时），也可以连线（跟几个坐标对时）。颜色用最近一个HCOLOR = 所设定的。

四、高清晰度图形程序举例

1.用画点的办法，在屏幕中间画一蓝色方框：

```
10 HGR
20 HCOLOR = 6
30 FOR I = 100 TO 200
40 HPLOT I, 50
50 NEXT
60 FOR I = 50 TO 120
70 HPLOT 200, I
80 NEXT
90 FOR I = 200 TO 100 STEP -1
100 HPLOT I, 120
110 NEXT
120 FOR I = 120 TO 50 STEP -1
130 HPLOT 100, I
```

```
140 NEXT
150 END
```

例2.上题改为画线方法，程序短，运行也快：

```
10  HGR
20  HCOLOR = 6
30  HPLOT 100, 50 TO 200, 50 TO 200,  120 TO 100, 120
    TO 100, 50
40  END
```

例3.下面的程序，可以在屏幕上画出八个同心圆。最内层的圆半径取10，逐层向外每个圆半径扩大10。圆心在屏幕的中间。

```
10  HGR2
20  HCOLOR = 7
30  R1 = 10 : R2 = 80 : S = 10 : CX = 140 : CY = 95
40  FOR R = R1 TO R2 STEP S
50  FOR X = -R TO R
60  Y = INT (SQR (R * R - X * X))
70  HPLOT CX + X, CY - Y
80  HPLOT CX + X, CY + Y
90  NEXT X, R
100 END
```

程序中的R，在各次循环中为各个圆的半径，X、Y则为圆上的坐标点。所以 $Y=\sqrt{R^2-X^2}$。60行就是用的这一公式。由于图形屏幕原点在左上角，70行和80行的HPLOT语句中对X和Y的坐标值分别加了CX和CY，相当于把原点移到了屏幕中心。虽然Y轴仍与数学上的Y轴反向，但由于圆的上下对称性，也就无关紧要了。

例4.上题程序所画出的圆，靠近Y轴的部位点较紧密，而远离Y轴的部位点较稀疏。

改进的办法很多。例如，以圆心为中心，向外均匀射出360根

等长线段，将各线段终点连结起来就是一个匀称的圆。

下面程序就可用这种办法画出上例中最外层那个圆：

```
10 HGR2
20 HCOLOR = 7
30 C = 3.14159/180
40 FOR I = 0 TO 359
50 X = 80 * COS (I * C)
60 Y = 80 * SIN (I * C)
70 HPLOT INT (X) + 140, INT (Y) + 95
80 NEXT
90 END
```

例5. 上述程序的缺点是速度慢。计算360个坐标点要花一定时间。而且每计算一次，只能画出一个点来。这样在每画出两个点之间要停留一段时间，会使人感到不耐烦。

改进的办法是每隔几度计算一个点，画圆时连接这些点之间的直线也就是一个近似的圆（在显示图形时一般没有必要十分精确）。此外还可以先将这些点的坐标值算出后，存放在数组中，画图时直接使用这些元素就行了。这样画图本身的动作显得很快。

现写出这个程序如下：

```
10 DIM X(120), Y(120)
20 C = 3.14159/180 * 3
30 FOR I = 0 TO 120
40 X = 80 * COS (I * C)
50 Y = 80 * SIN (I * C)
60 X(I) = INT (X) + 140
70 Y(I) = INT (Y) + 95
80 NEXT
90 HGR2
100 HCOLOR = 7
110 FOR I = 0 TO 119
```

```
120 HPLOT X(I), Y(I) TO X(I+1), Y(I+1)
130 NEXT
140 END
```

例6.美丽的波纹

程序如下：

```
10  HGR
20  A=RND (1)*279
30  B=RND (1)*159
40  FOR X=0 TO 278 STEP 3
50  FOR S=0 TO 1
60  HCOLOR=3*S
70  HPLOT X+S, 0 TO A, B TO 279-X-S, 159
80  NEXT S, X
90  FOR Y=0 TO 158  STFP 3
100 FOR S=0 TO 1
110 HCOLOR=S*3
120 HPLOT 279,Y+S TO A,B TO 0,159-Y-S
130 NEXT S,Y
140 FOR I=0 TO 1500:NEXT I
150 GOTO 10
```

这个程序中的60行和110虽然只给定了用0号和3号颜色，似乎只能是黑白图形，其实它是很美丽的彩色图案。这种颜色的自行改变，本节一开始已讲过。从这里可以看出这种颜色的自动改变并不完全是一件坏事。

例7.拉伸的方盒

程序如下：

```
10  HGR2
20  HCOLOR=3
30  FOR I=1 TO 37
40  A%=INT (SGN (-1)*2.121*I)
```

```
50  B% = INT (SGN (1) * 2.121 * I)
60  A0% = 5 + 4 * (I - 1)
70  B0% = 5 + 3 * (I - 1)
80  X1% = A0% + A% : X2% = A0% + B%
90  Y1% = B0% + A% : Y2% = B0% + B%
100 HPLOT X1%, Y1% TO X1%, Y2% TO X2%,Y2% TO
    X2%,Y1% TO X1%,Y1%
110 NEXT
120 FOR I = 1 TO 1000 : NEXT
130 GOTO 10
```

通过上述几例可以看出，高清晰度绘图语句很简单，各种图形都能画出。顺便提一句，凡规则图形（整幅的或局部的）一般可通过特定表达式的计算来求出有关点的坐标，这些计算过程与画图部分写在同一程序中。对于一些不规则的图形，最好的办法还是用一张方格纸，在上面画好图形，再人工地找出有关点的坐标数据，然后写入程序中。这虽然是个笨办法，但任何图形都可以画出。

五、高清晰度图形的存贮与调用

屏幕的任何一幅图形都是由程序运行时产生的。因此把这些画图程序存入磁盘，以后可以再调入内存并运行它，图形便能再现。但是，这种图形的产生仍离不开画图程序的运行。能否有一种办法直接把图形存入磁盘，以后直接将它调出显示呢？

这是完全可以的。这要用到操作系统的两个命令（这里先学会一般用法，以后还会进一步介绍它们）。

如果想把已经画好的第一页图形存入磁盘，可用命令：

BSAVE 名字，A8192，L8192

其中名字指的是图形名（不必与程序名同名），以字母打头就可以。A后边的8192，表明图形页的起始地址，L后边的8192，表明图形区的长度。

类似地，如果想把已画好的第二页图形存入磁盘，则用命令：

BSAVE 名字，A16384，L8192

这个命令叫BSAVE命令，它把指定地址开始的指定长度的二进制信息以给定名字存入磁盘上。

把存于磁盘上的图形调出来，用命令：

BLOAD 名字

这里的名字用存入时的名字。

值得注意的是，BLOAD 只能把图形信息调入图形存贮区，它本身不包含显示功能。

因此，想显示存于盘上的第一页图形，必须先用 HGR命令，将屏幕变为高清晰图形一页和文本混合方式后，再用 BLOAD 命令调入图形。这样一边调出一边显示。等到调入全部信息，整幅图形也就全部显示出来了。

同样，先用HGR2 命令之后，再用 BLOAD 命令调入的第二页图形，也可显示出来。

请读者考虑：先用 BLOAD命令将存入盘上的图形信息调入内存，再用 HGR 或HGR2命令，会出现什么情况？

高清晰度图形还可以打印出来。这要先接通打印机（用PR#命令，这要看你的打印机接在几号槽口上，在#之后标明槽口号）。然后用CTRL－Q或CTRL－IG（一页）或CTRL－IG2（二页）。由于打印机接口程序不同而有所不同。

六、屏幕软开关

上面我们介绍了屏幕的几种工作方式，它们之间的转换可通过如下几个命令：

TEXT	整幅		文本
GR	混合、低清晰度		
HGR	一页、混合	高清晰度	图形、清屏
HGR2	二页、整幅		

而这几个命令之所以能实现屏幕方式的转换，是通过对八个专用单元的控制作用实现的。这八个专用单元好象八个按钮开关，每个开关专管一件事；由于它们是受软件控制的(不是用手按的)，所以称作屏幕软开关。现将这八个软开关的地址和作用列于表8—2：

表8—2

内存地址		作用
十进制数	十六进制数	
49232	$C050	图形
49233	$C051	文本
49234	$C052	整幅
49235	$C053	混合
49236	$C054	第1页
49237	$C055	第2页
49238	$C056	低清晰度
49239	$C057	高清晰度

用软件来按这些开关时，通常用 POKE 语句或 PEEK函数。因为它们都可以使指定的地址线上产生一个高电位。至于写入什么数或读出什么数，都是无关紧要的。

因此，下列程序段：

```
⋮
100 POKE 49232,0 : POKE 49235,0
110 POKE 49236,0 : POKE 49239,0
⋮
```

基本上相当于执行一个HGR命令。

而下列程序段：

```
⋮
100 P = PEEK (49232) : P = PEEK (49234)
```

```
110 P = PEEK (49236) : P = PEEK (49239)
:
```

基本上接近于执行一个HGR命令，但它把屏幕变成了整幅图形方式，也就是说没有留出下面的四个文本行。

对于低清晰度图形方式和文本方式，同样可以用软开关进入。只是GR和TEXT只能使用第一页。

我们上面都用了“基本上”三个字，意思是指它们之间还是有差别的。这些差别主要有两个方面：用软开关可以组合出更多的屏幕方式；软开关不具有清除屏幕的作用。这些差别在多种场合下能使屏幕软开关发挥有效的作用。

关于软开关可以设置更多的屏幕方式，上面已经见到，它可以实现图形一页的全屏幕，而GR语句是做不到的。它的作用不言自明。

关于软开关不清屏幕，我们通过一个例子来说明它的某些用途。

有程序：

```
10  HGR : HCOLOR = 3
20  HPLOT 0,0 TO 100,0 TO 100,100 TO 0,0
30  HGR2 : HCOLOR = 7
40  HPLOT 100,100 TO 100,180 TO 270,180 TO 270,100 TO
    100,100
50  GOTO 10
```

这个程序运行时先清屏幕，在1页上画出一个三角形，然后再清屏幕，在2页上画出一个四边形，执行到50行又转回去重复。这样在屏幕上将出现三角形与四边形交替显示的现象。其中每次出现的图形，是通过执行绘图语句画出的。可以想象得到，如果每个图形都是非常复杂的，计算它们各点的坐标位置需要较长的时间，那么这样两幅图形的交替显示速度将大大地放慢。假使我

们将上述程序从50行开始改写成下面的样子：

```
50  POKE 49232,0
60  POKE 49239,0
70  POKE 49236,0
80  POKE 49234,0
90  POKE 49237,0
100 POKE 49236,0
110 GOTO 90
```

这个程序的前4行仍如上面提到的情形，它分别在一页和二页上画出三角形和四边形，50行至80行是置屏幕为高清晰图形第一页全屏幕方式，这时第一页的图形显示出来。执行90行时，又设置为第二页（其它开关未变），第二页图形显示出来。执行100行，又设置成第一页，第一页图形再被显示。转回90行后，再次变成第二页，显示二页图形。……这样仍然是两幅图形的交替显示。由于屏幕软开关不清除屏幕（也就是不清内存中相应的图形存贮区），所以两幅图形均无需重画，这就大大加快了显示速度，以至于往往不得不加入如下两个程序行：

```
95  FOR I=1 TO 1000:NEXT
105 FOR I=1 TO 1000:NEXT
```

如果把上面的程序中50行以前的部分作这样的改动：使在图形一页上画一个小孩，眼睛和嘴巴都张开着，而在图形二页上也画出这个小孩，眼睛和嘴巴都闭着。那么运行这个程序时，两个图形交替显示的效果将是：这个小孩一会儿张嘴睁眼，一会儿闭嘴合眼，好象在和你说话一样。

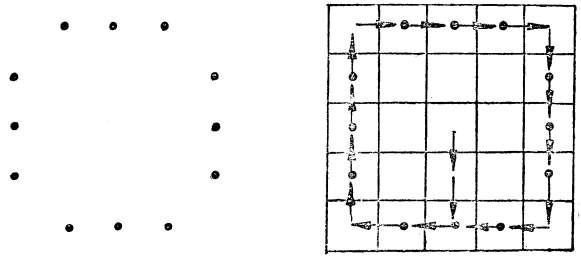
§4．用图形表方法绘制高清晰度图形

用图形表方法，是另一种绘制高清晰度图形的技巧。

这种方法的基本思想是把绘图时的笔划移动过程记录下来，做成一个图形表存入内存中，然后由特定的语句写成程序，这个程序运行时会使图形显示出来，而且这种图形可以旋转，还可以放大。

一、图形定义

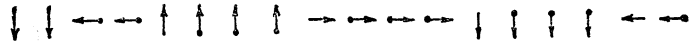
假定我们希望画出如图8—2(*a*)的图形，可拿一张方格纸来具体画出。我们用图形向量依次记录笔的移动过程：当笔向什么



(*a*)　　(*b*)

图8—2

方向移动，就画一个向那个方向的箭头；移出这一格子时如需画点，在箭根画一个圆点。如果我们先把笔置图8—2中心，可以画成图8—2(*b*)的样子。它的向量序列是：



为把这类图形向量序列保存到内存中，规定用三位二进制数表示一个图形向量：最左一位表示带点不带点（带点为1，不带点为0），后两位表示箭头方向：00为向上，01为向右，10为向下，11为向左。

这样一个字节的八位，可用后六位表示两个图形向量。为节

约内存，把左边的两位也尽量利用起来：用以表示不带点的向右、向下和向左，其它情况下都不用它而置为0。

同时还约定，图形向量存入内存时，先填右三位，再填中三位，后填左两位。当一个字节填完后，转内存下一个单元继续填。当一个图形的所有图形向量填完后，填一个全零字节作为结束标志（此前一个字节若未填满则补0）。

这样，把上面那个图形的向量序列存入内存中，将是如下的样子：

序号	图形向量	图形定义的二进制数								十进制数
1		0	0	0	1	0	0	1	0	18
2		0	0	1	1	1	1	1	1	63
3		0	0	1	0	0	0	0	0	32
4		0	1	1	0	0	1	0	0	100
5		0	0	1	0	1	1	0	1	45
6		0	0	0	1	0	1	0	1	21
7		0	0	1	1	0	1	1	0	54
8		0	0	0	1	1	1	1	0	30
9		0	0	0	0	0	1	1	1	7
10		0	0	0	0	0	0	0	0	0

以上是一个图形的定义。占用了内存连续10个字节。如果还有别的图形，用类似的方法也可构造另外的图形定义。

二、图形表

若干个图形定义，加上一些索引信息，可以构成一个完整的图形表（一个图形表最多可有256个图形定义）。

图形表安排在内存中，假定从S号内存单元开始放起，那么完整的图形表如图8—3所示。

假定我们仅就前面那个图形定义造一个图形表，它应该是图8—4那个样子。

三、如何将图形表送入机器中

要把一个图形表送入机器中，首先要确定S的值，即图形表在内存中的首地址。这个地址要取得合适，否则运行一个程序时会把它破坏掉。在此我们建议用十六进制的6000，它是十进制的24576。

图形表首地址确定后，下列程序可以将前面我们所举的那个图形表送入内存中：

```
100 S=24576
110 FOR I=0 TO 13
120 READ A
130 POKE S+I,A
140 NEXT I
150 DATA 1,0,4,0,18,63,32,100,45,21,54,30,7,0
```

此外，还需把这个首地址的十六进制表示送入232和233号单元中。这可以由下面两个命令实现：

```
POKE  232,0
POKE  233,96
```

232和233号单元是专门用来记图形表首地址的。96是十六进制60的十进制表示。

只有这样将图形表送入内存，并将其首地址记入232和233号单元，这个图形表才能够使用。

四、图形表的使用

使用图形表，指的是把图形表中所定义的图形显示出来。这要运行一个含有特殊语句的程序来实现。这些特殊语句有：

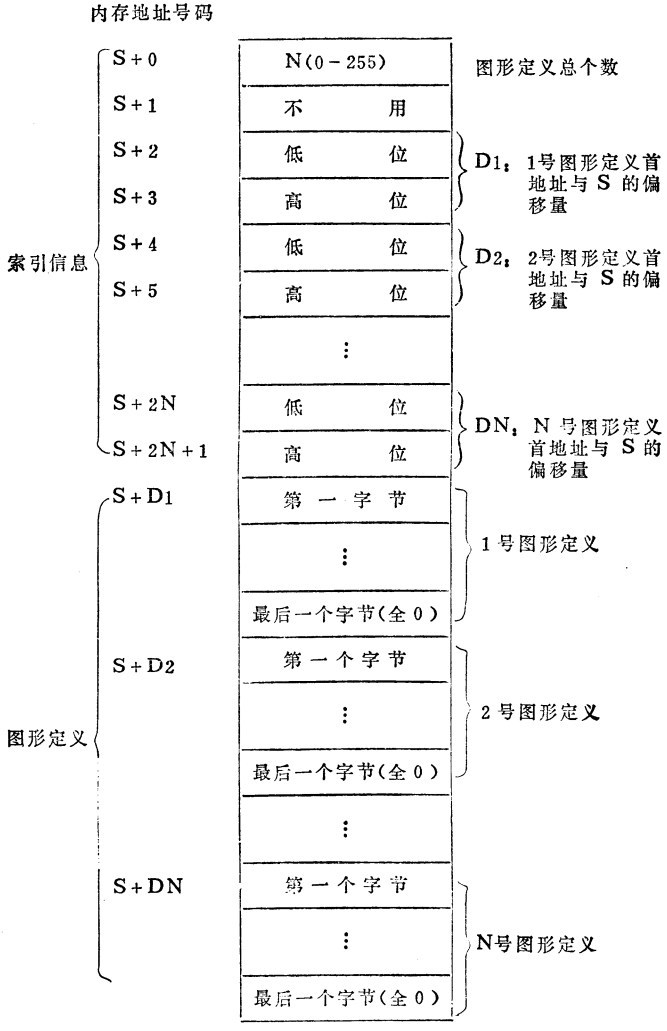

内存地址号码
索引信息
S+0
S+1
S+2
S+3
S+4
S+5
S+2N
S+2N+1
N(0-255)
图形定义总个数
不 用
低 位
高 位
D1：1号图形定义首地址与S的偏移量
D2：2号图形定义首地址与S的偏移量
DN：N号图形定义首地址与S的偏移量
图形定义
S+D1
S+D2
S+DN
第一字节
最后一个字节(全0)
第一个字节
1号图形定义
2号图形定义
N号图形定义

图8—3

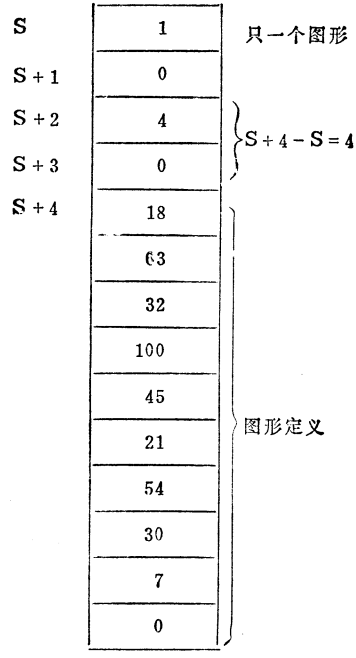


图8—4

1.SCALE=

SCALE=的格式是：

〔行号〕 SCALE=算术表达式

它的功能是用来设定画面的尺寸。

其中算术表达式的值应为1到255的数。

当算术表达式的值为1时，每个图形向量对应屏幕上的一个点，如果算术表达式的值为5，每个图形向量对应屏幕上的5个点（当算术表达式的值为0时，表示最大的形状，不是单独的一个点）。

一般用这个语句来实现图形的放大。

2.DRAW

DRAW的格式是：

〔行号〕 DRAW 算术表达式1〔AT算术表达式2，算术表达式3〕

其中算术表达式1，指明取图形定义表中的第几个图形。算术表达式2和算术表达式3构成了一个点的两个坐标值，以它作为画图的起笔点。不指明可选项时，则从程序的上一个HPLOT或DRAW语句画图时的笔的终结位置作为这次画笔的起点。

3.XDRAW

格式是：

〔行号〕XDRAW 算术表达式1〔AT 算术表达式2，算术表达式3〕

它和 DRAW 的功能一样，只是其颜色取原来的补色。颜色的互补关系是：

黑—白

兰—绿

紫—橙

使用这个语句，可以擦除上次已画过的图形；两次使用它，可以改变图形的颜色，而不改变背景的颜色。

4.ROT =

ROT = 的格式是：

〔行号〕 ROT = 算术表达式

其中算术表达式的取值范围为0到255。

ROT = 的功能是使图形转动一定的角度。

ROT = 0，不转。ROT = 16，按顺时针方向转90角。 ROT = 32，转180度，以此类推。ROT = 64，等同于ROT = 0，不转。

这个语句与SCALE = 语句有关。当SCALE = 1时，只有4个旋转值能为ROT = 使用(即0，16，32，48)。当SCALE = 2时，则有8个旋转值能为ROT = 使用(0，8，16，24，32，40，48，56)。

以此类推。不能为 ROT= 使用的旋转值，以比它较小的一个可使用值处理。

以上一组语句(也称命令)是专用于图形表方法的。现在我们使用有关语句来写一个程序，看怎样使用我们已建立的那个图形表：

```
10  HGR2
20  HCOLOR=3
30  FOR I=1 TO 50
40  ROT=I
50  SCALE=I
60  DRAW 1 AT 139,79
70  NEXT I
80  END
```

这个程序运行时，原来的图形显示出来，并且随着逐次循环，图形越来越大，还不断转动角度，角度变换也逐次加大。这个程序运行结束后，显示于屏幕上的是如图8—5的图形。

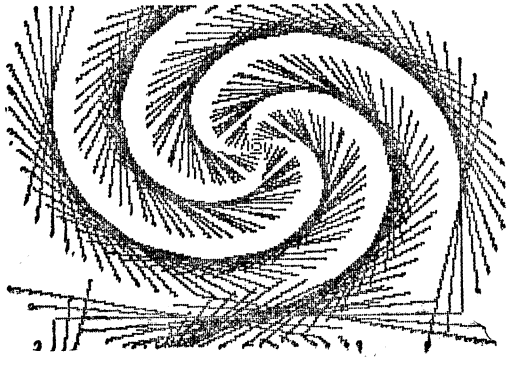


图8—5

五、图形表的存贮与调用

一个图形表在内存中建立好之后，也可以起个名字存入磁盘

上，以备以后直接使用。

存贮图形表可象存贮图形一样，可用立即执行的 BSAVE命令：

BSAVE 名字，A起始地址，L长度

对于我们所举的例子来说，就可以用命令：

BSAVE ABC，A24576，L14

这时用 CATALOG 命令，可以看到磁盘上有一个名为ABC的文件。

如果想调出来使用时，就用命令：

BLOAD ABC

这时不要忘记，要再次用POKE 命令把这个图形表的首地址送入232和233号单元中。因为这两个单元的内容在做 BSAVE 命令时，并未存入磁盘，。而每次开机后这两个单元的数据并不是你需要的地址。此外，也不要忘记先用HGR(或HGR2)命令。

六、一个有用的辅助程序

在编制一个图形定义时，我们费了几道手续：依次画出图形向量，变成对应的三位二进制数，填入一个个 8 位长的字节中，这时还得考虑最左端两位是否可以利用，不能利用的要填00等等，还是很麻烦的。

下面的辅助程序可以帮助你完成这些事。运行这个程序时，只要从键盘上回答机器的提问就行了。机器会问你笔的移动方向是什么?如果向上，回答U(UP)；如果向下，回答D(DOWN)；如果向右，回答R(RIGHT)；如果向左，回答L(LEFT)；如果画完了，不移动时，回答E(END)。回答完移动方向后，机器还问你移动前画不画点？要画，回答Y(YES)；不画，回答N(NO)。然后再问移动方向。这样你可一次次回答，直到回答 E（如果某一笔输入错了，还可以修改)。这时机器还会把这个图形定义给你打印出来。

程序如下：

```
5   REM SHAPE CREATION PROGRAM
10  DIM S1(100),V1(100)
20  I=0:B=0:Q=0
30  PRINT "CREATE SHAPE VECTORS"
40  PRINT
50  V=I:GOSUB 300
60  REM CONTINUE ENTRY UNTIL M$ EQUALS TERMIN
    AL VALUE "E"
70  IF M$<> "E" THEN S1(I)=M:I=I+1:GOTO 50
80  PRINT
90  INPUT "VECTOR TO CHANGE(0=END):";V
100 IF V>0 THEN V=V-1:GOSUB 300:S1(V)=M:GOTO 90
110 FOR V=0 TO I
120 IF B=2 AND S1(V)>0 AND S1(V)<4 THEN 150
130 IF B<2 AND (S1(V)>0 OR S1(V)>4) THEN 150
140 B=0:Q=Q+1
150 V1(Q)=V1(Q)+S1(V)*(8↑B)
160 B=B+1
170 IF B>2 THEN B=0:Q=Q+1
180 NEXT V
190 PRINT "BYTE", "VECTOR"
200 FOR V=0 TO Q
210 PRINT V,V1(V)
220 NEXT V
290 END
300 PRINT "VECTOR"; V+1; ":"
310 INPUT "MOVE:U/D/L/R?"; M$
320 M=0
330 IF M$= "R" THEN M=1
```

```
340 IF M$ = "D" THEN M=2
350 IF M$ = "L" THEN M=3
360 IF M$ = "E" THEN RETURN
370 INPUT "PLOT(Y=YES,N=NO)?"; P$
380 IF P$ = "Y" THEN M=M+4 : RETURN
390 IF P$ = "N" THEN RETURN
400 GOTO 370
```

程序中的50行至70用来完成图形向量的输入，并把它存入数组S1中。其中用了300行到400行的子程序，这个子程序把键盘回答的内容M$、P$转换成一个个图形向量数据。

程序90行到100行用来对某个图形向量的修改。

110行到180行完成S1数组到V1数组的转换。S1数组的一个元素存放一个图形向量，V1数组则是按图形定义构造的。其中一个元素可以是2到3个图形向量数据。

程序190行到220行可以把图形定义打印出来。

读者不难想到，只要给出图形定义在内存中的首地址，可以把这个图形表存入内存中。

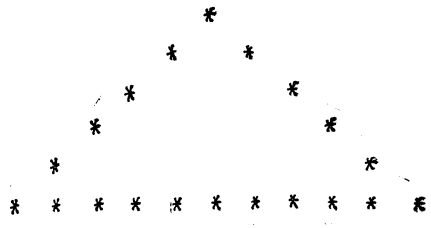
我们增加如下几行程序，这一目的即可达到：

```
230 INPUT "ADDRESS:"; A
240 FOR I=0 TO Q
250 POKE A+I,V1(I)
260 NEXT I
270 POKE A+Q, 0
```

其中230行要求给出图形表的首地址。

习　题

（一）编写一个程序，打印出以下图形：



（二）

$$\begin{cases} y_1 = \sin(x) \\ y_2 = 2\sin(x+30°) \end{cases}$$

其中x为0°，10°，20°，…，360°。试编写一下程序，用“*”号和“!”号分别把y_1和y_2的函数曲线画出来。

（三）已知1986年元旦为星期三，请写一个程序，使能打印出该年的月历来。

（四）读懂下列程序，说明它运行时在屏幕上能看到些什么？

```
10  GR
20  FOR I=39 TO 14 STEP-1
30  COLOR=6
40  VLIN 20,39 AT I
50  IF I=39 THEN 130
60  IF I>27 THEN 80
70  VLIN 20,39 AT I+12
80  COLOR=0
90  VLIN 20,38 AT I+1
100 IF I>26 THEN 120
110 VLIN 20,39 AT I+13
120 FOR J=1 TO 50:NEXT J
130 NEXT I
140 COLOR=9
150 FOR I=38 TO 20 STEP-1
160 HLIN 15,25 AT I
```

```
170 NEXT I
180 END
```

（五）读懂下列程序，你能想象出运行它的时候会在屏幕上出现什么图形吗？

```
10  GR
20  FOR W=3 TO 50
30  FOR I=1 TO 19
40  FOR J=0 TO 19
50  K=I+J
60  COLOR=J*3/(I+3)+I*W/12
70  PLOT I,K:PLOT K,I
80  PLOT 40-I,40-K
90  PLOT 40-K,40-I
100 PLOT K,40-I:PLOT 40-I,K
110 PLOT I,40-K:PLOT 40-K,I
120 NEXT J,I,W
130 END
```

（六）利用椭圆方程

$$\begin{cases} x = a\cos\alpha \\ y = b\sin\alpha \end{cases}$$

用高清晰度作图方式画出一个椭圆来。其中a取90，b取40，建议每4°取一点再连成曲线。要求圆心要在屏幕中间。

（七）下列数据是一个数字8的图形定义，要求编写一个程序，完成下列事项：

（1）建立一个图形表；

（2）把图形表送入内存24576开始的地址中；

（3）把图形8显示并让其旋转放大（但不要让每个8都留在屏上，只希望出现一个8字）

这个图形定义的数据是：

45，53，54，63，39，36，36，44，45，54，6，0。

（八）下列程序运行时，在屏幕上先后出现什么图形？

```
10 HGR
20 HCOLOR = 6
30 FOR I = 0 TO 50
40 HPLOT 150 - I * 1.6,30 + I * 1.2 TO 150 + I * 1.6,30 + I * 1.2
50 NEXT I
60 FOR I = 50 TO 0 STEP - 1
70 HPLOT 150 - I*1.6,150 - I*1.2 TO 150 + I*1.6,150 - I*1.2
80 NEXT I
90 FOR I = 1 TO 1000 : NEXT I
100 HGR2
110 HCOLOR = 3
120 FOR I = 0 TO 16
130 HPLOT 4,8 + I * 11 TO 274,8 + I * 11
140 HPLOT 3 + I * 17,9 TO 3 + I * 17,183
150 NEXT I
160 FOR I = 1 TO 1000 : NEXT I
170 TEX : GR
180 COLOR = 12
190 FOR I = 0 TO 39
200 VLIN 0,39 AT I
210 NEXT
220 COLOR = 11
230 FOR I = 0 TO 15
240 HLIN 20 - I,20 + I AT 15 + I
250 NEXT
260 EOR J = 1 TO 1000
270 NEXT
280 POKE 49232,0 : POKE 49239,0
290 POKE 49236,0 : POKE 49234,0
300 FOR I = 0 TO 1000 : NEXT
```

```
310 POKE 49237,0
320 FOR I=0 TO 1000:NEXT
330 POKE 49232,1
340 POKE 49238,0
350 POKE 49236,0
360 POKE 49235,0
370 FOR I=1 TO 1000:NEXT
380 TEXT
400 END
```

（九）自行设计一个图案（譬如两个汉字，或一只熊猫等），用低清晰度绘图语句写一个程序，使之运行时能在屏上显示出来。

（十）自行设计一个图案（譬如天安门城楼、红五角星等），用高清晰度图形绘图语句写一个程序，以便在运行时能在屏幕上显示出来。

（十一）将图形方法一节中最后提供的那个辅助设计程序输入机器，按你的需要建立一个图形定义及图形表。最后写一个程序，使之运行时能显示出你设计的图形来。

第九章　唱歌与游戏程序的编写

苹果机主机箱内装有一只喇叭，用程序控制它可以发出声音，甚至奏出曲子来。此外，苹果机也配有一组游戏控制器，在程序中反复访问它们，可以把游戏者操纵它们时的有关信息读入，从而使屏上的图形发生变化，再配上声音，可以实现很好的游戏效果。这对广大青少年来说无疑有着相当大的吸引力，从而也能更加激起他们的学习兴趣。

本章介绍的保留字只有四个，即 SPEED=，GET，PDL 和 CALL。

§1．唱歌程序的编写

所谓唱歌，在这里只是奏曲子，并不唱出歌词；而且曲子也是单音的，没有和声。在苹果机上唱出歌词或奏出和声，也是可以实现的，这需要添置另外的硬件。我们这里介绍的方法，无需增加任何设备，是由程序来完成的。

一、发音专用单元

苹果机内装的喇叭，是由一个触发器推动的。当触发器输出高电位时，喇叭的放大电路将使其中有电流通过。大家知道，喇叭的线圈中有电流通过时，线圈将产生运动，从而带动喇叭的纸盆也跟着运动，这就使纸盆周围的空气振动，发出我们可以听得见的“咔啦”声；而当触发器输出低电位时，放大电路就使喇叭线圈中没有电流通过，线圈又恢复到原来的位置，这又使喇叭发出声响。如果适当控制触发器输出高电位和低电位的交换频率，将

使喇叭纸盆产生不同频率的振动，从而发出不同的音来。这就是苹果机发音唱歌的基本原理。

具体地说，在苹果机内控制喇叭的触发器的地址是49200(十六进制的 C030，以后我们在一个数字之前加上 $， 表示这个数是十六进制数。在此可写成 $C030)。 这也是一个专用单元（喇叭专用单元)，它象个喇叭按钮开关,往这个单元写一个数或从中读一个数，相当于按了一次开关，发出一个声音。读写什么数是无关紧要的。实际上对这个单元读写数的目的，是使这个单元的地址线上产生一个高电位方波（从低电位变成高电位，再变成低电位),经放大后送入喇叭线圈，使喇叭纸盆运动发出声音来。

因此，下列PEEK(49200)函数的使用， 将会使喇叭发声：

```
PRINT  PEEK (49200)
```

或程序段：

```
10  FOR I=1 TO 100
20  P=PEEK (49200)
30  NEXT
40  END
```

或程序段：

```
10  A=PEEK (49200)-PEEK (49200)+PEEK (49200)-PEEK
    (49200)+PEEK (49200)-PEEK (49200)
20  END
```

在上面三种使用方法中，第一种使用方法，发声时间很短。不注意可能听不到。第二种使用方法，通过循环语句重复100次也就是使喇叭振动100次，声音是可以听到的。但声音较低沉。第三种使用方法，频率稍高一点，但改进不大(注意,这里采用一加一减的方法，可以防止P的溢出。本来P的值就无意义)。

声音之所以低沉，是由于运行BASIC程序时的解释过程造成的。让喇叭振动一次只要一条机器指令就能完成，而解释一个

PEEK 或 POKE至少要执行数百条乃至数千条指令。因此，要想发出悦耳动听的音律，仅使用PEEK或POKE是绝对办不到的。

要想唱出一支歌来，必须求救于机器语言程序。因为它不需要解释过程，可以使喇叭的振动频率大大提高。

为此，这里专门设计了一个用机器语言写的发音子程序，只要告诉它音阶的高低和节拍的长短，它就能发出那个声音来。

二、机器语言发音子程序简介

机器语言发音子程序框图如图9—1所示：

框图中的A和B，是BASIC语言中的两个变量。A用来控制音阶的高低，B用来控制节拍的长短。X和Y是6502微处理器的两个寄存器，长度都是八位，因此它们可以寄存的数据只能在0到255（即＄0到 ＄FF）的范围内。它们有加1和减1的功能。当它们由255再加1时变成了0，而由0减1时则变成了255。

现在我们来看一下这个框图。我们暂且忽略图中右边那一部分，也就是说无论Y是否为0，都让它直接作X－1→X的操作（相当于当Y＝0时，走虚线）。

进入这个子程序时，X中的数是随机的，但由于后边的减1操作(X－1→X)，X总会变为0的。让我们从X＝0 说起。此时，程序应走框图中最下一个功能框，使X中得到变量A 的值。然后程序顺大圈线返到头上使喇叭振动一次。接着程序反复执行小圈内的程序(从中间往下，顺小圈线返上)，每转一圈X的值减去1。直到X＝0 时，X再取A值，重复上述过程。由此可见，喇叭两次振动之间的时间间隔，是由A 控制的：A 值大时，转小圈次数要很多次才能出现一次X＝0，使喇叭振一次。也就是说，A值大时，喇叭振动的周期长，频率低。反过来，A 值小时，喇叭振动的周期短，频率高。所以A 的大小可以控制喇叭振动的频率。因此选取适当的A值，这个子程序会发出不同音阶的声音来。

现在我们再把框图中右边那一部分考虑进去。Y 的值开始也

是随机的，但由于不断地执行减1操作，它也总会为0的。在这以

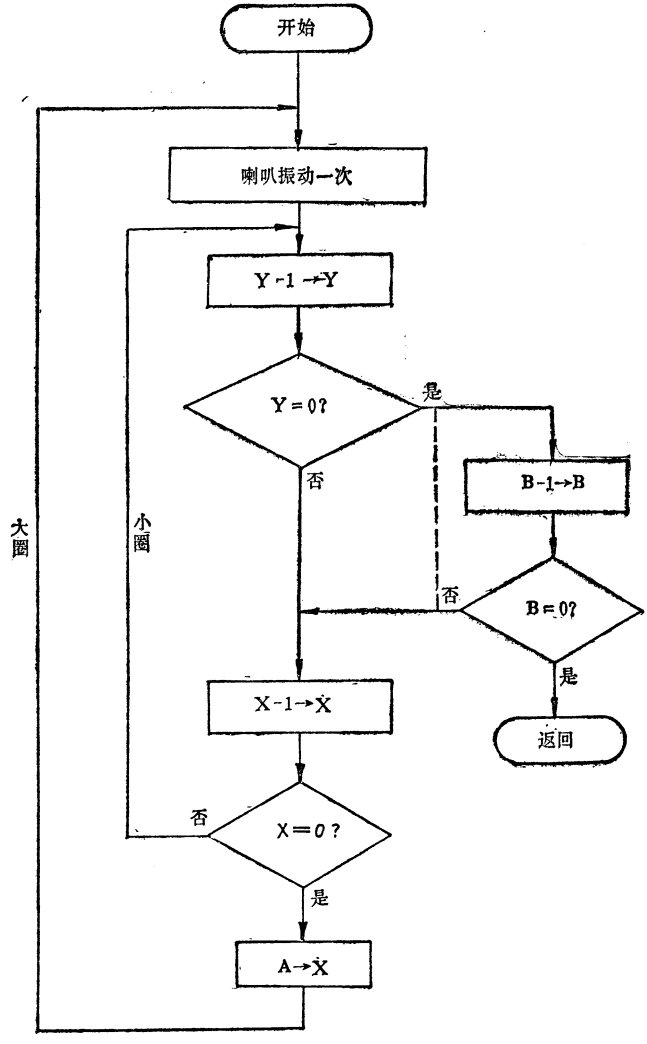


图9—1

后，继续执行程序时，Y的值总是由255逐次减1再变为0。在Y从

0再次变成0时，程序反复兜过256个圈子。其间有大圈，也有小圈，时间稍有不同(走大圈时间稍长)，但总的看每转256圈的时间是很接近的，我们可视为一个固定的时间。这样框图右部的功能就是：每转256圈执行一次B－1→B的操作，并判断B是否为0：不为0返回去继续执行；为0退出子程序。于是B值可用来控制程序转多少个256圈。由于256圈时间是固定的，给出适当的B值，子程序发出的声音就可以是半拍或一拍。

因此，只要给出一个A值和一个B值，子程序就会发出一个声音来。其高低受A值控制，长短受B值控制。

三、机器语言子程序的建立与使用

机器语言发音子程序必须送入机器并使之运行才能发出声音。为安全起见，我们通常把它存放在内存的770号开始的单元中（这里有一片内存区，不会被一般程序运行时所破坏)。另外，由于机器语言中不能使用变量名，变量的值必须经由POKE语句送入某个单元中，因此我们也常把A和B分别送到内存的768和769号单元中（相应地，前面框图中的A和B，最好分别改写成768和769。这里我们约定：在机器语言框图中，这样写出的数，用来表明地址号码，而不指明数本身。数本身可在前面加＃号标出)。

在作了这些考虑后，这个机器语言子程序，用十进制数写出来，将是以下21个数据：

173，48，192，136，208，5，206，1，3，240，9，202，208，245，174，0，3，76，2，3，96

关于这些数据的意义，我们将在第十一章中作进一步说明，这里我们只要知道这21个数是用十进制写出的机器语言发音子程序就行了。

按照我们上面的考虑，下列程序可将这21个数送入机器内存中(建立机器语言发音子程序)：

```
10 FOR I=0 TO 20
```

```
20  READ N
30  POKE I+770, N
40  NEXT
50  DATA 173, 48, 192, 136, 208, 5, 206, 1, 3, 240, 9,
    202, 208, 245, 174, 0, 3, 76, 2, 3, 96
```

内存中有了这个子程序，只要用POKE 语句将控制音阶和节拍的值分别送入768、769号单元中，运行这个子程序就可以发出一个有一定高度、一定长短的声音。

那么，在 BASIC 中如何运行一个机器语言子程序呢？这要用CALL命令。

CALL的格式是：

〔行号〕 CALL 算术表达式

算术表达式应是机器语言子程序的入口地址。因此，在768号单元和769号单元得到适当的数据后，命令

CALL 770

的执行，会发出相应的声音。

CALL非常类似于GOSUB。它们的区别在于GOSUB是转去执行一段BASIC程序，GOSUB保留字之后跟着一个行号。而CALL 则是转去执行一个机器语言子程序，保留字CALL 之后跟着机器语言子程序的入口地址。子程序返回来后，都是执行后续语句。

四、音阶和节拍参考值

控制音阶高低和节拍长短的值，到底如何选取呢？这里提供一些参考数据。实际计算比较麻烦，也没有必要。

音阶	控制值	节拍	控制值
1	192	十六分音符	30
2	172	八分音符	70
3	152	八分音符加附点	110
4	144	四分音符	160

5	128	四分音符加附点	255
6	114		
7	102		
$\dot{1}$	96		
0	1		

中音1(do) 取192，而高音$\dot{1}$(do) 则取96 (=192/2)，可见音阶每提高八度，音阶控制值减小一半，反之音阶每降低八度，音阶控制值提高一倍。这样可以推算出未列出的音阶的控制值。

由于这两个控制值都将被送入机器内存的单元中，它们的值也只能在0到255的范围内。这样低于$\dot{5}$和长于一拍半的声音，就不能实现了。对于低于$\dot{5}$的音，我们只好把它提高八度；而对于两拍的长度也只好改为两个一拍。

一般用这个子程序奏一些活泼欢快的曲子，效果较好。

五、唱歌程序的构成

编写唱歌程序时，最好先抄出曲谱，然后在每个音阶下标出两个数：音阶值和节拍值。比如 | $\underline{1}$ 1 $\underline{2}$ | 3 3 | 可写成：

$\underline{1}$	1	$\underline{2}$	3	3
192	192	172	152	152
70	160	70	160	160

这样，下列程序的运行，就会达到目的：

```
100 READ I, J
110 IF J=0 THEN END
120 POKE 768, I : POKE 769, J
130 CALL 770
140 GOTO 100
200 DATA 192, 70, 192, 160, 172, 70, 152, 160, 152,
    160, 0, 0
```

这个程序中的DATA，列出了歌谱信息，它们是两两一组的，

不可搞错（少输入一个数，就会把节拍控制值当音调控制值使用，发出难听的声音，也就是走调）。最后加两个0作为结束标志。

我们运行这个程序能唱出歌来，不要忘记这是已将唱歌子程序建立起来才实现的。如果没有那个子程序，执行CALL 770 一句，一般会执行不下去而出错。所以，一定要先建立这个机器语言子程序，然后才能用 CALL 调用它。有时我们干脆把它们合在一个程序中：程序的开头部分是建立这个子程序的程序段，在后续程序中就可以使用CALL 发声。

六、机器语言子程序的存贮与调用

机器语言子程序也可以存入磁盘上，需要时再装入内存来。

存入磁盘用BSAVE命令，从磁盘装入内存用BLOAD命令。

假定我们把这个子程序起名为MUSIC，这两个命令就可以分别写成：

```
BSAVE  MUSIC, A770, L21
BLOAD  MUSIC
```

七、发音子程序的其它应用

机器语言发音子程序，主要用于唱歌，但也可以灵活运用。请看下面的程序段：

```
100  FOR I=200 TO 20  STEP-5
110  POKE 768, I: POKE 769, 10
120  CALL 770
130  NEXT
```

当机器语言发音程序建立后，运行这个程序时，将以等长的节拍发出从低到高的声音，像竖琴发出来的一样。当然你也可以想象成往杯子里倒水的声音。如果你在屏幕上画出一只杯子，用另一种颜色在其中逐次向上画水平线，这又可以想象成往杯子里倒水。把声音的由低变高和杯中水面的不断上升结合在一起，效果更为理想。下面的程序就可以完成这件事。

```
10  GR
20  COLOR = 6
30  VLIN 15, 39 AT 14
40  HLIN 15, 25 AT 39
50  VLIN 15, 39 AT 26
60  COLOR = 9
70  FOR I = 1 TO 14
80  HLIN 15, 25 AT 39 - I
90  POKE 768, 200 - I * 14 : POKE 769, 20
100 CALL 770
120 FOR J = 1 TO 30 : NEXT J
130 NEXT I
140 END
```

§2. 游戏程序的编写

BASIC语言的人机对话功能，为编写游戏程序提供了可能性。因为机器可以在程序运行的适当时刻向用户（在此就是游戏者）提出某些问题，要求用户回答。程序可以根据回答的内容作出判断，或者转向不同的分支，或者给用户以必要的反馈信息。

在一般计算机上编写游戏程序时，受到了一定的限制。一方面没有专门的游戏控制器，从键盘上瞬间输入连续量不易实现；另一方面是游戏现场不大容易显示，特别是随时变化着的现场。

苹果机有一组专门的游戏控制器，而且有丰富的图形功能，为编写游戏程序提供了强有力的手段。

正因为这样，苹果机的游戏程序特别多。然而看懂这些程序并不是一件很容易的事情。我们这里要介绍的，也只是一些一般方法，不可能介绍编写游戏程序的所有方法，特别是用机器语言编写游戏程序的方法。

一、游戏程序的特点

游戏程序，可以是一个人独玩，也可以是两个人的对垒（或者是人与机器对垒），还可以是几个人的比赛。

一般讲，一个好的游戏程序，应具备下面一些内容和特点：

1.必须在显示屏上有图形显示，以形象地显示游戏的现场状况。如若是下棋游戏，必须显示棋盘和棋子，还要显示棋子的变化情况；如果是飞机轰炸坦克，必须有移动着的飞机、坦克图形，飞机抛出炸弹后，空中应有炸弹下落的图形，炸弹爆炸时最好还应显示爆炸的样子甚至发出声音。

2.必须有一个较为完备的算法（即游戏规则）。好比下中国象棋，马走日，象（相）走田，小兵（卒）不能后退……。飞机扔炸弹，炸弹只能下落不能横飞，碰到地面要炸开；坦克只能平走，不能跳起来……。算法的好坏，直接影响游戏的效果。

3.程序一般应能随时允许用户输入信息，并不断根据用户的输入信息作出反应。例如飞机扔炸弹，应该是游戏者认为需扔出时，能立即通知机器（下达命令），这时在飞机的下方应出现一颗炸弹。而不应该是机器允许你输入时才下达这种抛弹命令。

4.应有一些必要的说明和提示，用以告诉用户这个程序如何玩法或此刻你该做什么。还应能给出游戏者成绩评价：可以给分数，也可以给出评语，甚至给出一些建议或鼓励性的话。

5.必要时，应给出一些声音：一个爆炸声，一个撞击声，甚至一首赞歌或一支哀乐。

二、关于提示和说明

对一个普通游戏者来说，我们不能要求他读懂一个游戏程序、或看完一份游戏程序说明书之后再来运行这个程序。所以对于一个好的游戏程序来说，应在运行过程中给用户一些必要的提示和说明。比如运行一开始，应在屏幕上告诉用户：这是一个什么程序，如何玩法等。用户立即知道自己该干什么。有些程序的玩法

可能要复杂些，那就等到需要用户干什么事时再作提示。

对于游戏者给出的错误回答，必须给出提示信息，指明正确的回答应该是什么。

对于成绩的评定，可以做完一次给一个分数，也可以是作完给定次数后给平均分，还可以把屡次得分记下来进行分析比较，作出进步还是退步了的评语。甚至可以对几个游戏者的成绩进行总评，给出名次来。

程序中给出提示的方法很简单，就是使用PRINT语句。在图形和文本混合方式下，这些PRINT输出只能在下端四个文本行上。如果嫌位置太小，可用TEXT 命令转回文本方式，整个屏幕均可用作提示和说明。这时最好能用前一章介绍的控制输出位置的语句，把屏幕布置得好看一些。

对于一些特别需要用户注意的显示内容，应在这个PRINT语句之前加入一个 FLASH 命令，使其后的输出呈闪烁状态。闪烁输出的PRINT之后，要用 NORMAL 命令使其后的输出返回正常状态。

比如，希望输出闪烁的“GOOD”，这个语句在100行上，那么加入下列99行和101行就行了：

```
99   FLASH
100  PRINT  "GOOD"
101  NORMAL
```

有时，为了让用户看的更清楚些（特别是对于较长的输出显示），可以用SPEED = 语句，以放慢输出的速度。

SPEED = 的格式是：

〔行号〕 SPEED = 算术表达式

算术表达式的取值范围为0到255。

这个语句的功能是用来设定输出输入设置和内存交换信息的速度的。算术表达式的值为0时速度最慢；为255时，速度最快。

比如下列程序：

```
10  SPEED = 255
20  FOR I = 65 TO 90
30  PRINT CHR$ (I);
40  NEXT I
50  PRINT
60  SPEED = 0
70  FOR I = 48 TO 57
80  PRINT CHR$ (I) ;
90  NEXT
100 PRINT
110 END
```

程序运行时，先以最快的速度依次显示从A到Z的26个英文大写字母，然后以最慢的速度显示0—9这10个数字。

不使用SPEED时，机器自动取最大速度（相当于SPEED = 255）。

三、对游戏者输入信息的查询

游戏者向机器送入信息，有两种手段：一是键盘输入，二是游戏控制器输入。现分别介绍对这两类输入信息的查询方法。

1.对键盘输入的查询

(1) 对INPUT语句的查询

一个游戏程序中，当着需要用户输入某些信息时，可以设置INPUT语句。程序运行中遇到这个INPUT语句时，就等待用户的输入。用户输入的信息，被保存在INPUT保留字后随的变量中。以后查询这些变量，便可知用户输入的是什么信息。

游戏程序中，一般要求用户的回答都比较简单，如YES, NO, 或0, 1, 2, …，不大需要较长的字符串或多于两位的数字。在这种情况下，对于字符串输入，经常使用LEFT $函数。

比如程序段：

```
100  INPUT  "YES OR NO" ; A$
110  IF LEFT$ (A$, 1) = "Y" THEN 3000
120  IF LEFT$ (A$, 1) = "N" THEN 2000
130  GOTO 100
 ⋮
```

这个程序的2000，3000分别为两个程序段（具有某种独立功能）的首行号，在此从略。

在110行和120行，我们用了LEFT $ 函数。这样，只要用户对INPUT的回答中首字符是“Y”，就当YES处理，转3000 行去；同样，只要对INPUT回答中首字符是“N”，则当NO处理，转2000行去。换言之，回答“NOT”和回答“NO”以及回答“N”的效果一样。如果有人误把“YES”记成了“YIS”，机器也会当“YES”处理。

此外，INPUT语句特别适合于对多种情况的处理。

例如下列程序段：

```
100  INPUT "1, 2, 3, 4"; N
110  IF N<1 OR N>4 THEN 100
120  ON N GOTO 1000, 3000, 2000, 3500
 ⋮
```

当输入 1 时，程序转1000行完成某件事；当输入2时，程序转2000行完成另一件事…。当输入小于1或大于4时机器不知道干什么，就返回去等待用户重新输入（在前一个例子中的130行，也是这个意思）。

(2) 对GET语句的查询

GET是一个新语句，它的格式是：

行号 GET 变量名

这个语句也是键盘输入语句，与INPUT语句相同之处是：程序运行中遇到它时，同样等待用户由键盘给那个变量输入数据。

但它与INPUT语句又有不同：

第一，对GET回答时，键盘上输入的字符将不被显示；

第二，GET只接收一个字符。如果GET后边是个字符串变量名A$，而你回答的是“ABC”，那么A$得到的只是字符“A”。类似地，如果对GET A，你回答“123”，那么A得到的是数字1。

第三，GET语句只要接收到一个字符的回答后，无需按回车键，就自动执行后续语句了。因此对于GET回答时，如有错误，不能用左移键返回重新输入。甚至你对GET回答的第一个字符就是左移键，它也当作输入接收进去，使程序执行后续语句。

第四，GET语句本来不是为接收数型量设计的，但如果输入数字尚可利用。若是输入+、-，E等，与输入0一样；如输入“.”、回车键或其它非数字字符，将认为是错误的。

因此使用GET时，要小心一些。

GET的使用与查询，与INPUT一样。

(3) 对键盘接收单元的查询

苹果机内存中有专用的键盘接收单元，任何从键盘上输入的字符，首先进入这个单元。这个单元的地址号码是49152（$C000)。它存放的是键盘字符的ASCII码。对这个单元进行查询，可以知道用户敲了什么键。

因此，在每次由键盘输入之前，要对这个单元清除。对这个单元的清除也很特别：它是利用另外一个称为清键单元的专用单元实现的。这个清键单元的地址号是49168（$C010)。这也是一个按钮开关式的单元，只要给这个单元送入一个数，就能起到清键作用。通常我们都用

```
POKE 49168, 0
```

这种方式。它的清键作用并非把键盘接收单元清成全0，而是将其最高位清0，使键盘接收单元中的数小于128。这是因为键盘字符的ASCII码都大于等于128的缘故，所以从最高位可以判断出有没

有输入（127是二进制的01111111，128是二进制的10000000）。

现在举个例子来说明键盘接收单元的查询方法。

从ASCII码表中可以查出：Space（空棒）的键盘ASCII码为160，ESC的键盘ASCII码为155，RETURN（回车键）的键盘ASCII码为141。这样，设程序中遇到输入为空格时，转2000行做一件事；输入为RETURN时，转1000行做另一件事；输入为ESC时使程序结束，显示“BYE”。下列程序段就可以完成这些工作：

```
⋮
100  POKE 49168, 0
110  P=PEEK (49152)
120  IF P<128 THEN 110
130  IF P=141 THEN 1000
140  IF P=160 THEN 2000
150  IF P=155 THEN HOME : PRINT "BYE" : END
160  GOTO 100
⋮
1000
⋮
1999 GOTO 100
2000
⋮
⋮
2990 GOTO 100
```

程序的100行就是清键工作。110行把键盘接收单元的内容读入P中。120行的作用是：当尚未接收键盘输入时，转110行，重新读键盘接收单元。而160行则表明，虽然有键盘输入，但不是程序需要的字符，因而重新清键，再检查新的输入。

对于键盘接收单元的查询，也只能接收键盘上输入的一个字符，它也不需要敲回车键表示输入的结束，而且输入的字符不在屏

幕上显示。这些都具有GET的特点。但它不受输入数据类型的限制，而且它接收的字符更广泛，广泛到键盘上的任何一个键，包括回车键。这在INPUT和GET中是无法实现的。因此在游戏程序中广泛采用这种键盘输入的查询方法。

这里我们再向读者介绍一下负地址的问题。内存中的地址编号是从0号开始，一直编到65535（对于RAM为48K 的系统），如果从65535号起往0号数，我们用负地址的表示方法。换句话说，65535号单元就是－1号单元，65534号单元就是－2号单元，……于是，清键单元的负地址是－16368，而键盘接收单元的负地址是－16384。用负地址改写上面程序中的前两行，就变成了：

```
100  POKE－16368，0
110  P＝PEEK (－16384)
```

在许多游戏程序里经常这样使用。

对键盘输入的查询，主要有以上三种方法。这三种方法的共同特点是，只能查询来自键盘的输入。不同之处是：对于前二者（INPUT和GET）总是要接收到输入后，程序才能继续运行，否则机器将处于等待状态；第三种方法不等待，查不到输入也可以使程序下滑。

2.对游戏控制器输入的查询

游戏控制器象一只黑盒子，上面有一只按钮开关和一个旋钮，游戏时，游戏者可通过它们向机器输入信息。近来也有改用游戏操纵杆的，原理是一样的。我们仍按盒子介绍。

（1）对按钮开关的查询

游戏控制器上的按钮开关，时刻对应着内存中的一个固定单元。当未按下此开关时，它所对应的单元为0值；当按钮按下时，它所对应的单元为255（全1）。因此，对有关单元进行查询，可以知道你是否按下按钮开关。

一部苹果机可以同时带三个这样的按钮开关，它们所对应的

内存单元分别是：

$SW_0 = 49249$ (-16287, $C061)
$SW_1 = 49250$ (-16286, $C062)
$SW_2 = 49251$ (-16285, $C063)

如果想写一个飞机炸坦克的游戏程序。用绘图语句在屏幕的上方和下方分别画出一个移动着的飞机和坦克，不是十分困难的(下面还会具体介绍)。假定能够做到：飞机和坦克在不断运动中，游戏者可以根据它们的相关位置，在某一时刻按一下按钮，表示投下一枚炸弹。程序运行中查询到“按了按钮”这一信息之后，也能在飞机的下方出现一个代表炸弹的不断下落的形象。当着这个形象碰到代表坦克的图形时或碰到地面时，应产生爆炸，……这样的游戏程序才有点意思。这里我们提供一个程序轮廓，示意怎样对按钮单元进行查询。

这个程序轮廓可以是这样(坦克暂作固定目标)：

```
⋮   屏幕顶部某侧显示一飞机，下方显示一坦克
100 T=0
110 P=PEEK (-16287)
120 IF P<128 THEN 350
130 在飞机当前位置的下沿显示一炸弹图形
⋮
200 T=1 (表示空中有炸弹)
210 炸弹是否碰到坦克？如未碰到转250
220 (这时炸弹命中坦克) 在坦克处显示爆炸图形、发出爆炸声音，
    消去坦克和炸弹图形
230 T=0 : GOTO 350
250 炸弹是否碰到地面？如未碰到，转350
270 类似220行，但不消去坦克图形。
300 T=0
```

```
350 ⎫
 ⋮  ⎬ 飞机向前移动一格
    ⎭
400  IF T=0 THEN 110
410  炸弹下落一格
500  GOTO  210
```

这个轮廓中的110行就是用来对0号按钮进行检索的。120行的条件表达式P＜128，条件很宽裕。改用P＝0或P＜＞255也可以，但条件很苛刻。

在上面轮廓中可以看出：不管你是否按了按钮，飞机总是向前飞行的（如果用INPUT语句或GET语句接收来自键盘的按下按钮信息，飞机无论如何要停下来等待你的输入。你几个小时不按键，飞机可以在那里停几个小时。这显然是不好的），而且不论空中有无炸弹，飞机也总是向前飞行的。

这个轮廓是按这样一个想法设计的：只要投下一枚炸弹，在它落地开花之前不允许再投第二枚炸弹。如果不作这种限制，可以删去400行，把500行改为GOTO 110，这时轮廓中的其它部分也要修改。因为这时空中可以存在多发同时下落的炸弹，因素更多，程序也会复杂得多。但基本处理方法是一样的。

（2）对于旋钮的查询

游戏控制器上的旋钮，带动一个150kΩ的电位器，＋5V电压经过电位器接到机内芯片上，通过旋钮改变电位器的阻值，由电压频率转换器输入到机内相应内存中。阻值不同，内存中对应单元的值也跟着改变，最大为255（全1），最小为0（全0）。

机器允许带四个旋钮。它们所对应的内存单元分别是：

PDL_0 = 49252 （$C064，－16284）

PDL_1 = 49253 （$C065，－16283）

PDL_2 = 49254 （$C066，－16282）

PDL_3 = 49255 （$C067，－16281）

对旋钮的查询，虽然也可以访问这些专用单元，但苹果BASIC提供了一个专用的函数，可用来完成这一任务。这就是PDL函数。

PDL函数的格式是：

PDL（算术表达式）

其中算术表达式的值表示旋钮号码。因为可接4个旋钮，通常算术表达式的值应取0，1，2或3四个数之一。从语法上讲，算术表达式的值可以从0到255。但由于4到255不对应任何旋钮，使用它们会产生不可预测的副作用，建议不要使用。

PDL函数的值，是给定号码的那个旋钮专用单元中的数。它总是0到255范围内的一个整数。

旋钮的转动，常可用来控制某些图形的移动。随着转角的不同，对应单元的值也不同。在程序中定时地从中读出数据，可以作为图形位置的参数。从而实现通过旋钮的转动来控制图形的移动。

举例来说，假如设计一个打乒乓球的游戏程序。在屏幕的某侧，应绘制一个代表球拍的图形（通常用一短垂线），它可以跟着游戏者转动旋钮而上下移动。下面的程序轮廓，就用来示意这种方法。为了象个游戏，我们假定在屏幕上有个运动着的乒乓球，当它碰到左侧的球拍时，认为成功一次；当球碰到左壁，算失败一次；当乒乓球碰到球拍或碰到右墙，都应有反弹回去的功能。球拍只作上下的运动，由游戏者控制，目的是想把球挡回去不让它碰左墙（成功、失败就是从这个意义上讲的）。

在下面的程序中，我们只用PDL（0），表明只查0号旋钮。又由于旋钮对应单元的值最小为0，最大为255。用255作为纵坐标值无论在那种图形方式下也是太大的。所以通常总不是直接用PDL值，而是将它乘以一个比例因子，作为图形坐标参数。这个比例因子应使所控制的图形落在屏幕内，有时还要求能复盖它所能涉及的全部活动范围（对这个例子而言，球拍应移不出屏幕，又要能上下移动到顶部或底部）。

程序的要点如下：

```
100  T=1：K=0：L=0
110  在屏上随机地出现一个球
120  VTAB21：PRINT K, L
130  P=PDL(0)：P=P*Q
     根据P的值，在0列相应位置上显示球拍
200  根据T值使球左移或右移1格（T=1左移，T=-1右移）
300  判断球是否碰击右墙?如碰上了,则使T取1:发出碰墙声:GOTO
     130
400  判断球是否碰击球拍子上，如没有转500
450  K=K+1：T=-T：发出声音：GOTO 120
500  球是否碰左墙？若不是，转130
550  L=L+1：发出碰墙声：GOTO 110
```

这个程序的120行，打印成功和失败的次数；130行依PDL(0)值改变球拍位置。

下面我们把130行具体化些。设在低清晰度图形方式下，球拍为上下5个色块。由于球拍下端纵坐标最大为39，从而球拍上端最大纵横为35。用P值作为拍子上端的纵坐标值，130行可改写成

```
130  P=PDL (0)：P=INT (P/7.2)
...
```

新的P值总保持在0到35的范围内，从而保证球拍能在屏幕内从上到下移动，既不会出现移不到的地方，也不会移出屏幕。

PDL的值虽然在程序中是间隔一段时间再来查询一次所得到的，但在存放那个数据单元中的数却随时随刻依着游戏者转动旋钮而改变着。所以它实际上是一种实时的模拟量的输入。如果把游戏控制改作实时控制和实时数据采集，经少许改动也是可以的。

四、图形显示

游戏中的图形显示，主要应体现游戏中变化着的游戏现场。

用低清晰度绘图语句和高清晰度绘图语句绘制游戏现场，一般情况下并不太困难。这里进一步说明以下几点：

1.一个点（或多个点）的出现与消失

一个用低清晰度绘图语句写的棋类游戏，可以用一个色块代表一个棋子。两人下棋，可用两种颜色的色块，一种颜色代表一方。当甲放在某个位置上一个棋子时，就用COLOR=，给出甲方的颜色，再用PLOT语句在相应的位置上画出一个色块来。当甲方吃掉了乙方某个棋子，就用COLOR=0，再用PLOT语句将相应位置上的色块涂掉。当然棋盘底色可以不是0号颜色，那时要用棋盘底色来涂掉这个点。

2.一个色点的移动

一个色点的移动，在游戏中经常用到。表示一个运动着的乒乓球、一个飞行着的枪弹、一个下落着的炸弹等，均可用一个色点的移动来实现。

在程序中实现的方法，一般是先画点的出现，然后把这个点的坐标由另两个变量保存下来，以便下次返回来时，用底色把它涂掉。

比如下面的程序段：

```
100  X=0:Y=0
110  GR
120  COLOR=0
130  PLOT M, N
140  COLOR=6
150  PLOT X, Y
160  M=X:N=Y
170  X=X+1:Y=Y+1
 ⋮
500  GOTO 120
```

这样一段程序可以使色点从左上角跑到右下角。它在X方向

和Y方向上是均速变化的。对170行的表达式作一些改变，色块的移动将会发生变化。比如：

```
170  X=X+1 : Y=Y+1.4
```

Y方向的速度要快些。

```
180  Y=Y+Y*0.5
```

X方向不变，Y方向的速度越来越快。

参照上一章第2节例5，不难写出点的变向运动来。

3.整个图形的移动或改变

整个图形的移动，虽然也可用屏幕底色全部将图形涂掉，然后再用适当的颜色在新的位置上画出来。但由于图形不只一个点，涂一遍、画一遍还是要时间的，特别是较大一点的图形。从效果上看，图形移动也不连贯。所以通常是只作图形的部分修改。也就是说，把图形移动中的每一个变化部分（涂掉一些点，增加一些点），用绘图语句来画出就行了。图上那些不变的部分，让它呆在那里不管。请别误会，设定颜色的语句COLOR和HCOLOR只给画点、画线语句指明颜色，而并不改变当前图形的颜色。

正是这个目的，有些图形应尽量画得规整些，以便移动时方便些。

4.色号的判别

确定一个炸弹是否炸到了坦克，或者确定一个球拍是否击到了球，这类判断检查，一般可用比较两个图形有无重叠的坐标点的办法。但其中某个图形点数较多时，这种判断检查也是很麻烦的，因为要比较的坐标点较多。而对低清晰度图形来说，函数SCRN可以有效地帮助你解决这个问题。

还以飞机炸坦克为例，设坦克是用4号颜色(深绿)画出的。那么判断炸弹是否炸中了坦克，可用条件式：

```
SCRN (X, Y) = 4
```

其中(X, Y)是炸弹的当前坐标位置。如果这个表达式为真，

表明炸中了；否则，没炸中。这比逐点比较就好多了。

5.关于用机器语言程序绘制图形

上章讲过，高清晰度图形的每个点，对应着内存中图形存贮区某个单元内的一位。用机器语言实现向内存某一单元放一个数，或使其中数的各位移动，要比 HPLOT 语句快得多。因此对一些要求图形变化较快的游戏程序，常用机器语言来实现。我们打算在第十二章里再作一些简单介绍。

五、关于音响

在游戏程序中，若能建立机器语言发音子程序，只要给出歌谱信息，在程序运行的适当时刻唱一支歌，是很容易的。

连续多次使用PEEK（－16336）也可以发出类似撞击的声音来。

六、游戏程序一例

这个游戏好象两人持枪对射。甲乙双方各持一枝手枪，每人射六发子弹，以击中对方的枪为目的。各自的枪可上下移动，由游戏旋钮控制，游戏按钮按下时表示射出一粒子弹。屏幕中部还有两个障碍物，子弹飞行中碰到障碍物，或未击中对方枪身，则不得分；子弹击中对方枪身得一分。六弹得5分者，为胜一局。这时在屏幕上显示一个墓碑，并奏出哀乐，以示对失败者治丧。然后再给双方发六发子弹，重新对射，若双方六发子弹全射完，仍无得5分者，也再发六发子弹，重新对射。当从键盘上敲入 ESC 键时，显示双方所胜局数，判定谁为胜者(或平局)，在屏幕上显示一枚勋章，以示奖给胜者。

程序稍长，但较好读：

```
0   GOTO 1000
10  COLOR = 14
20  FOR I = 2 TO 12 STEP 2
30  PLOT I, 38 : PLOT 40 - I, 38
```

```
40  FOR K = 1 TO 60 : NEXT K
50  NEXT I
60  R = 26 : T = 14
70  HOME
80  HTAB 10
90  PRINT "NUMBER OF TIMES SHOT"
100 HTAB 7 : PRINT F;
110 HTAB 33 : PRIN G
120 IF G = 5 THEN PRINT "BROWN WINS!!!!"
130  IF F = 5 THEN PRINT "WHITE WINS!!!!"
140  IF G = 5 OR F = 5 THEN 700
150  A = PDL (0) * 2/15
160  B = PDL (1) * 2/15
170  N = INT (RND (1) * 2)
180  COLOR = 12
190  VLIN 21, 28 AT 16 : VLIN 22, 24 AT 14
200  VLIN 21, 25 AT 18 : PLOT 15, 24 : PLOT 17, 25
210  VLIN 6, 12 AT 24 : VLIN 7, 10 AT 22
220  VLIN 6, 8 AT 26 : PLOT 23, 10 : PLOT 25, 8
300  REM GUNS
305  IF A = AO THEN 310
306  COLOR = 0 : PLOT 5, AO + 1 : HLIN 6, 7 AT AO :
     HLIN 6, 9 AT AO - 1
310  COLOR = 8 : PLOT 5, A + 1
320  VLIN A - 1, A AT 6
330  COLOR = 10 : HLIN 7,9 AT A - 1 : PLOT 7, A : AO = A
331  IF B = BO THEN 333
332  COLOR = 0 : PLOT 35, BO + 1 : HLIN 33, 34 AT BO :
     HLIN 31, 34 AT BO - 1 : K = 1
333  COLOR = 15 : PLOT 35, B + 1 : VLIN B - 1, B AT 34
```

```
334  COLOR = 10 : HLIN 31, 33 AT B - 1 : PLOT 33, B
335  BO = B
336  Y = PEEK (- 16286 - N) : IF Y<128 THEN 338
337  ON N + 1 GOTO 600, 500
338  X = PEEK (- 16286 - N) : IF X<128 THEN 340
339  ON N + 1 GOTO 500, 600
340  A = 35 - PDL (0) * 2/15
345  B = 35 - PDL (1) * 2/15
350  IF PEEK (- 16384) = 155 THEN 800
355  IF R<>38 OR T<>2 THEN 180
360  F = 0 : G = 0
365  HOME : HTAB 17 : PRINT "RELOAO"
370  FOR I = 1 TO 1000 : NEXT I
380  GOTO 10
500  IF T = 2 THEN 340
505  T = T - 2
510  FOR I = 1 TO 10
515  S = PEEK (- 16336) + PEEK (- 16336) - PEEK
     (- 16336) + PEEK (- 16336)
520  NEXT I
525  COLOR = 0 : PLOT T, 38
530  FOR I = 10 TO 34
535  IF R = 38 THEN 550
540  IF SCRN (I, A - 1)<>12 THEN 550
545  COLOR = 0 : PLOT I - 1, A - 1 : GOTO 340
550  COLOR = 15 : PLOT I, A - 1
555  COLOR = 0
560  IF I<>10 THEN PLOT I - 1, A - 1
565  NEXT I
570  PLOT I - 1, A - 1,
```

```
575  IF A=B OR A=B+1 OR A=B+2 THEN 585
580  GOTO 340
585  G=G+1
590  FOR K=1 TO 5
593  FOR I=1 TO 10
595  S=PEEK (-16336)
597  NEXT I, K
598  GOTO 365
600  IF R=38 THEN 340
605  R=R+2
610  FOR I=1 TO 10
615  S=PEEK (-16336) -PEEK (-16336) +PEEK
       (-16336) -PEEK (-16336)
620  NEXT I
625  COLOR=0 : PLOT R, 38
630  FOR I=30 TO 6 STEP-1
635  IF T=2 THEN 650
640  IF SCRN (I, B-1) <>12 THEN 650
645  COLOR=0 : PLOT I+1, B-1 : GOTO 340
650  COLOR=13 : PLOT I, B-1
655  COLOR=0
660  IF I<>30 THEN PLOT I+1, B-1
665  NEXT I
670  PLOT I+1, B-1
675  IF A=B OR B=A+1 OR B=A+2 THEN 685
680  GOTO 340
685  F=F+1
690  FOR K=1 TO 5
692  FOR I=1 TO 10
694  S=PEEK (-16336)
```

```
696  NEXT I, K
698  GOTO 365
700  GR
705  COLOR = 10
710  FOR I = 16 TO 28
715  VLIN 15, 39 AT I
720  NEXT I
725  COLOR = 0
730  FOR I = 17 TO 25 STEP 2
735  HLIN 19, 25 AT I
740  NEXT I
742  COLOR = 5
744  VLIN 6, 14 AT 22
746  HLIN 19, 25 AT 9
748  COLOR = 1
750  HLIN 16, 27 AT 38
752  HLIN 18, 25 AT 37
754  HLIN 20, 23 AT 36
756  PLOT 22, 35
758  COLOR = 12
760  FOR I = 13 TO 31 STEP 2
762  PLOT I, 39 : PLOT I, 38
764  NEXT I
765  COLOR = 12
766  HLIN 13, 31 AT 39
768  S = 100 : POKE 768, S : POKE 769, 255 : CALL 770
770  POKE 768, S : POKE 769, 255 : CALL 770
772  POKE 768, S : POKE 769, 127 : CALL 770
774  POKE 768, S : POKE 769, 255 : CALL 770
776  POKE 768, 85 : POKE 769, 190 : CALL 770
```

```
778  POKE 768, 90 : POKE 769, 127 : CALL 770
780  POKE 768, 90 : POKE 769, 127 : CALL 770
782  POKE 768, 100 : POKE 769, 110 : CALL 770
784  CALL 770
786  POKE 768, 105 : POKE 769, 110 : CALL 770
788  POKE 768, 100 : CALL 770
790  IF F=5 THEN FF=FF+1
792  IF G=5 THEN GG=GG+1
794  F=0 : G=0
795  INPUT "PLAY AGAIN?  (Y/N) " ;A$
796  IF LEFT$ (A$, 1) = "Y" THEN 799
797  IF LEFT$ (A$, 1) = "N" THEN 800
798  GOTO 795
799  HOME : GR : GOTO 10
800  GR
810  FOR I=15 TO 24
820  FOR J=10 TO 29
830  COLOR=INT (RND (1) *15+1)
840  PLOT I, J
850  NEXT J, I
860  COLOR=13
870  HLIN 19, 20 AT 30 : HLIN 18, 21 AT 31 : HLIN 17,
     22 AT 32 : HLIN 16, 23 AT 33 : HLIN 15, 24 AT 34
880  COLOR=13
890  HLIN 16, 23 AT 35 : HLIN 17, 22 AT 36 : HLIN 18,
     21 AT 37 : HLIN 19, 20 AT 38
900  HOME
910  VTAB 21 : PRINT "BROWN GUN WON" ; GG,
      "CAME" ;
920  IF GG<>1 THEN PRINT "S" ;
```

```
930  PRINT
940  PRINT "WHITE GUN WON" ; FF; "CAME" ;
950  IF FF<>1 THEN PRINT "S" ;
960  PRINT
970  IF FF<GG THEN PRINT "BROWN WON!!!!"
980  IF FF>GG THEN PRINT "WHITE WON!!!!"
990  PRINT "IT WAS ATIE!!!!"
995  VTAB 23 : HTAB 30 : POKE - 16368, 0 : END
1000 TEXT : HOME
1010 FOR I = 0 TO 20
1020 READ A
1030 POKE 770 + I, A
1040 NEXT I
1050 DATA 173, 48, 192, 136, 208, 5, 206, 1, 3, 240,
     9, 202, 208, 245, 174, 0, 3, 76, 2, 3, 96
1060 VTAB 7 : HTAB 14
1070 PRINT "* * * SHOOT OUT * * *"
1080 VTTA 10 : HTAB 11
1090 PRINT "THE OBJECT IS TO SHOOT"
1100 HTAB 11 : PRINT "THE OTHER GUY 5 TI MES"
1110 HTAB 13 : PRINT "USING THE PADDLES"
1120 FOR I = 1 TO 3000 : NEXT I
1130 VTAB 20 : HTAB 8
1140 PRINT "THE GAME IS AB OUT TO BEGIN"
1150 FOR I = 1 TO 1000 : NEXT
1160 AO = 2 : BO = 2
1170 POKE - 16298, 0
1180 GR : GOTO 10
```

程序中的336到339行，采取了一点措施，目的是使访问PDL时，使得0号和1号取得相同的优先机会（但又不是轮流来，而由

随机数控制它)。

一个游戏程序要写得好一点，并不是一件轻而易举的事。通常可以先剖析几个别人写好的程序，从中掌握一些要领和方法。如果自己能写出一个满意的游戏程序，那时会体会到：写一个游戏程序比一般计算程序收益要大。

游戏程序选题要好，还要注意知识性、趣味性。否则写出来用处不大。

§3. 趣味教学程序一例

用计算机进行辅助教学活动，是计算机应用的一个方面。

计算机辅助教学程序，有它自身的特点和规律。这里不讨论这些问题。

我们这里想介绍一个带游戏性质的教学程序。希望通过阅读这个程序，使读者体会一下这类程序的编写方法。

这个程序可用来让小学低年级同学或幼儿园的小朋友们在计算机上练习整数四则运算。算题由随机函数产生。加、减、乘两个操作数都是10以内的，除数也是10以内的，被除数在100以内。题目以立式（竖式）用低清晰度的图形显示。也就是说，每个数字和运算符号，很粗很醒目，而且是彩色的。题目的横线下还有一个闪烁的大问号，等待用键盘给出答案。答案正确时，还会显示一个人的笑脸；不正确时则显示一个哭脸，并让你重新回答。屏幕上可以显示三道题，做对一道再出下一道。这些题目也可以不用人回答，只要按 CTRL-P，机器会给出正确答案。如果不玩了，敲入SEIFT-S，可以结束。

关于数字的图形化，是这样实现的。它把 0 到 9 这十个数字分作两类，一类是用低清晰度绘图语句按笔画的布局，一笔一笔画出的，如1，4，7，8。另外几个数字则是先用某种颜色画一个

“8”字，然后用0号颜色涂掉某些部分而成的。比如把“8”字的右上方涂掉一小竖条，便成了“6”字。这样处理程序稍短，运行也快。

笑脸和哭脸，采用了另一种方法。由于HLIN语句可一次画出若干个点，就选用些语句实现。但HLIN要三个参数，这样，将相应的笑脸或哭脸的所有用HLIN语句画出的三个参数（相应于图块左上角的相对数值）找出，将它们三个一组地连接成一个字符串（当某个相对值大于9时，依ASCII顺序取后续字符，比如10用“:”，11用“;”,…）。真正使用HLIN画线时，再通过ASC函数将三个字符转换为三个参数。

数字、人脸以及运算符号、问号、横线，程序中都使用相对坐标位置，这样便于在屏幕的不同位置都可以使用。

程序中循环变量R，既用来控制循环次数，也用来控制显示的位置。变量LC用来控制文本窗口（底部四行）的显示位置。变量O的意义是：当它为真时由游戏者给出答案，为假时由机器给出正确答案。变量X，Y指明显示的绝对位置，就是给出显示图形的左上角的绝对坐标值。

现将程序列示如下：

```
10  HOME : O=1 : LC=1
20  PRINT
30  PRINT "·····        COLOR  MATH        ·····"
40  PRINT : PRINT
50  PRINT " I'LL  GIVE  YOU  THE  MATH
    PROBLEMS, AND  YOU  GIVE  ME THE ANSWERS.
    IF  YOU  GET  IT  RIGHT, I'LL  SMILE!"
55  FOR  I=1  TO  2000 : NEXT  I
60  GR
70  FOR  R=0  TO  24  STEP  12
80  A=INT  (RND (1) * 10)
```

```
90   B=INT (RND (1) * 10)
100  S=INT (RND (1)*4 )
110  Q=INT  (RND (1)*14)+1
120  Q=Q+(Q>9)
130  IF  Q=LQ  THEN  100
140  COLOR=Q : LQ=Q
150  IF  S=1  AND  A>B  THEN  170
160  D=A : A=B : B=D
170  IF  S<>3  THEN  230
180  B=B+NOT  B : A=A*B
190  X=R : Y=0 : N=INT (A/10)
200  COLOR=Q*NOT  NOT  N
210  GOSUB  4000
220  COLOR=Q : GOTO  240
230  Y=0
240  X=R+6 : N=A-(INT(A/10)*10)
250  GOSUB  4000
260  Y=8 : N=B
270  GOSUB  4000
280  X=R
290  ON  S+1  GOSUB  1020, 1022, 1024, 1026
300  HLIN  R, R+11  AT  16
310  X=R+6 : Y=18 : I=0
320  I=Q-I
330  COLOR=I
340  GOSUB  1028 : Z=0
350  Z=Z+1
360  IF  PEEK (-16384)>127 THEN  400
370  Z=Z+1
380  IF  Z<100  THEN  350
```

```
390  GOTO 320
400  IF PEEK (-16384)=211 THEN 5000
410  O=(PEEK (-16384)<>144)
420  COLOR=0 : GOSUB 1028
430  IF O THEN 470
440  GOSUB 3000 : V=RE
450  FOR Z=1 TO 300 : NEXT Z
460  GOTO 550
470  INPUT V
480  VTAB 23 : PRINT "  " : VTAB 23
490  IF V>=0 THEN 530
500  PRINT "NO MINUS SIGNS";
510  PRINT "... TRY AGAIN" : GOSUB 2000
520  GOTO 310
530  IF V<100 THEN 550
540  PRINT "TOO MUCH"; : GOTO 510
550  N=INT (V/10) : COLOR=Q : X=R : Y=18
560  IF NOT N THEN 580
570  GOSUB 4000
580  N= V-INT (V/10)) * 10
590  X=R+6 : GOSUB 4000
600  GOSUB 3000
610  IF V<>RE THEN 800
620  IF NOT O THEN FOR U=1 TO 1500 : NEXT
     U; GOTO 660
630  COLOR=0 : GOSUB 1045
640  GOSUB 1043
660  VTAB 21
670  FOR Z=1 TO 4
680  FOR LC=1 TO 13
```

```
690  PRINT "    ";
700  NEXT LC
710  PRINT
720  NEXT Z
730  VTAB 23 : LC=1
740  NEXT R
750  GOTO 60
800  X1=X : Y1=Y
810  GOSUB 1042
820  Y=Y1 : X=X1
830  IF LC>37 THEN 890
840  VTAB 22 : HTAB LC : VTAB 21
850  PRINT V
860  LC=LC+2
870  IF V>9 THEN LC=LC+1
880  VTAB 23
890  COLOR=0
900  FOR M=0 TO 10
910  VLIN Y, Y+7 AT X-6+M
920  NEXT M
930  COLOR=0
940  GOTO 310
1000 HLIN X+1, X+3 AT Y+3 : RETURN
1002 VLIN Y, Y+6 AT X+4 : RETURN
1004 VLIN Y+1, Y+2AT X : PLOT X+4,
     Y+4 : RETURN
1006 VLIN Y+2, Y+4 AT X : PLOTX+1, Y+4 :
     RETURN
1008 VLIN Y, Y+3 AT X : HLIN X+1, X+3
     AT Y+3 : VLIN Y, Y+6 AT X+4 : RETURN
```

```
1010 VLIN Y+1, Y+2 AT X+4 : PLOT X, Y+4 :
     RETURN
1012 VLIN Y, Y+2 AT X+4 : RETURN
1014 HLIN X+1, X+4 AT Y : VLIN Y+1, Y+6
     AT X+4 : RETURN
1016 VLIN Y, Y+6 AT X : HLIN X+1, X+3
     ATY : HLIN X+1, X+3 AT Y+3 : HLIN X+1,
     X+3 AT Y+6 : VLIN Y, Y+6 AT X+4 :
     RETURN
1018 VLIN Y+4, Y+7 AT X : RETURN
1020 HLIN X+1, X+3 AT Y+3 : VLIN Y+1,
     Y+5 AT X+2 : RETORN
1022 HLIN X+1, X+3 AT Y+3 : RETURN
1024 FOR Z=0 TO 4 : PLOT X+Z, Y+1+Z :
     PLOT X+4-Z, Y+1+Z : NEXT Z
1025 RETURN
1026 HLIN X, X+4 AT Y+3 : PLOT X+2,
     Y+1 : PLOT X+2, Y+5 : RETURN
1028 IF NOT O THEN RETURN
1029 VLIN Y, Y+1 AT X : HLIN X+1, X+4
     AT Y : VLIN Y+1, Y+3 AT X+4 : HLIN
     X+2, X+4 AT Y+3 : PLOT X+2, Y+4 :
     PLOT X+2, Y+6 : RETURN
1042 A$ = "4202611820335037330 : 404564503673610742790
     7268429420420": GOTO 1044
1043 A$ = "4202611820335037330 : 401531561591502642682
     61 27727268429"
1044 COLOR = INT (RND (1) * 15) + 1
1045 X=R : Y=27
1048 FOR Z=1 TO 52 STEP 3
```

```
1049  Z1=ASC (MID$ (A$, Z, 1))-48+X:HLIN
      Z1, Z1+ASC (MID$ (A$, Z+1, 1))-48  AT
      ASC (MID$ (A$, Z+2, 1))-48  +Y
1050  NEXT  Z
1060  RETURN
2000  FOR  Z=1  TO  50
2010  M=PEEK (-16336)-PEEK (-16336)+PEEK
      (-16336)
2020  NEXT  Z
2030  RETURN
3000  IF  S=0  THEN  RE=A+B:RETURN
3010  IF  S=1  THEN  RE=A-B:RETURN
3015  IF  S=2  THEN  RE=INT (A*B+.5):RETURN
3020  RE=INT (A/B+*5):RETURN
4000  IF  N=1  OR  N=4  OR  N=7  OR  N=8  THEN
      4030
4010  GOSUB  1016
4020  COLOR=0
4030  ON  N+1  GOSUB  1000, 1002, 1004  1006,
      1008, 1010, 1012, 1014, 1016, 1018
4040  COLOR=Q
4050  RETURN
5000  Z=PEEK (-16368)
5010  TEXT
5020  HOME
5030  END
```

这个程序的粗框图如图9-2所示。读者可对照阅读。

程序中80到140行给出题目的初始数据、运算符号和显示用的颜色号。S的值从0到3分别代表+、−、×、÷。颜色号不使用0号和10号，且保证和上次用的颜色不一样。

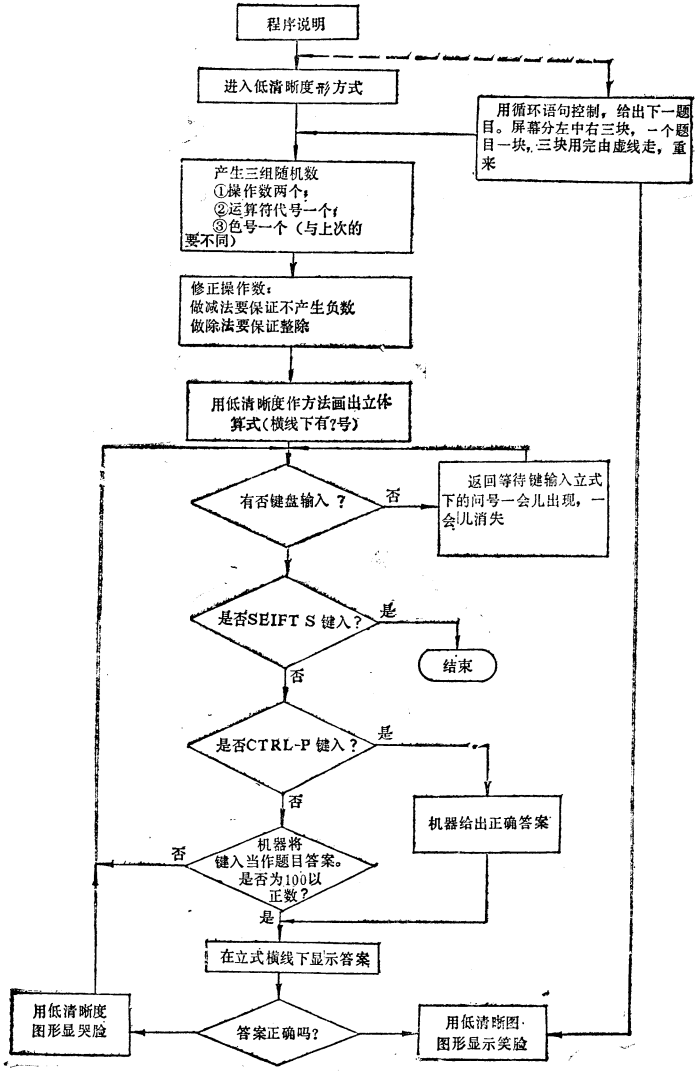

程序说明
进入低清晰度形方式
用循环语句控制，给出下一题目。屏幕分左中右三块，一个题目一块，三块用完由虚线走，重来
产生三组随机数
①操作数两个；
②运算符代号一个；
③色号一个（与上次的要不同）
修正操作数：
做减法要保证不产生负数
做除法要保证整除
用低清晰度作方法画出立体算式（横线下有?号）
有否键盘输入？
否
返回等待键输入立式下的问号一会儿出现，一会儿消失
是否SEIFT S 键入？
是
结束
否
是否CTRL-P 键入？
是
否
机器给出正确答案
否
机器将键入当作题目答案。是否为100以正数？
是
在立式横线下显示答案
用低清晰度图形显哭脸
答案正确吗？
用低清晰图·图形显示笑脸

图9—2

150到160行保证被减数大于减数。

170到220行保证被除数能被整数整除，并且当被除数有十位数字时把它靠左显示。注意200行的表达式：当有十位数字时，表达式的值为Q，颜色用Q号；没有十位数字时，用0号颜色。子程序4000是显示数字的。

230到250行控制显示第一个数的个位数。

260到270行控制显示第二个数的个位数。

280到290行控制显示运算符号。子程序1020，1022，1024，1026分别是+、−、×、÷的绘制语句组。

300行控制画一横线。

310行到340行控制画问号。子程序1028是画问号的语句组。

350行至470行接收键盘输入。这里用变量Z计算等待反复次数。反复100次没有输入时，涂掉问号；再反复100次，若还无输入时，再出现问号，重复下去，使问号呈闪烁状态，直到有键盘输入。320行的I值用来控制是显示问号，还是涂掉问号，它总是从0变到Q，再从Q变到0。360行专门用来检查有无键盘输入。400行控制输入SEIFT-S时的结束处理（5000行以后为程序结束处理）。410行专用于检查是否敲入CTRL-P。当输入CTRL-P时，变量O为假(0)，否则O为真（1)。

440行控制当O为假时，计算题目的正确答案，赋给V。子程序3000专用于求当前题目的结果，存于变量RE中

490到540行检查输入的答案是否为小于100的正数，如果不是，重新输入。

550到590行，显示答案。

610行查答案对不对。答案正确时涂去哭脸、显示笑脸(机器给答案时，不显脸谱)；回答不正确时由810行显示哭脸。显示笑脸的画图语句在1043行的子程序中，显示哭脸的画图语句在1042行的子程序中。两个子程序首行号中变量A$的数据，分别是它

们图形数据，执行画图动作的是1048到1050行的程序段。

660到730行和830到880行，由LC控制打印位置的打印语句，可以在屏幕底部的文本行上显示用户的输入。当前输入在22行上显示，然后将每次输入在21行上记录下来。

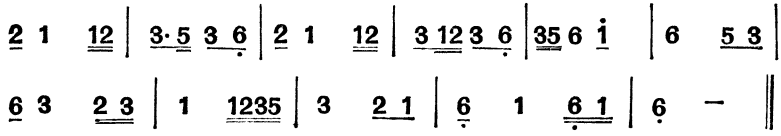
890到920行用于涂去错误答案。

940行控制返回去重做。

子程序2000行，用于遇到回答数据小于0或大于99时的警告，会发出声音来的。

习　题

(一)写一个唱歌程序，奏出下面的曲子：



(二)读懂下面的程序，说明这个程序的作用是什么，并指出这个程序还缺什么内容？

```
10   REM SOUND GENERATOR PROGRAM
20   REM ARRAY REMEBERS ENTERED TONES
30   DIM A%(500, 2)
40   REM CLEAR DISPLAY
50    TEXT : HOME
60   INPUT "(E) NTER, (L)ISTEN, (P)RINT?"; A$
70   IF A$ = "L" THEN 1000
80   IF A$ = "P" THEN 1200
90   IF A$<> "E" THEN 50
100  REM ENTER EACH TONE
110  PRINT
120  I = 0
```

```
130  M=I
140  GOSUB 3000
150  REM END OF TONE ENTRY?
160  IF F=0 AND D=0 THEN I=I-1:GOTO 210
170  REM NO-REMEMBER TONE
180  A%(I, 1)=F:A%(I, 2)=D
190  I=I+1
200  GOTO 130
210  REM CHANGE ANY ITEMS HERE
220  PRINT "WHICH NOTE TO CHANGE (0-"; I;
     ")";
230  INPUT E
240  IF E=0 THEN 50
250  IF E<1 OR E>I THEN 220
260  M=E:GOSUB 3000
270  A%(E, 1)=F:A%(E, 2)=D
280  GOTO 220
1000 REM LISTEN TO THE NOTES SO FAR
1010 FOR K=0 TO I
1020 F=A%(K, 1):D=A%(K, 2)
1030 GOSUB 3200
1040 NEXT K
1050 GOTO 50
1200 REM PRINT OUT THE NOTES
1210 PRINT "NOTE#", "FREQ", "DURATION"
1220 FOR K=0 TO I
1230 PRINT K, A%(K, 1), A%(K, 2)
1240 NEXT K
1250 PRINT
1260 GOTO 50
```

```
3000 PRINT "TONE"; M
3010 INPUT "ENTER FREQUNCY, DURATION"; F, D
3020 IF F=0 AND D=0 THEN RETURN
3030 IF (F<0 OR F>255) OR (D<1 OR D>255)
     THEN 3010
3040 GOSUB 3200
3050 RETURN
3200 REM SPEAKER DRIVER
3210 POKE 768, F
3220 POKE 769, D
3230 CALL 770
3240 RETURN
```

(三)请认真读一下游戏程序一节所列的那个手枪对射程序例，并画出它的框图来。

(四)自行设计一个BASIC游戏程序，并上机调试。

第十章　磁盘操作系统命令与文件

在前面一些章节中，我们曾提到过带名字的SAVE、LOAD、BSAVE、BLOAD及CATALOG命令。这些命令必须在磁盘操作系统(DOS——Disk Operating System 的简称)支持下才能使用。操作系统有它自己的一套命令和使用方法。BASIC解释程序是解释不了这些命令的，它只能转交DOS处理。因此当没有DOS时，这些命令被BASIC认为是错误的。

目前大部分苹果BASIC用DOS3.3来支持它。DOS3.3存放在一张系统磁盘上。开机时只要把这张盘插入1号磁盘机内，便会自动把它装入内存中。

假定你的磁盘控制器插在主机的6号槽口上，PR #6命令也可以启动磁盘机，把DOS3.3装入内存。

把DOS3.3装入内存时，磁盘上的第一个BASIC程序（通常用HELLO这个名字）也被调入内存并投入运行。

本章将介绍DOS3.3的命令与使用方法，特别是要介绍一下顺序文件和随机文件。

§1. 软　磁　盘

苹果机目前使用较普遍的是$5\frac{1}{4}$时的单面单密度软磁盘。它是一个表面带磁的软的圆盘，封装在一个方形塑料保护袋里。袋上开了一个槽形窗口，磁盘放入磁盘机后，磁盘机的读写磁头就落在这个窗口内。随着软件对磁盘机内小型电机转动的控制，磁头

也就在这个窗口内前后移动，并把转动着的磁盘上的有关信息读入内存，或将内存中有关信息写入磁盘。塑料袋的边角上还开了一个小方口，它表示这张磁盘上既能写入信息，也能读出信息；如果把这个小方口堵上（或干脆不开缺口），这张磁盘只能往内存中读信息，而内存中的信息写不到磁盘上。这叫写保护。

磁盘从外向内分为35道（即35个同心圆，编号从0开始），每个道上又分作16个区（又称扇区，好象把磁盘分为16把扇子。扇区也从0区开始编号）。每个区可存放256个字节的信息。

由此算来，磁盘总容量应是$35\times16\times256=143360$，通称143K字节。但由于DOS占用了三道，还有一道专用于记载这块磁盘本身使用情况的（如已有那些文件，各有多长，占用了哪些道的那些区，磁盘上何处还没有信息等），所以真正留给用户的只有31道，总容量为：$31\times16\times256=126976$，通常称127K字节。

软磁盘对环境要求不算太高，但要防止掉进灰尘，也要防止用坚硬的东西划它，不能折叠，也不要靠近强磁场。

§2. DOS命令简介

DOS命令可分五大类：内务命令、访问命令、机器语言命令、顺序文本文件命令和随机文本文件命令。

这一节只介绍前三类命令。后两类将专节介绍。

一、内务命令

内务命令的一般格式是：

命令名 f〔，V_v〕〔，S_s〕〔，D_d〕

格式中的大写字母照样写出。小写字母代表一个参数，要给出具体数据，在立即执行的DOS命令中，不能用变量名给出。可选项的顺序可以改变。

下面分别说明各项的意义。

f：代表文件的名字。文件名可由1到30个字符组成，第一个字符必须是字母。原则上除了逗号“，”、“CTRL-M”、“RETURN”等以外，键盘上的字符均可使用。但控制字符被用于文件名中时，将不被显示。这一点常常被用作文件的保密。在立即执行方式下，*f*若以字符串变量名的形式给出，它将不理会该变量的具体值（即使已被赋值），而以给出的变量名本身为文件名。

V项：用以给出磁盘的编号。小写*v*代表一个磁盘号码数字，其范围为1到254。如果给出0值，或不选用此项，那么DOS会使用磁盘上的编号。一张新磁盘在首次使用时不给出编号或用0号，DOS会自动以*v*=254处理。相当于使用了V254。

S项：表示磁盘控制器插在主机内的槽口号。苹果机共有8个槽口，编号从0到7。由于0号槽口被定义为显示终端和键盘，所以可选用的槽口号只能是1到7。通常把磁盘控制器插在6号槽口上。

只有一个槽口上插有磁盘控制器时，这一项可不选用。机器会自动查找到的。

D项：代表磁盘机的编号。一般一个磁盘控制器可控制两个磁盘机，于是把它们编为1号磁盘机和2号磁盘机。因此，*d*的值只能为1或2。当某号磁盘机被启动后，它就取得了控制权（称为当前盘机），只要不通过相应命令中D选项来改动，内存信息只能和这个盘机打交道。首次和磁盘交换信息而又未指定盘机号时，DOS总是把1号盘机当作当前盘机来处理。

如果只带一个磁盘机时，这项可不选用。

以下分别说明DOS的内务命令：

1. INIT命令

INIT命令格式是：

INIT *f* [, V*v*][, S*s*][, D*d*]

这是磁盘初始化命令（初始化也称格式化）。

从工厂或商店买来的空白磁盘，是不能直接使用的，必须将它初始化。初始化的任务是把磁盘划分为35道，每道分成十六个区，同时把当前正在内存的DOS写入磁盘中，并把当前在内存的BASIC程序作为这张磁盘上的第一个程序存入磁盘，还要在磁盘上登记这块盘的使用情况。

一个磁盘被初始化之后，就可以用来和内存交换信息了。

初始化的方法很简单，首先把DOS引导出来装入内存（不做到这一点，根本无法使用磁盘机）。然后写一段 BASIC 程序。这段程序的内容随你的意愿，主要用于刚开机时对用户的问候或简要提示，因此不要太长，运行时间要短。然后给这个程序起个名字（通常用HELLO）。这时就可以使用INIT命令了。

比如文件名（在此即程序名）叫 HELLO，磁盘编号用35，槽口号取6，盘机号取2。这时要把准备初始化的那块空白盘，插入指定的 2 号盘机中，然后敲入下列命令（别忘了敲回车键）：

```
INIT  HELLO, V35, D2, S6
```

2 号磁盘机被启动，初始化就开始了，等到磁盘机的小红灯熄灭后， BASIC 提示符又显示出来，并以闪烁光标等待你的输入。这张新盘的初始化就完成了。

如果你不取那些可选项，直接打入命令：

```
INIT  HELLO
```

也可以。但必须把空盘插入当前盘机中。这样初始化的磁盘编号为254。

磁盘初始化，也叫格式化。这个命令一定要慎用。因为对一张有用的磁盘初始化后，原来的信息也就全部找不到了。

2.CATALOG命令

CATALOG命令的格式是：

CATALOG[, S_s][, D_d]

这个命令不带文件名，也不带V 可选项。如果给出文件名，会

出错；如果带V选项，将被忽略去。

这个命令会把指定盘上的磁盘号和文件目录列示出来。

磁盘号被列在第一行，接下去是各文件目录。每个文件分成左中右三栏：文件类型、占用扇区数和文件名。前两栏连接在一起。

文件类型用一个大写英文字母表示，文件类型共四种：

A　　表示浮点BASIC程序

I　　表示整数BASIC 程序

B　　表示二进制文件（含机器语言程序）

T　　表示文本文件

我们前面几章的程序，都是浮点BASIC程序，存入磁盘后列示出来都以A作为文件类型。

高清晰度图形、图形表和唱歌用的发音子程序，都是二进制文件，存入磁盘后列示出来，前面是B字。

整数BASIC程序和文本文件，以后会讲到。

有些文件在其类型标志的前面还有一个“*”号，表明那个文件用“锁”保护起来了。这种文件只能提供使用，不能予以修改（除非你采取适当措施）。也就是说，它只能读，不能写。这也是一种写保护。它是由软件来实现的。

CATALOG列出的每个文件的扇区数，紧跟在类型标志之后，它表明该文件在磁盘上占用多少扇区（一个扇区256个字节），所以它也可以认为是文件的长度。它只用三位十进制数表示。由于一个文件存入磁盘时要登录一些索引项，因此一个文件再小也要占用两个扇区。

一个文件用同一名字多次存入时，取其最长占用扇区数。即使最后一次存入时内容可能很少，但仍要占用原来最多占用的扇区，除非先把原来存入的文件删除掉。

3. SAVE命令

SAVE命令的格式是：

SAVE f [, V_v][, S_s][, D_d]

这个命令可以将当前内存中的BASIC程序以给定的文件名存入指定的磁盘中。

当磁盘上已有同一类型的同名文件时，将以新文件取代旧文件。占用扇区数以其中最大者为准。

如果这个同名文件已加锁保护，再SAVE 它无法实现，只能给出出错提示。

如果磁盘上已有一个同名文件但不属于同一文件类型,SAVE它也会给出出错信息。

所以使用这个命令时，一般先用CATALOG 命令查看一下磁盘上已有的文件名，以免破坏掉有用的文件或给出不适当的文件名。

一个程序存入磁盘后,可用 CATALOG 看到它的名字已被登录。

SAVE命令只能将内存中当前的BASIC 程序存入磁盘。而不能存入别的类型的文件。

4． LOAD命令

LOAD命令的格式是：

LOAD f [, V_v] [, S_s] [, D_d]

这个命令的功能是把指定磁盘上指定的BASIC程序调入内存中来。

用LOAD将磁盘上的程序调入内存后，内存中原来的 BASIC 程序、变量表及数组表都将被冲掉。

如果磁盘上没有指定的程序，将会告诉你该文件没有。因此，一般用LOAD命令前，最好先用 CATALOG 命令，查看一下当前盘上有没有你要用的程序。

LOAD命令只能调出BASIC程序。别的类型的文件调不出来。

5. RUN命令

RUN命令的格式是：

RUN f 〔, V_v〕 〔, S_s〕〔, D_d〕

这个命令和BASIC中的RUN命令有区别，它允许带文件名和其它可选项。

这个命令的功能相当于先执行一次LOAD命令，再执行BASIC中的RUN命令。换言之，它是把指定磁盘上指定的BASIC程序调入内存，并接着运行它。

这个命令同LOAD命令一样，会把内存中原来的程序、变量表和数组表清除掉。

这个命令也只对BASIC程序有效。别的类型的文件不能使用它。

6. RENAME

RENAME命令的格式是：

RENAME f, g 〔, V_v〕 〔, S_s〕 〔, D_d〕

这是一个换名命令。它可以把指定磁盘上的文件f，换名为文件g。在执行这个命令时并不检查磁盘上是否已用过文件名g，因此它可能在磁盘上留下两个同名文件，以致造成混乱。所以在使用这个命令时，要先用CATALOG命令查看一遍文件目录。

7. DELETE命令

DELETE命令的格式是：

DELETE f 〔, V_v〕 〔, S_s〕 〔, D_d〕

这是一个删除命令。它可以把指定磁盘上的指定文件删除掉。当然这必须在没有写保护时才能做得到。

如果一个程序经过多次修改，而且每次都以同名文件存入了磁盘，最后的程序可能是精练的，最好先用这个命令删除掉旧的文件后，再存入新的文件，这样可以节省占用的磁盘空间。

已经没有用处的或无保留价值的程序或别的文件，也应当用

这个命令把它们删除掉。

8. LOCK命令

LOCK 命令的格式是：

LOCK f 〔, V_v〕 〔, S_s〕 〔, D_d〕

这是一个加锁命令。它可以给指定磁盘上的指定文件加锁保护。被锁保护的文件，用CATALOG 命令列示出来时，前面带有一个星号“*”。

这个命令可以防止无意的改写和删除。加锁保护后的文件只能使用。一些珍贵的文件存入磁盘后，最好加锁保护起来。

这种软的写保护比在磁盘袋上封死那个小方口的硬的写保护有一定的方便之处。软的写保护只对加锁的文件起作用，对未加锁的文件照样可以写入或删除；而硬的写保护对整个盘片都起作用，你想往其上存入任何一点新内容都办不到。

9. UNLOCK命令

UNLOCK命令格式是：

UNLOCK f 〔, V_v〕 〔, S_s〕 〔, D_d〕

这是一个开锁命令。它可以把加锁保护的文件恢复成未加锁保护状态。使用这一命令后，再用CATALOG 命令查看，目录前面的“*”不见了。

任何一个已加锁保护的文件，只有用开锁命令将其解脱出来，才能被改写或删除。

10. VERIFY命令

VERIFY命令的格式是：

VERIFY f 〔, V_v〕 〔, S_s〕 〔, D_d〕

这个命令用来检查指定磁盘上的指定文件，信息是否正确。检查结果无误时，不给出任何信号。若检查有错，给出信息

I/O ERROR

一个文件建立时，是经过一个缓冲区将数据一批一批地送往

磁盘的。DOS会计算缓冲区内的代码和，并将这个代码和也存入磁盘中。这个命令则是将磁盘上那个文件的信息逐段求代码和，看与原先存入的代码和是否一致。如果一致，表明存入过程中没有发生错误。

11．MON命令

MON命令的格式是：

MON 〔C〕 〔, I〕 〔, O〕

通常，所有DOS命令、所有计算机内存与磁盘之间交换的信息，在终端显示屏上是不显示的。这有时使用户感到不方便。这个命令可以改变这种状况。

这个命令不带文件名，而可以带三个规定的大写字母，它们的意义是：

C：代表磁盘命令（即DOS命令）；

I：代表由磁盘向计算机输入信息；

O：代表由计算机向磁盘输出信息。

因此，使用命令：

MON O

之后，由计算机向磁盘输出的所有信息将依次显示在屏幕上。

使用命令

MON C, I, O

之后，DOS命令及计算机与磁盘之间交换的全部信息都被显示出来。

这个命令的三个选项次序可以任便，选用几个都行，但至少得选一个。多于一个时，两两之间用逗号分隔开。

12．NOMON 命令

NOMON命令格式是：

NOMON 〔C〕 〔, I〕 〔, O〕

这个命令专用于终止 MON 命令的。可选项的意义同上，选

用几个都可以，每两两之间以逗号隔开，而且至少要选用一个。没有使用过MON命令而用NOMON命令毫无意义。

13. MAXFILES

MAXFILES命令格式是：

MAXFILES *n*

计算机和磁盘交换信息时，要经过在内存中专门为之开辟的缓冲区（也称通道）。一个缓冲为595个字节的空间。建立DOS时，自动建立了这样三个缓冲区。有时三个缓冲区不够用，有时又嫌它占用的地方太多。MAXFILES命令可以改变缓冲区的个数。命令中的*n*就是用来设定缓冲存区个数的。*n*的值允许为1到16的整数。换言之，这种缓冲至少得一个，最多为16个。

以上十三条内务命令（也有称为管家命令的），其中除了LOAD和RUN命令外，其它命令都不会破坏当前内存中的BASIC程序。即使你的命令是针对别的文件（包括BASIC程序）使用的。

二、访问命令

这组命令共有六个。

1. FP命令

FP命令的格式是：

FP 〔, V_v〕 〔, S_s〕 〔, D_d〕

这个命令是将机器设置成浮点BASIC系统。一般是在已将系统建立为整数BASIC时，才使用此命令。

如果你的苹果机装有固化的APPLE SOFT BASIC，使用这个命令后，DOS将它用作语言系统，不管板上的开关设置在什么位置上。如果你使用的机器上没有固化的APPLE SOFT BASIC，DOS会把指定磁盘内的、名为APPLE SOFT的程序装入内存供你使用。

磁盘上的APPLE SOFT不能用LOAD或RUN命令使用。这

一点要注意。

可见，命令格式中的选项是针对没有固化APPLE SOFT BASIC的情况使用的。目前绝大部分机器都有这块固化的浮点BASIC，所以不要使用可选项。

2. INT 命令

INT命令的格式是：

INT

这个命令将使你的机器变成整数BASIC系统。

整数BASIC的提示符是：>

注意，不管使用FP命令或是INT命令，命令一经使用，内存中原来的程序将被清除。

3. CTRL-D命令

CTRL-D是一个不可显示字符，在操作系统中它又是一个命令。

BASIC解释程序是不认识操作系统命令的。用户每次从键盘上输入信息时，只要首字符（除空格外的有效字符）不是数字，BASIC解释程序先按立即执行命令处理。如果头几个字符既不是BASIC保留字也不是赋值语句，那么它不认为是BASIC语句或命令，就把处理权转交DOS。而对于你输入的第一个有效字符是数字的情况，BASIC认定是延迟执行方式下的程序，这样它就把解释大权包揽下来了。因此，DOS命令用于带行号的程序中，将得不到DOS的控制。这为用户带来的好处是DOS命令可以用作变量名（因为它们不是BASIC保留字）。但也带来了不便：DOS命令无法用于延迟执行方式。

CTRL-D命令，就是为此目的设置的。

BASIC解释任何一个PRINT语句时，对输出的每一个字符都要经过DOS的检查。利用这一特性，让PRINT语句首先输出一个CTRL-D字符作为标志，紧接在后面的输出字符（直至遇到RE

TURN) 都按DOS命令处理。这样DOS 命令就可以出现在程序之中。

CTRL-D 是一个不可显示字符，写在程序中从屏幕上只可看到一个引号对。所以通常用一个字符串先取得它，再加一个注解予以说明。如：

```
10  D$ = " " : REM  CTRL-D
```

此外还常用CHR $(4) 来代替它。因为 CHR $ (4) 就是字符CTRL-D。

现在我们举几个例子说明 CTRL-D的使用方法。

譬如我们介绍过的唱歌子程序，因为要经常使用，那就应存到磁盘上（假定用 MUSIC 名字)。过 去 我们总是先运行下列程序：

```
10   FOR  I=770  TO  790
20   READ  A
30   POKE  I, A
40   NEXT  I
50   END
```

然后再执行下列命令：

```
BSAVE  MUSIC, A  770, L  21
```

需要两道手续。现在，使用 CTRL-D命令，只需在上列程序中增加如下一个程序行：

```
45   PRINT  CHR$(4);  "BSAVE  MUSIC, A770, L21"
```

只要给出RUN命令，运行结束后，唱歌子程序也就以MUSIC为名存到了磁盘上。

类似上述方法，以后编写唱歌程序或需要这个子程序发音的任何程序时，在需要发音之前(或在程序的开始)，加 入 下面一个程序行：

```
行号   PRINT  CHR$ (4);  "BLOAD  MUSIC"
```

就可以了，那个子程序就装入了内存。

再例如，若有两个或更多个程序已经存入磁盘。希望运行完第一个程序之后接着运行第二个，然后再运行第三个，…。假定这些程序的名字依次叫A1，A2，…。不使用CTRL-D命令时，只能一次地敲入LOADA1，RUN；LOAD A2，RUN，…等命令，再简单点，也得一次次敲入RUN A1，RUN A2，…。有了 CTRL-D命令，你可以在第一个程序的尾部加入一个程序行：

```
行号 PRINT " "；"RUN A2"：REM CTRL-D
```

在第二个程序的结束处加入一个程序行：

```
行号 PRINT CHR$ (4)；"RUN A3"
…
```

这样，运行完第一个程序之后，会自动把第二个程序调入内存并运行它，第二个程序运行结束，又自动调入第三个程序运行，…，这样就实现了程序的自动链接。好象是一个程序分段存在磁盘上，运行时可以自动链接起来。

这种链接方法，对于一些表演性程序很有用。它不需要对每个程序都给出运行命令。然而这种链接的最大缺点是数据不能从一个程序传入另一个程序。因为这种链接是经 由 RUN 命令实现的，而RUN命令都有清除原来程序和变量表、数组表的功能。

CTRL-D命令和DOS 的其它命令不同。立即执行的 CTRL-D只能当作一个字符。它必须用在保留字PRINT之后，且在其后直至敲入RETURN之前的字符串必须是DOS命令。

CTRL-D命令的使用，将使DOS 命令中的参数，能够用变量的形式给出。

比如想运行一个名为AA的 BASIC 程序，这个程序确实在当前盘上。假定你用下列两个命令：

```
A$ = "AA"
RUN A$
```

第一个命令使A$取得AA两个字符的值，这一点确定无疑。你可用下列命令予以检查：

```
PRINT  A$
```

它输出确实是AA。但第二个命令（RUN A$）执行的结果将可能告诉你没有这个文件。这是因为DOS把A$两个字符当作了文件名。由此可见DOS不懂得变量名是什么意思，它根本不去查找变量表。对数型变量也一样不认识。这使DOS命令的使用受到了很大限制。

CTRL-D命令的使用，这类问题得到了满意的解决。因为PRINT CTRL-D只是把引号括起来的部分当作DOS命令，而引号对外面的内容仍由BASIC解释程序处理的。这样把变量名放在这种方式下的引号对的外面就可以了。

比如：

```
10  D$=CHR$ (4) : A$= "AA"
20  PRINT  D$; "RUN"; A$
```

运行时，会把AA程序调入内存并执行它。

再如：

```
44  D$=CHR$ (4) : P=770 : Q=21
45  PRINT  D$; "BSAVE  MUSIC, A"; P; ", L"; Q
```

与前面的45行程序等效。

4. CHAIN 命令

CHAIN命令的格式是：

CHAIN *f* 〔, V_v〕〔, S_s〕〔, D_d〕

这个命令是专为整数BASIC设计的。它的功能是在不破坏当前内存中的变量表和数组表的前提下，把指定磁盘上的指定文件调入内存并投入运行，而且原来的变量表和数组表能继续使用。这样就实现了两个程序之间的数据传递。

假定磁盘上有一个程序，名为PRINT，它只有三个程序行：

```
10  PRINT  A,
20  PRINT  A * A
30  END
```

那么，可先执行另外一个程序（或用立即执行命令），使变量A得到一个数，比如为100。这时用下列命令（实际上可视为第一个程序）：

```
CHAIN  PRINT
```

结果会在屏幕上输出：

```
100                          10000
```

第一个程序运行结束后，一方面把指定的第二个程序（名为PRINT）装入内存并使之运行，同时第一个程序中有关变量的取值也传入第二个程序的同名变量。所以第二个程序输出了那样的结果。

再如，有两个整数BASIC程序。第一个是：

```
10  A=15 * 8
20  B=7+3
30  C=A-B
40  END
```

第二个整数BASIC程序是：

```
10  D=A+B
20  PRINT  A, B, C, D
30  END
```

我们把第二个程序起个名字（假定为BB）存入磁盘，并把第一程序的第40行改为(引号中敲CTRL-D)：

```
40  PRINT  "          ";"CHAIN  BB"
```

这样，运行第一个程序时，除了执行那些赋值语句外，还会把第二个程序BB调出来投入运行，而且第一个程序中的变量表为第二程序继续使用（也就是可以把第一个程序中的数据传递给第二个程序）。于是运行的结果将输出：

120　　10　　110

130

由于CHAIN命令是专为整数 BASIC程序设计的，所以在浮点BASIC中不能使用。

但经过特殊处理后，CHAIN 也可以在浮点BASIC中使用。这些处理是：

(1) 系统主磁盘上有一个名为 CHAIN 的二进制文件，要把它拷贝到你的当前盘上。

这需先将系统主磁盘插入盘机内，先把它调入内存。用命令

```
BLOAD  CHAIN, A24576
```

然后将系统主盘取出，而把当前盘插入盘机，使用命令

```
BSAVE  CHAIN, A24576, L430
```

等磁盘机上小红灯熄灭后，当前盘上就有这个 CHAIN 文件了。

当然，你的当前盘上已有这个文件时，这一步就不需要了。

(2) 现在可以做浮点 BASIC 的 链 接了。假定被链接的程序（第二个程序）名为 A2，它必须存于当前盘上。这时你可以在第一个程序的结束处加入如下两个程序行（行号是假设的）：

```
30000  PRINT  CHR$ (4);   "BLOAD  CHAIN, A520"
30010  CALL  520"A2"
```

这样，运行第一个程序后会把其变量表、数组表全留着，并接着运行第二个（在此名 为 A2）程序。第一个程序的数据可以为第二个程序继续使用。

这里要注意，CALL 后边的数字必须 是 520，而且它和后边的引号之间不得有任何字符（包括空格符）。

由此可见，CHAIN虽然可以为浮点BASIC使用，但是它不是以命令形式出现的，而是要构造一个特殊的语句结构。

CHAIN 对 于 一些较长的程序很有用。因为一个长的程序有

时在内存中放不下。你可以把它分成几个程序，用 CHAIN 来链接它们，这样做，数据可以不断传递下去。

5. PR#s命令

此命令的格式就是：

PR#s

其中s指明一个设备的槽口号(0～7)。

命令的功能是接通s 号槽口上的输出设备。比如 1 号槽口上为打印机接口板，那么用命令

PR#1

可以接通打印机。

写在程序行中可用：

行号 PRINT CHR$(4);"PR#1"

但由于PR#也是BASIC 中的保留字，在程序行中直接用：

行号 PR#1

也是可以的。

6. IN#s命令。

此命令与PR#s命令类似，它是接通接在 s号槽口上的输入设备的。

同样，IN#s 也是BASIC 的命令，也可直接写入行号中。

PR#s和IN#s命令都要求 s 槽口上有相应的设备。否则系统会挂起来停止工作。这时可用RESET命令恢复系统。

三、机器语言命令

DOS的机器语言命令有三个。

1. BSAVE 命令

命令的格式是：

BSAVE f, A_a, L_l 〔, V_v〕 〔, S_r〕 〔, D_d〕

命令中A后边的a，表明一个地址号码，可以用十进制数写，也可以用十六进制数写。当用十六进数写时，数字前面加符号

“$”。L后边的$j$表明一个长度，同样可以用十进制数或十六进制数写出。

这一命令我们前面用到过。高清晰度图形、图形表和唱歌用的子程序都是用这个命令存入磁盘的。A项代表要存入磁盘的数据从内存的那个地址开始，L项表明要存入的字节数（长度）。这个命令的功能就是把从a开始的j个字节的内容，以给定的文件名存入指定的磁盘上。

从前面的例子中可以看出，BSAVE存入磁盘的可以是图素，也可以是机器语言程序，它们都是二进制文件。对于BASIC程序不能用此命令。

2. BLOAD命令

BLOAD命令的格式是：

BLOAD f〔, A_a〕〔, V_v〕〔, S_s〕〔, D_d〕

这个命是把指定磁盘上的指定文件（二进制文件）调入内存中。

当不给出A选项时，该文件会调至内存中存入磁盘以前所在的地址中。给出A选项后，将调入从a指定的地址开始的区域。

这个命令没有长度L选项。调出时的长度以存入时长度为准。这可以确保文件的完整性。

用BLOAD命令只能调出BSAVE进去的文件。任何BASIC程序都不能用它调出。

3. BRUN命令

BRUN命令的格式是：

BRUN f[, Aa][, Vv][, Ss][, Dd]

这个命令相当于先执行一次BLOAD命令，并从头开始当作机器语言程序运行它。

有些二进制文件并不是机器语言程序(比如图形、图形表)，是运行不起来的，不能使用此命令。

关于机器语言程序，下章将进一步介绍。

§3. 顺序文本文件

一、文件的概念

内存可以和磁盘交换信息，这些信息在磁盘上的集合，称为磁盘文件。

一个BASIC程序，在内存中实际上是一批信息的集合。通过SAVE 命令将其存入磁盘，变成在磁盘上的一组信息的集合。这就是BASIC文件。通过LOAD命令，这个BASIC 文件装入内存，就是BASIC程序。

一组二进制信息(图形、图形表、机器语言程序)在内存中也是信息的集合，通过BSAVE 命令使其变成磁盘上一组信息集合，这就是二进制文件。二进制文件可通过 BLOAD 命令装入内存。

一个程序在运行时，会输出一些数据。如果把这些数据输出到磁盘上，它们在磁盘上的集合也是文件，称为文本文件。文本文件中的信息一样可以读入程序中使用（由文本文件输入数据）。这种文本文件是不同于BASIC文件和二进制文件的一种新的文件类型。

这种新的文件类型，为我们提供了这样几种可能性：当一个程序的数据大到内存放不下的时候，可以把这些数据送到文本文件中去，需要使用时再相应地读入内存；一个程序运行中的某些中间结果也可以存到文本文件中去，以后另外的程序还可以把那些数据读入使用。

因此，文本文件的使用，不仅相当于扩大了内存的容量，使一些数据量太大的程序得以运行，而且提供了长久保存数据的手段。这就是文本文件的功能。

二、顺序文本文件的存放格式

文件中的数据，都是以每个字符的ASCII 码的形式存放的。

每一个字符的ASCII码占一个字节的位置。

假如往磁盘上存放如下信息：

7↙AT↙ONE↙BLOW，OP↙

其中↙代表RETURN字符。

它们在磁盘上的存放格式是：

字符	7	↙	A	T	↙	O	N	E	↙
ASCII码	55	13	65	84	13	79	78	69	13
字节号	0	1	2	3	4	5	6	7	8
字段号	0		1			2			
记录号	0		1			2			

字符	B	L	O	W	,	O	P	↙	空	空
ASCII码	66	76	79	87	44	79	80	13	00	00
字节号	9	10	11	12	13	14	15	16	17	18
字段号	3					4				
记录号	3									

这里字节号、字段号和记录号都从0开始编号。

由若干个字节组成一个字段，由若干个字段组成一个记录，由若干个记录组成一个文件。

字段的终结符为↙(RETURN)或逗号“，”。

记录的终结符为↙(RETURN)。

顺序文件中字符总是一个一个顺序排下去的。

三、顺序文本文件命令

把内存当前程序中的数据输出到磁盘文件上去，这叫建立一个文件。反过来，从文本文件中将数据读入内存中供程序使用，

这叫检索一个文件。建立或检索一个文本文件，必须使用相应的命令。

1．OPEN命令

OPEN命令的格式是：

OPEN f [，V_v] [，S_s] [，D_d]

OPEN 命令的作用是建立内存当前程序和指定磁盘上的指定文件之间的联系。也就是在内存中开辟一个由595个字节组成的缓冲区。内存中程序和磁盘上文本文件之间交换的所有信息，都经过这个缓冲区，所以又把它叫作通道。这一手续通常称为在通道上打开某个文件，或简单地称为打开文件。用OPEN 命令打开一个文件时，总是把一个内部读写指针指向文件的第0号字节，也就是指向文件的头，以表明向文件写入信息或从文件读出信息时的起始位置。

2．WRITE命令

WRITE命令的格式是：

WRITE f [，B_b]

这个命令的作用是使其后的PRINT输出项不是送往屏幕，而是经由缓冲区写到指定的文本文件中去。

其中的B 选项指明从该文件的第b个字节开始写起。因此b必须是一个非负整数。当不选此项时，从0号字节开始写起，相当于取B0。注意这个 b值应是该文件的一个绝对位置（从头开始的字节号）。

这个命令中的文件，必须是已经打开的，因此不能再有 V，S，D的可选项。

WRITE命令不能用于立即执行方式。这个命令的功能在遇到任何一个DOS命令或INPUT语句之后被取消。

3．READ命令

READ命令的格式是：

READ f [, B_b]

READ命令的功能是使其后的INPUT语句不是从键盘上读入数据，而是从指定的文本文件中读入数据。

其中B选项意义同上，指明从文件的第b个字节开始读起。

这个命令只能从已打开的文件上读取数据，所以不能再给出V、D、S等选项。

READ 命令不能用于立即执行方式。任何一个延迟执行的DOS命令（如PRINT CTRL-D）可以停止这个命令的作用。

4．CLOSE命令

CLOSE命令的格式是：

CLOSE [f]

这个命令的作用是关闭指定的文件，释放缓冲区。如不给出文件名时，关闭所有被打开的文件(除EXEC打开的以外。EXEC在下节介绍)。

所谓关闭，就是使内存中的程序和磁盘上的文件脱离联系，终止它们之间的信息交换。

在执行这个命令时，如果前面执行的是写(WRITE)命令，它会把缓冲区中的最后一批数据送入文件，然后再行关闭。

5．APPEND命令

APPEND命令的格式是：

APPEND f [, V_v] [, S_s] [, D_d]

这个命令也是用来打开指定的文件的，但是它把内部读写指针指向文件的尾部。因此跟在其后的WRITE命令，可以使PRINT的输出接在原文件之后。所以这个命令常用于在一个文件中增加新的数据。

为了实现这种功能，APPEND之后，不能再用OPEN命令。因为OPEN又使内部读写指针指向文件头了。

6．POSITION命令

POSITION命令的格式是：

POSITION f [, Rp]

这个命令中的R选项指明一个相对记录位置，p的值只能取非负整数。命令的功能是使指定文件的内部读写指针向后移动p个记录位置，也就是从当前位置开始向后扫描，跳过p个 RETURN。因此，当$p=0$时，其后的读和写从指针的当前位置开始；当$p=1$时，内部指针从当前位置开始向后扫过一个RETURN停下来，以后的读写将从那里开始。如此类推，但在扫描过程中若发现某一个字节中为全0时，说明那里没有信息，给出提示：

END OF DATA

这表明给出的p值太大了，已超过了文件的最后位置。

R选项不取时，相当于R0。

POSITION命令应放在OPEN命令之后，WRITE 或 READ命令之前。如果把它置于OPEN之前，OPEN命令会使指针指向文件头；把它置于WRITE或READ之后，它会取消WRITE 或 READ的功能。

三、建立一个顺序文件的例

假定某单位有1000人，想把每个人的姓名、性别、籍贯、基本工资等数据存入一个文本文件中，可以将这些人的各项数据写在DATA语句里，通过循环语句，用READ 将这些数据读入有关数组(当然这些数据也可以用INPUT语句从键盘上输入)。

下面的程序中，我们只在DATA语句中列出了五个人的情况作为示意：

```
10  DATA "MAO ZHEN-HUA", "NAN", "BEIJING",107.5
20  DATA "LI WEI-GUO", "NAN", "SHANGHAI",79
30  DATA "WANG LIANG", "NAN", "TIANJAN",69
40  DATA "LU HUA", "NU", "BEIJING", 74.5
50  DATA "WEN LING", "NU", "HEBEI TANGSHAN",
```

```
      56
:
1000  DIM A$(1000, 3), B(1000)
1010  FOR I=1 TO 1000
1020  FOR J=1 TO 3
1030  READ A$ (I, J)
1040  NEXT J
1050  READ B (I)
1060  NEXT I
2000  D$=CHR$ (4)
2010  PRINT D$; "OPEN ABC"
2020  PRINT D$; "WRITE ABC"
2030  FOR I=1 TO 1000
2040  FOR J=1 TO 3
2050  PRINT A$ (I, J)
2060  NEXT J
2070  PRINT B (I)
2080  NEXT I
2090  PRINT D$; "CLOSE ABC"
2100  END
```

显然，这是一个BASIC程序，如果给它起个名字(比如叫AAA)通过SAVE命令可以存入磁盘。用 CATALOG 命令列出时，最前面的类型标志是A字符。

我们关心的是这个程序运行时是什么样子，运行的结果又是什么。

对于2000行以前的程序段我们是很熟悉的。它会把列在DATA语句中的1000个人的情况读入A、B数组中。

2010行，将打开一个叫ABC的文本文件（如果磁盘上没有这个文件名，它会登记这个文件名，再打开它)。2020行指明后面的

PRINT输出要经缓冲区送入已打开的ABC文件中。真正往缓冲区送数据的是2050行和2070行。2030、2040、2060、2080只是用来控制传送数据的次数和顺序的。2090行则是关闭ABC文件。

传送数据的过程是这样的：依着PRINT输出的每个字符的顺序，按ASCII码在缓冲区中依次排放。当缓冲的595个字节位置存满后，把这595个字节的内容及其代码和（也称检查和）一并送入磁盘上ABC文件中（依指针指向的位置开始放，每放入一个字节，指针后退一个字节位置…）。然后缓冲区接收PRINT的后续输出。收满后再存盘。直至遇到CLOSE命令时，把缓冲区中最后一批数据及其代码和送入磁盘文件ABC中，然后关闭ABC文件。

这个程序运行结束后，磁盘上将出现一个以T为类型标志的ABC文件。这个文件的数据，就是DATA语句中那1000个人的数据。

这里需要进一步说明以下几个问题：

1．PRINT语句格式对文件格式的影响

（1）2050行和2070行的PRINT语句，都只输出一个数据（只跟一个表达式）。而且语句之尾部既不是逗号，也不是分号。实际上是一个回车符RETURN（因为我们结束该程序行输入时，正是敲了一个回车键）。这个回车符也被送入文本文件中，作为一个记录的终结符。换言之，上述程序运行后所建立的文本文件ABC中，将有4000个记录（在此也是4000个字段）。把它们用字符写出来将是如下的序列：

MAO ZHEN-HUA↙NAN↙BEIJING↙107.5↙LI WEI-GUO↙NAN↙SHANGHAI↙79↙WANG LIANG↙……

这种格式的缺点是：没有一个信息能标志出两个人情况的分界。这是由于一个记录就是一个字段，记录分得太小造成的。

（2）如果在2050行的PRINT语句尾部加上一个逗号（,）或分号（;），将会是什么样子呢？根据过去讲过的PRINT语句的规定：

分号将使下一个输出项紧接在这次的输出项之后，而逗号将使下一个输出项从下一个输出区输出。这就是说，逗号将使输出字符之间空下空隙。在屏幕上或打印纸上，这种空隙是有用的，而在文件中这将使数据间留下一个“空洞”，这对于磁盘顺序文件是不好的。它不但浪费了磁盘空间，也造成了处理上的复杂性，没有任何好处。因此系统规定：在往文件上输出数据的PRINT语句中，逗号视同于分号，所以在这种情况下，文件中相应的字符序列将成为：

MAO ZHEN-HUANANBEIJING107.5↙LI WEI-GUONANSHANGHAI79↙WANG LIANG……

这就把每个人的情况做成了一个记录。但这样一来，每个人的姓名、性别、籍贯和工资几个数据又不可区分地连在一起了。这当然也很不方便，这是把字段搞得过大造成的。

(3) 根据上面的情况，我们不难设想一种更为理想的文件结构。试看把2050行改为以下的格式：

```
2050  PRINT A$(I, J); ",";
```

它强行把逗号送入文件中某些数据之后，文件的相应字符序列便成为：

MAO ZHEN-HUA, NAN, BEIJING, 107.5↙LI WEI-GUO, NAN, SHANGHAI, 79↙WANG LIANG, ……

于是我们看到了，在文件中，每个人的一项情况(一个数据)被当作一个字段，而一个人的情况又被当作一个记录。显然这是一种较好的文件结构。

可见，输出到文件的PRINT语句的格式，一定要经过慎重考虑，切不可马虎从事。

如果把上面那个程序改成下面的样子：

```
10    DATA
⋮                                  } 同上
```

```
1000  D$ = CHR$ (4)
1010  A$ = "ABC"
1020  PRINT D$; "OPEN";A$
1030  PRINT D$; "WRITE";A$
1040  FOR I = 1 TO 1000
1050  READ B$, C$, E$, F
1060  PRINT B$;",";C$;",";E$;",";F
1070  NEXT I
1080  PRINT D$;  "CLOSE";A$
```

这个程序更短些。它运行的结果也是把每个人的每个数据当作一个字段，而把每个人的全部数据当作一个记录。

程序中的1050行的READ，是BASIC保留字，不是操作系统命令。

2．对使用INPUT语句的限制

(1) 前文已提及，对于331页程序例中的1030行和1050行的READ，改为INPUT是完全可以的。

(2) 而对刚刚写出的那个较短的程序例中，如果把1050行的READ换成INPUT，将是不行的。因为这个INPUT将会使1030行的WRITE命令失效。这一点千万要注意。

3．文件内部读写指针位置的确定

假定上面建立的ABC文件相应字符序列为：

MAO ZHEN-HUA, NAN, BEIJING,107.5↙LI WEI-GUO, NAN, SHANGHAI, 79↙WANG LIANG, ……

若是第二个人的工资增加9元(变成88元)，应如何来改写其中这一个字段呢？

下面几种方法都是可行的：

(1) 通过设定B_0选项

程序如下：

```
1000  D$ = CHR$ (4)
```

```
1010  PRINT D$;"OPEN ABC"
1020  PRINT D$; "WRITE ABC, B55"
1030  PRINT 79+9
1040  PRINT D$;"CLOSE ABC"
1050  END
```

这个程序中1020行 B项参数55，是从文件头开始一个个字节数出来的。原来79的7字，字节号为55。

(2) 使用POSITION命令

程序如下：

```
1000  D$=CHR$ (4)
1010  PRINT D$; "OPEN ABC"
1020  PRINT D$; "POSITION ABC, R1"
1030  PRINT D$; "WRITE ABC, B55"
1040  PRINT 88
1050  PRINT D$; "CLOSE ABC"
1060  END
```

这个程序中的1020行虽把指针移至第1号记录之首，但由于B参数总是从文件开头数起的，所以1030行B的参数仍需用55。

如果在用POSITION 命令时，WRITE 命令不取B选项，程序可写成：

```
1000  D$=CHR$ (4)
1010  PRINT D$; "OPEN ABC"
1020  PRINT D$; "POSITION ABC, R1"
1030  PRINT D$; "WRITE ABC"
1040  PRINT "LI WEI-GUO";",";"NAN";",";"SHANGHAI";
      ",";88
1050  PRINT D$; "CLOSE ABC"
1060  END
```

这个程序的实质是重写第1号记录。

(3) 重写整个文件。

上述(1)、(2)方案有两个缺点：一是要一个个数文件的各个字节，以找出被修改字节号，这对较长的文件，特别是被修改的内容在文件的尾部时，数数是很困难的，弄不好会把文件搞乱。二是当修改的字段中字节数有改动时，上述修改方法也会把文件搞乱；或者把两个字段合成一个字段，或者把一个字段分成了两个字段，甚至对记录也可能产生这类结果。

因此，重写整个文件(即从头到尾重写一遍)，把相应的数据以新的更换旧的。这种方法虽笨一些，但可靠性最高。

4．关于DELETE命令的利用

从头到尾往文件上写一批数据时，这个文件可能已经存在，最好用DELETE命令删除它。删除的目的是为了重写时比较干净，不会有杂乱的东西。但当删除的文件根本不存在时，这个删除命令将执行不下去。

下列形式的程序，可以较完满地解决这些问题（这里我们用CA作为文件名)：

```
1000  D$ = CHR$ (4)
1010  PRINT D$; "OPEN CA"
1020  PRINT D$; "DELETE CA"
1030  PRINT D$; "OPEN CA"
1040  PRINT D$; "WRITE CA"
1050  PRINT…
  ⋮                          } 输往文件的数据
2000  PRINT…
2010  PRINT D$; "CLOSE CA"
2020  END
```

程序中的1010行和1020行的作用是：如果已有这个文件的话，先打开它，再删除它；如果还没有这个文件的话，先建立它，然后删除它。这样执行DELETE命令时，总有那么个磁盘文件（虽然它可能是空的，但名字是有的)。真正建立这个文件的是1030行

开始的程序。

5．关于屏幕显示

上面建立文件的程序，运行时在屏幕上什么也看不到。如果想看到某些显示，则可用命令MON。

这个命令可用于立即执行方式，在运行程序前先敲入该命令，然后再运行你的程序。

MON 命令也可用于延迟执行方式。比如对刚刚写的那段程序，可加入程序行：

```
1005  PRINT D$; "MON C, O"
```

运行时可以看到DOS命令和输出到文件的一个个字符。

不需显示时用NOMON命令来撤销它。不显示时运行效率要高。

四、检索一个顺序文件的例

建立文本文件的目的，不仅仅是为了在磁盘上保存一些信息，更重要的是为了将来的使用。不提供使用的文本文件是没有任何价值的。

检索一个文件，就是从这个文件中搜索你所需要的对象。它是使用任何一个文本文件必不可少的步骤。

检索一个文件，必须知道这个文件在磁盘上存放的格式。也就是说，必须知道你的文件是如何建立的。否则检索出来的数据无法使用，甚至你什么也检索不到。

现在我们设定已有一个名为ABC的文件，它记载了1000个人的情况，每个人的全部数据是一个记录。每个记录有四个字段，依次为某人的姓名、性别、籍贯和工资数，前三个数据为字符串型量，后一个数据为数型量。

如果你指名道姓，要了解某人的情况，那么检索程序可以是：

```
10   INPUT A$
20   N = 0
```

```
30  D$=CHR$ (4)
40  PRINT D$; "OPEN ABC"
50  PRINT D$; "READ ABC"
60  FOR I=1 TO 1000
70  INPUT B$, E$, F$, A
80  IF A$<>B$ THEN 110
90  N=N+1
100 B$ (N) =B$ : C$ (N) =E$ : E$ (N) =F$ : F (N)
    =A
110 NEXT I
120 PRINT D$; "CLOSE ABC"
130 IF N=0 THEN "NO FOUND" : GOTO 170
140 FOR I=1 TO N
150 PRINT B$(I); TAB (18); C$(I); TAB (24); E$(I);
    TAB (34); F(I)
160 NEXT I
170 END
```

由于50行的READ命令的作用，70行的INPUT不是从键盘而是从文件中读入数据。程序中的N，表示同名的人数。

如果想把这1000个人的工资总额求出来，则可用另一个检索程序：

```
10  S=0
20  D$=CHR$ (4)
30  PRINT  D$; "OPEN ABC"
40  PRINT  D$; "READ ABC"
50  FOR I=1 TO 1000
60  INPUT A$, B$, C$, D
70  S=S+D
80  NEXT I
90  PRINT D$; "CLOSE ABC"
```

```
100  PRINT S
110  END
```

这两个检索程序虽不一样，但它所用的命令和语句的顺序是一样的：

OPEN　命令
READ　命令
INPUT　语句
CLOSE　命令

如果把READ改成WRITE，把INPUT改成 PRINT 正好是建立一个文件时命令和语句使用的顺序。

对于检索程序，应注意以下几个问题：

1．用 INPUT 语句从一个文本文件上向内存的程序中读数据时，要遵守这样的规定：一个INPUT语句总是从一个记录的头上读起，把从文件中读出的各个字段分别赋值给INPUT所跟的各个变量名。当执行到INPUT语句结束尾的RETURN符时，把读写指针置于文件的下一个记录之首，以指明下一个INPUT语句将从那里读起。因此：

(1) INPUT语句中各变量的数据类型必须与相应记录中的相应字段的数据类型一致或相容；

(2) 若一个INPUT语句中的变量个数少于相应记录中的字段个数时，多余的字段将被忽略不管；

(3) 若一个INPUT语句中的变量个数多于相应记录中的字段个数时，多余的变量将从下一个记录中继续取得数据。这时，INPUT语句可能会取出两个或更多个记录中的内容。

比如打开ABC文件，想从中取出数据，若用下列程序段：

```
⋮
50  FOR I=1 TO 900
60  INPUT A, B$, C$, D
```

```
70  NEXT I
:
```

这段程序的60行中，INPUT跟的第一个变量名是数型量，而ABC文件中每个记录的首字段都是字符串型量，这将出现类型不匹配的错误，你连一个数据也找不到。

而若用下列程序：

```
:
50  FOR I=1 TO 900
60  INPUT A$(I), B$(I), C$(I), D(I), E$(I)
70  NEXT I
:
```

当I为1时，A$(1)，B$(1)，C$(1)，D(1) 分别得到文件中第一个人的四个数据（0号记录中的4个字段），而E$(1) 得到的是第二个人的姓名(1号记录的0号字段)。这个INPUT结束时，读写指针将移至 2 号记录的头上（即将读写指针向后移过一个RETURN字符),于是当I为2时,INPUT语句将使A$(2),B$(2)，C$(2)，D(2) 分别得到第三个人的四个数据（2号记录的4个字段），E$(2)得到的是第四个人的姓名(3号记录的首字段)。……

这种检索办法，虽也可取得一些数据，终归不是一个好的办法。

2．检索一个文件时,DOS命令和从磁盘文件中来的信息一样不会显示在屏幕上。需要显示时，应给出MON命令。

五、同时打开多个文件的例

假定有500个学生，第一学期每人考六门课,可通过下列程序将每人的6个考分和平均分存到一个磁盘文件(若名为C1)上：

```
10  N=6 : A$="C1"
20  DIM A(500, N)
30  FOR I=1 TO 500
40  S=0
```

```
50   FOR J=1 TO N
60   INPUT A (I, J)
70   S=S+A (I, J)
80   NEXT J
90   A (I, 0)=S
100  NEXT I
200  D$=CHR$(4)
210  PRINT D$; "OPEN"; A$
220  PRINT D$; "WRITE"; A$
230  FOR I=1 TO 500
240  FOR J=0 TO N
250  PRINT A(I, J);",";
260  NEXT J
270  PRINT
280  NEXT I
290  PRINT D$; "CLOSE";A$
300  END
```

运行这个程序时，依次从键盘上输入每个学生的六个考分。这样将在磁盘上建立一个名为C1的文本文件。每个记录表明一个学生的成绩：第0号字段为平均分，其它字段为这个学生的各门考分。

如果这500个学生第二学期每人只考5门课，只需把上列程序的第10行改为：

```
10  N=5:  A$="C2"
```

运行时，一样从键盘上输入每个学生的5个考分。运行结束后将建立一个名为C2的文本文件。

在上述两个文件建立之后，下列程序可建立一个名为C0的文件，其中将记载着这些学生这一学年的平均分：

```
10   D$=CHR$ (4)
20   PRINT D$; "OPEN C1"
```

```
30  PRINT D$; "OPEN C2"
40  PRINT D$; "OPEN C0"
50  FOR I=1 TO  500
60  PRINT D$; "READ C1"
70  INPUT A
80  PRINT D$; "READ C2"
90  INPUT B
100 PRINT D$; "WRITE C0"
110 PRINT A+B
120 NEXT I
130 PRINT D$; "CLOSE"
140 END
```

这个程序中80行的READ命令，会使60行的READ命令失效；100行的WRITE命令又使80行的READ命令失效；NEXT 回来之后，60行的 READ 命令重新下达，发生作用，而它却使100行的WRITE失效。这正好是我们所需要的。

从这个例子中我们看到了同时打开几个文件的方法，并且也看到了文件的另外一种使用方法：可以把一个程序的运行结果（或中间结果）存入文件中为另外一个程序所使用（这个新程序也可建立新文件。如 C0）。这样也可以实现程序之间数据的传递。这比 CHAIN 的好处是传递的数据有选择性，不需传递的数据全部可以弃之不管。此外数据在盘上，不仅容量大，而且能脱机保存。

§4. 自动APPLE：EXEC命令的使用

EXEC也是一个DOS命令，其格式是：

EXEC f〔, R_p〕〔, V_v〕〔, S_s〕〔, D_d〕

EXEC命令与RUN 命令很类似，它们都可以用来使文件投入运行。但RUN只能运行BASIC程序，而EXEC 可以运行一个含有

命令（包括DOS命令）或BASIC程序的文本文件。

一、EXEC的功效

把一些命令或程序行输出到一个顺序文本文件中去，这同样可以通过建立一个文本文件的方法实现。

当这种文件建立好之后，用EXEC 命令来执行它，这时从文件中读回内存的信息，像从键盘上输入的一样：如果是立即执行的命令，就执行它；若是带行号的程序行，就把它们送入内存的程序区。对于从文件来的程序，通过RUN命令也同样可以运行。

让我们先看一个例子：

假定有如下一个程序：

```
10  PRINT "*TIEMS*"
20  PRINT "APPLE:"; TAB(10);.35
30  PRINT "PEAR:"; TAB(10);.28
40  PRINT "ORANGE:";TAB(10);.46
50  END
```

运行这个程序会显示出三种水果的名称和价格。假定以FRUIT为程序名，可以先把它存在磁盘上。

现在，如果我们打算依次做以下三件事：

(1) 运行FRUIT程序；

(2) 把内存中当前程序的20行至40行列出来；

(3) 把当前盘文件目录列出来。

通常我们是依次敲入下面三个命令分别实现的：

```
RUN FRUIT
LIST 20, 40
CATALOG
```

如果把这三个命令作为程序输出的数据，存入到一个顺序文件中，例如取文件名IT，这件事可以通过下列程序的运行来实现：

```
10  D$ = CHR$ (4)
```

```
20  PRINT D$; "OPEN IT"
30  PRINT D$; "WRITE IT"
40  PRINT "RUN FRUIT"
50  PRINT "LIST 20, 40"
60  PRINT "CATALOG"
70  PRINT D$; "CLOSE"
80  END
```

这时再使用EXEC命令：

```
EXEC IT
```

这个DOS命令会自动打开名为IT的文件，并从中顺序将一个个记录送入内存(好象从键盘输入的一样)。由于文件IT中是三个命令，它就一个一个地执行。先运行FRUIT程序，然后列出当前内存程序的20行到40行（在此，内存程序就是FRUIT)，最后把当前盘的目录列示出来。

由此可见，EXEC可以实现由文本文件向内存输入以代替从键盘上向内存的输入。当一个文本文件由一些命令构成时，只要用EXEC命令运行这个文件，这些命令可以顺序地一个一个地进入内存，并执行它们，从而变成一部自动的苹果机（不需用户来敲入这些命令)。

现在我们再来看一个例子。

假定内存中已有一个BASIC程序，我们对其中的某一段程序(比如2000行到10000行）很感兴趣，认为很有用。那么也可以通过运行下面的程序，把这一段程序保存到一个文本文件上（比如文件名叫LISTING)：

```
1  D$ = CHR$ (4)
2  PRINT D$; "OPEN LISTING"
3  PRINT D$; "WRITE LISTING"
4  POKE 33,  30
5  LIST 2000, 10000
```

```
6  PRINT D$; "CLOSE"
7  END
```

这个程序中的第4行、第5行虽无PRINT语句输出，但在执行LIST命令时，实际上执行了BASIC解释程序中对PRINT的解释执行部分。换言之，实质上是执行了PRINT语句。否则，那些程序行怎么又输出到屏幕上了呢？在这个程序里，由于第3行的作用，使这些程序行不是送往屏幕，而是送给文件LISTING了。另外，用LIST将程序送屏幕显示时，并未占满屏幕，而是从左边开始向右，行宽只有30个字符位置。因此用了第4行的改变窗口宽度命令，以避免存入文件时在每个终端行上加入10个多余的空格符。

当如上的LISTING文件建立好以后，执行下列命令：

```
EXEC LISTING
```

会将LISTING中的2000到10000行程序装入内存中，为其它程序使用，而不必键盘输入。

可能有人会想，把那些有用的程序段用SAVE命令存入磁盘，以后LOAD出来，不也可以为别的程序用吗？这当然是可以的。但用EXEC运行一个存有程序行的文件，要比LOAD功能强得多。至少可以有以下三点：

(1) 当一段新的程序输入内存后，再LOAD出一些有用的程序行，将会把已输入机器的程序冲掉。而EXEC命令从文件中读入的程序行则会加在内存中原有的程序中。因为EXEC命令读入的内容，相当于从键盘上敲入的一样。

(2) LOAD进入内存的程序段，只能来源于一个BASIC程序。因为你没有办法将两个源程序存入一个程序名中。而EXEC运行的文本文件，其数据（在此为程序行）可以来至不同的BASIC程序。因为一个文件建立好之后，还可以再次增加数据。这样，一个文本文件可以收集多个程序的有用部分，经EXEC后，在内存中构成一个新程序。当然这种文本文件要组织好才能使构造出来的

程序有价值。

(3) 浮点BASIC和整数BASIC 分属两种不同的文件类型，通过SAVE和LOAD命令，不会改变任何一个 BASIC 程序的类型属性。而通过建立文件的方法，将某些程序行写入文件后，它们的属性为文本文件类型，失掉了BASIC的属性。用EXEC 把它们装入内存时，当时系统属性就是它们的属性。换言之，如果当时机器属于浮点BASIC系统，那么EXEC进来的程序便属浮点 BASIC程序；若是当时机器处在整数BASIC系统下，EXEC 来的程序属于整数BASIC程序。这样，不管采自什么属性的BASIC 程序行构成的文本文件，用于整数BASIC或浮点BASIC都是可以的。

三、EXEC的特点

知道下列特点，可以帮助你用好EXEC命令：

1. 由EXEC打开的文件，只有在它的全部信息输入内存之后(如果这些信息是命令，则要全部执行完之后)才自行关闭。CLOSE命令是关闭不住它的。并且在任何时刻只有一个 EXEC 打开一个文件，如果EXEC中又有EXEC时，关闭掉第一个EXEC打开的文件，由第二个EXEC打开另一个文件。

2. EXEC文件中若有 RUN 命令，EXEC 命令会耐心地等待它，RUN结束后，再执行EXEC文件中的下一个命令。

3. EXEC打开一个文件时，若是运行一个BASIC程序，其中任何一个INPUT语句不理会键盘的输入。若INPUT得到一个DOS命令，将立即执行此DOS命令，这将产生预料不到的后果。

4. EXEC命令允许R_p可选项，它是相对于EXEC 文件头的记录位置，实际上它是一个绝对记录位置(与POSITION中的意义不同)。

5. EXEC命令不会被CTRL-C中止。

§5. 随机文本文件

一、随机文本文件的存放格式

顺序文本文件在磁盘上的存放格式是一个接着一个，每个记录以回车符为界，记录长度是不固定的。因此要想随机地使用或修改每个记录时，要一个个数回车符的个数或字节的个数，这是很不方便的。

随机文件则不同。它在磁盘上的存放格式总是以记录为单位的，而且每个记录都有固定的长度。这为随机存取提供了方便。

现在假定有下列一些数据存在一个随机文件上，记录长度为5：

7↙AT↙ONE↙BL, W↙

那么它们在磁盘上的存放格式是：

字符	7	↙				A	T	↙		
ASCII码	55	13	00	00	00	65	84	13	00	00
字节号	0	1	2	3	4	5	6	7	8	9
字段号	0					0				
记录号	0					1				

字符	O	N	E	↙		B	L	,	W	↙		
ASCII码	79	78	69	13	00	66	76	44	87	13	00	00
字节号	10	11	12	13	14	15	16	17	18	19	20	21
字段号	0					0			1			
记录号	2					3						

二、随机文本文件命令

1．OPEN命令

OPEN命令的格式是：

OPEN f, L [, V_v] [, S_s] [, D_d]

其中L项用来指明指定文件的记录长度。如L5表明每个记录占五个字节的位置。

除L项外，与顺序文件中的OPEN无异。

2．WRITE命令

WRITE命令的格式是：

WRITE f [, R_r] [, B_b]

其中R项用来指明从第几个记录开始写。如从第10号记录开始写，R项用R10，当R项不选时按R0处理。B选项用以指明从指定记录的第几个字节写起(b值是相对位置值)。

此命令功能是指明其后的 PRINT 输出写往指定文件中从第r号记录的第b号字节开始的地方。

3．READ命令

READ命令的格式是：

READ f [, R_r] [, B_b]

这个命令的R_r、B_b意义同上。

它指明从指定文件的第 r 号记录的 b 号字节开始读入内存中INPUT语句中的变量。

值得注意的是：在随机文本文件中，任何一个向文件输出的PRINT语句，必须跟在一个WRITE命令之后；同样，任何一个从文件上读取数据的INPUT语句，必须跟在一个READ命令之后。并且通过WRITE或READ中命令中的Rr选项指定读写记录号（即内部指针位置）。否则文件实际被当作顺序文本文件处理。

三、随机文本文件的例

下列程序可以建立一个随机文本文件BCD：

```
10    DATA …  }  (仍用建立一个顺序文件例的1000个
⋮             }  人的姓名、性别、籍贯、工资数据)
1000  DIM A$ (1000, 3), B (1000)
1010  FOR I=1 TO 1000
1020  FOR J=1 TO 3
1030  READ A$ (I, J)
1040  NEXT J
1050  READ B(I)
1060  NEXT I
2000  D$=CHR$ (4)
2010  PRINT D$; "OPEN BCD,  L100"
2020  FOR I=1 TO 1000
2030  PRINT D$; "WRITE BCD, R"; I-1
2040  PRINT A$(I, 1);",";A$(I, 2); ",";A$(I, 3);",";
      B(I)
2050  NEXT I
2060  PRINT D$;"CLOSE"
2070  END
```

运行这个程序后，将在磁盘上建立一个名为BCD的随机文本文件，共1000个记录，每个记录长度都是100个字节，其中为四个字段。

请读者注意：若将上述程序中修改如下两行：

```
2020  PRINT D$; "WRITE BCD"
2030  FOR I=1 TO 1000
```

运行后得到的BCD文件，将是一个顺序文本文件。

如果要改写BCD文件中的某个记录（如501号），可运行下面的程序：

```
10  D$=CHR$ (4)
20  PRINT D$; "OPEN BCD, L100"
30  PRINT D$; "WRITE BCD, R501"
```

```
40  PRINT (被改写的内容)
100 PRINT D$;"CLOSE BCD"
```

被改写的内容可写在40行的PRINT语句中。改写时最好改写整个记录（如记录内又分字段，要强行加上逗号）。这样改写后的记录不会留下原来记录的多余部分。

如果改写某个记录的某一字段，可找出此字段的首字符在该记录中的相对位置，用B*b*加以设定。

针对上述方法建立的BCD文件，写一个检索程序是很方便的。比如我们查找工资在100元以上的人员名单，可运行下列程序：

```
10  DIM A$ (1000), B(1000)
20  D$ = CHR$ (4) : N = 0
30  PRINT D$; "OPEN BCD, L100"
40  FOR I = 1 TO 1000
50  PRINT D$; "READ BCD, R"; I - 1
60  INPUT A$, B$, C$, D
70  IF D >= 100 THEN N = N + 1 : A$(N) = A$ : B(N) = D
80  NEXT I
90  PRINT D$; "CLOSE BCD"
100 FOR I = 1 TO N
110 PRINT A$(I), B(I)
120 NEXT I
230 END
```

同建立随机文件时一样，如改写下列两个程序行：

```
40  PRINT D$; "READ BCD"
50  FOR I = 1 TO 1000
```

运行时将把BCD当作顺序文件来读，导致出错。

文件是一种很有用的手段。初学者使用时可能出现一些预想不到的错误。但不要灰心，多实践几次总会用好的。

习　题

（一）BASIC中的SAVE、LOAD、RUN同DOS中的SAVE、LOAD和RUN有什么区别？

（二）说明顺序文本文件和随机文本文件各有什么优缺点，在什么情况下使用顺序文件较好，在什么情况下使用随机文件较好？

（三）有以下一个BASIC程序，说明运行时能在屏幕上看到些什么？磁盘上会产生一个什么文件？其ASCII码序列是什么（用对应字符序列写出）。假定运行中你先后从键盘上敲入如下一些字符串：

HERE'S STRING 1↙
AND MY SECOND STRING↙
ON WE GO↙
ENOUGH ALREADY! ↙
↙
TEST ↙

程序如下：

```
5   D$=CHR$ (4) : REM CTRL-D
10  PRINT D$; "MON C, I, O"
20  DIM A$(100) : I=0
30  HOME : TEXT
40  PRINT "YOU GET TO TYPE ONE STR INGAT A
    TIME"
50  PRINT "A STRING MAY HAVE UP TO 239 CHARAC
    TERS"
60  PRINT "THIS PROGRAM LETS YOU WRITE TEXT
    FILES"
70  PRINT : I=I+1
80  PRINT " (PRESS THE RETURN KEY TO QUIT."
90  PRINT "TYPE STRING #";I;":";
```

```
100  INPUT " ";A$ (I)
110  IF A$ (I) < > " " THEN 80
120  PRINT
130  INPUT "WHAT FILE NAME?";N$
140  PRINT D$; "OPEN";N$
150  PRINT D$; "DELETE";N$
160  PRINT D$; "OPEN";N$
170  PRINT D$; "WRITE";N$
180  PRINT I-1
190  FOR J=1 TO I-1
200  PRINT A$ (J)
210  NEXT J
220  PRINT D$; "CLOSE";N$
```

（四）设磁盘上已有(三)题建立的 TEST文件，请回答下列程序运行后会产生什么结果。程序运行时从键盘输入:TEST↙

程序如下：

```
10   D$=CHR$ (4)
20   PRINT "THIS PROGRAM RETRIEVES TEXT  FILES"
30   PRINT "CREATED BY THE'MAKE TEXT' PROGRAM"
40   PRINT "MON I IS IN EFFEAT"
50   PRINT
60   INPUT "NAME OF TEXT FILE?"; P$
70   PRINT D$; "MON I"
80   PRINT
90   PRINT D$; "OPEN";P$
100  PRINT D$; "READ";P$
110  INPUT I
120  DIM A$ (I)
130  FOR J=1 TO I
140  INPUT A$ (J)
```

```
150  NEXT J
155  PRINT D$; "CLOSE";P$
160  PRINT D$; "NOMON I"
170  END
```

（五）假定唱歌用的机器语言子程序已送内存的770到790号单元中。下列程序运行后会产生什么结果？如果以后开机时执行命令

EXEC CODE-POKES

又会产生什么结果？

```
10  D$ = " " : REM CTRL-D
20  PRINT D$; "OPEN CODE-POKES"
30  PRINT D$; "DELETE CODE-POKES"
40  PRINT D$; "OPEN CODE-POKES"
50  PRINT D$; "WRITE CODE-POKES"
60  LINENUMBER = 10
70  FOR PLACE = 770 TO 790
80  COUNTER = COUNTER + 1
90  IF COUNTER = 10 THEN COUNTER = 1
100 IF COUNTER< >1 THEN 140
110 PRINT
120 PRINT LINENUMBER;
130 LINENUMBER = LINENUMBER + 10
140 PRINT "POKE";PLACE;",";PEEK (PLACE) ;":";
150 NEXT PLACE
160 PRINT
170 PRINT D$; "CLOSE CODE-POKES"
190 END
```

（六）假定已由运行下列程序建立了一个名为RA-FILE的文本文件。请写一个程序。把其中前三个字符为"DOS"的记录找出来。

```
10  D$ = CHR$ (4)
20  PRINT D$; "OPEN RA-FILE"
```

```
30   PRINT D$; "DELETE RA-FILE"
40   PRINT D$; "OPEN RA-FILE, L30"
50   FOR I=12 TO 15
60   PRINT D$; "WRITE RA-FILE, R";I
70   PRINT "NAME ADDRESS";I
80   NEXT I
90   PRINT D$; "WRITE RA-FILE, R 13"
100  PRINT "DOS"
110  PRINT D$; "CLOSE RA-FILE"
```

（七）假定某人已写好一个数据处理程序，行号从100到10000。它每次输入一批数据，产生50个中间结果，存放在A数组的第0号到49号元素中。下次运行时，这50个中间结果又作为原始数据参加运算，再产生新的50个中间结果。在不使用操作系统命令时，每次运行完结后，需把50个中间结果打印出来，下次再计算时又需将它们一一从键盘敲入。这是很麻烦的。而他又需要这样做几十次。请你通过使用操作系统命令，帮他减去这种麻烦。

（八）运行系统主盘上的随机文件示范程序，其名为RANDOM

（九）运行系统主盘上的EXEC示范文件EXEC DEMO。由于系统主盘是整盘硬保护的，要把这个程序用LOAD和SAVE命令转到一个没有写保护的盘上，再运行它。

（十）使用文件写一个小型的实用的管理程序包，如学分管理、人事管理等。它至少有两个程序组成：建立和修改文件，检索文件。若能用EXEC运行的文件将它们统管起来更好。

第十一章　6502指令系统及监控命令简介

我们在第一章里提到过，机器语言是计算机能直接认识的唯一语言。汇编语言、算法语言等都是经过运行一套特定的机器语言程序，计算机才知道它们的意义的。完成任何高级语言的功能，最终必将经过执行机器语言程序才能实现。

所以想要更清晰地理解一个 BASIC 程序在机器运行时的细节，或者更完满地处理BASIC程序，以至解释、处理一些预料之外的问题，读一下系统程序（如BASIC解释程序、操作系统、监控程序）将会得益不浅。而这些程序都是由一条条指令写出来的，这就要了解它的指令系统。

还不止此。在任何一个BASIC程序中，如果用一些机器语言子程序加以配合，可以实现 BASIC 语言无法实现的功能（如果没有发音的机器语言子程序的使用，用BASIC语言实现唱歌是办不到的）。甚至有些应用程序为了加快运行速度，大部分或全部用机器语言来写。不熟悉指令系统是无法进行这类工作的。

监控命令为机器语言的使用（如机器语言程序的输入、修改、读出、检查、传送等）提供了良好的手段，不会使用将是一大缺陷。

这一章我们只是对这些内容的基本部分作概略的介绍。读者若希望进行深入的研究和了解，可参阅北师大出版社85年3月出版的荣树熙副教授所编《6502微处理机原理与应用》一书的有关章节。

另外，需要说明，在 BASIC 程序中所用的数型常量，都是用十进制数表示的。在这一章及下一章的大部分内容里，我们大

都用十六进制的数来表示。象前面指出过的那样，用十六进制表示一个数时，我们通常在它的前面加一个字符$，以示区分。对于十六进制不熟悉的读者，要进一步熟悉它，学会使用它。

§1. 6502的寄存器

6502微处理器有六个寄存器，现分别介绍如下：

1.累加器 (accumulator)，用大写字母A作标志。它可表示8位二进制数。算术运算、逻辑运算等都在这里进行。

2.变址寄存器 (index register)，共两个，每个可表示8位二进制数，分别用大写字母X，Y作标志。变址寄存器可进行加1和减1操作，它们主要用于控制变址值（详见下节介绍）。

3.指令计数器 (program pointer)，用 PC 标志，可存放16位二进制数，左8位表示地址的高位部分，用 PC_H 标示；右8位表示地址的低位部分，用PC_L 标示。程序运行中，将根据这里给出的内存地址，转到内存的相应地址取出指令执行，同时又自动将下条指令的首地址送入指令计数器中，这样可使程序继续运行下去。

4.堆栈指示器 (stack pointer)，用S作标志，可表示8位二进制数。

堆栈的概念前面已提到过。这个堆栈指示器是为系统程序用的，它安排在内存中$100到$1FF的范围内（这里的1是高位地址，00和FF是低位地址），由于这里高位部分恒为1，不需再行标出，所以只用S的8位二进制数，用以指明栈指针位置。

5.标志位寄存器 (processor status register)，用P作标志，可存放8位二进制数。其中第5位未用，其它各位分别标志7种状态：

第0位：C(carry) 标志位，表明进位位状态。两个8位二进

制数做加法运算时，可能产生进位；做减法运算也可能需借位，C标志位就是用来标明这种情况的：有进位或无借位时，C位用1表示；无进位或有借位时则用0表示。

第1位：Z(zero) 标志位，表明刚才运行结果是否为0。结果为0时，Z标志位用1表示；结果非0时，Z标志位用0表示。

第2位：I(interrupt disable) 标志位，表明是否允许中断。I为1时，表明禁止中断（屏蔽中断）；I为0时，表明允许中断。

第3位：D(decimal mode) 标志位，D为1时，采用十进制方式运算；D为0时，表示用十六进制方式运算。

第4位：B(break command) 标志位，标志中断状态。为1时表明中断；为0时表明未中断。

第6位：V(overflow) 标志位，表明溢出状态。为1时，表明有溢出；为0时，表明无溢出。

第7位：N(negative) 标志位，表明一个数是否为负（一个八位二进制数，最高位为1时为负，否则为正）。为负时，N取1，非负时，N取0。

6502各寄存器示意图见图11—1

§2. 6502指令分类

6502指令有56条，可将它们归为八大类，每大类中又可分为几小类。

一、传送指令

它们的功能就是传送数据，共12条，又可分两个小类：

1.内存和寄存器之间的传送。有以下六条：

LDA	M→A
LDX	M→X
LDY	M→Y

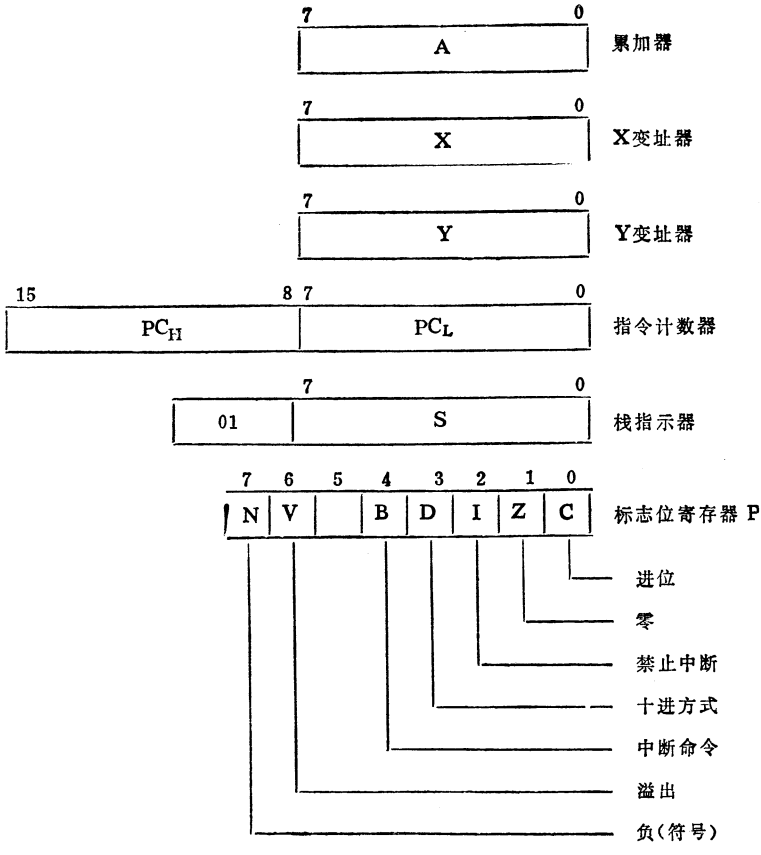


图11—1

STA	A→M
STX	X→M
STY	Y→M

上面的左栏，是指令的汇编助记符（汇编码），右栏是该指令的意义。A表示A累加器，X表示X变址器，Y 表示 Y变址器，M表示内存某一个单元号码（有时 M 可能就是一个数）。箭头表示

数据传送方向。

2.寄存器和寄存器之间的传送，有以下6条：

TAX	A→X
TXA	X→A
TAY	A→Y
TYA	Y→A
TSX	S→X
TXS	X→S

其中S为堆栈指示器中的数（即栈指针位置）。

二、算(术)逻(辑)运算指令

共11条，又分两小类。

1.算术运算指令，8条：

ADC	$A+M+C\rightarrow A,\ C$
SBC	$A-M-\bar{C}\rightarrow A$
INC	$M+1\rightarrow M$
DEC	$M-1\rightarrow M$
INX	$X+1\rightarrow X$
DEX	$X-1\rightarrow X$
INY	$Y+1\rightarrow Y$
DEY	$Y-1\rightarrow Y$

这类指令中的前两条，为带进位位的加和减。作加法时，要把进位位的值当作末位值加上去（好象从后边进位上来的）；加时也要将产生的进位送入C标志位中。第二条中的 $\bar{C}$表示C的非(或反)。作减法时有借位相当于无进位（C为0）。

2.逻辑运算指令，3条：

AND	$A\wedge M\rightarrow A$
ORA	$A\vee M\rightarrow A$
EOR	$A\veebar M\rightarrow A$

“∧”为逻辑**与**运算符，“∨”为逻辑**或**运算符，“∀”为**逻辑异或**运算符。三条指令都是位对位的运算方法。当两个对应位均为1时，“∧”结果才为1；否则为0。两个对应位中只要有一位为1时，“∨”的结果为1；只有两个对应位均为0时，“∨”的结果才为0。两个对应位不同时，“∀”的结果为1；否则为0。八位全部作完后，结果在A累加器中。

三、比较指令

共4条：

CMP	A－M
CPX	X－M
CPY	Y－M
BIT	A∧M，M_7→N，M_6→V

前三条指令的比较，相当于作一次减法，但其结果并不送往任何寄存器或内存。比较的结果可正，可负，还可能是0。因此三条指令比较的结果，会改变相应标志位：N（结果为负时置1，为正时置0）、Z（结果为0时置1，非0时置0）和C（前者≥后者时，置1，否则置0）。

最后一条指令，相当于作一次“∧”运算，但结果也不送任何寄存器或内存，仅改变标志位：Z（结果为0时置1，非0时置0）、N（取M的第7位）和V（取M的第6位）。

四、置标志位指令

共7条：

CLC	0→C
SEC	1→C
CLI	0→I
SEI	1→I
CLD	0→D
SED	1→D

CLV　　0→V

从这一类指令和上一类指令中可以看到，标志位因各指令执行情况而改变。每条指令对标志位的影响能力是不一样的。详细情况可参阅后面给出的指令表。

五、栈指令

共4条：

PHA	A↓	即A→M_S，S－1→S
PLA	A↑	即S＋1→S，M_S→A
PHP	P↓	即P→M_S，S－1→S
PLP	P↑	即S＋1→S，M_S→P

其中S为栈指示器。M_S 为栈指针所指的栈地址。

指令PHA，功能是使累加器A中的数进栈。也就是将A中的数送入S所指明的栈地址中（即A→M_S），然后S－1→S，使S指向较小的一个相邻单元，以便接收新数进栈。

指令PLA，功能是将栈顶上的一个数取入A中。也就是使S－1→S，使栈指针正好指向栈顶上那个数，然后将此数送入A中，即M_S→A。这时的S值，正指向等待一个新进栈数的位置。

PHP和PLP也是进栈和出栈。只不过它不是对累加器A中的数，而是对标志位寄存器中的数（即各标志位）。

六、转移指令

共13条，分两小类：

1.条件转移指令，8条：

BCC	转移条件：C＝0
BCS	转移条件：C＝1
BNE	转移条件：Z＝0
BEQ	转移条件：Z＝1
BPL	转移条件：N＝0
BMI	转移条件：N＝1

BVC　　转移条件：V = 0

BVS　　转移条件：V = 1

2.无条件转移指令，5条：

JMP　　无条件地转移到给定的地址

JSR　　转入子程序

RTS　　退出子程序

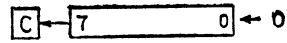
RTI　　中断返回

BRK　　强迫中断

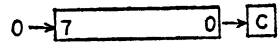
七、移位指令

共4条：

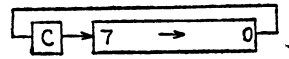
ASL　　算术左移



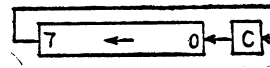
ASR　　逻辑右移



ROR　　循环右移



ROL　　循环左移



八、空操作

只一条：

NOP　　什么也不做。

§3. 6502的寻址方式

上面我们列出了6502的56条指令，并将它们分八类。这56条指令，各有各的功能。计算机的所有动作，都必须在这56条指令的支持下才能完成。

但是，我们在上节中是用汇编助记符给出这56条指令的。也就是说，它只是指令的代号，并不是机器能直接认识的。机器能

认识的只是由不同的0和1组成的数。换言之，只有代表1的高电位和代表0的低电位的不同组合，才能驱动电路做不同的动作。这就是说，至少得找出56个不同的数，来代表56条指令的不同动作。控制机器产生不同动作的码子，我们称之为“操作码”。

一条指令仅仅一个操作码往往是不行的。比如LDA，它可以把内存某个单元中的数送入累加器A中，这就需要在操作码之后给出一个地址号码，指明操作数来自何。处所以通常称操作码后边的内容为地址部分。一个地址一般由两个字节组成，但也可由一个字节给出。地址部分还可能不是地址号码，而是一个具体的数，意思是把给出的数送入累加器A中。这样地址部分就出现了几种可能性。于是假定LDA只用一个操作码，就无法处理上述种种不同情况。事实上，地址部分的情况还要复杂得多，并且有些指令还不需要地址部分。这样，一条指令就需要有几个不同的操作码，用以区分它们构成地址的不同方法，并把这个地址寻找出来。这就被称为不同的寻址方式。

6502指令系统共有13种寻址方式，但对每条指令而言，只具备其中的一部分，这样组合起来，共有151个操作码。因而操作码用一个字节来表示。

现将6502的十三种寻址方式介绍如下：

1.立即寻址——IMMEDIATE

立即寻址方式，指的是操作码后边的字节中的内容，就是一个被操作数本身。这个数又叫立即数。

例如指令

A9 10

这是两个字节构成的一条指令。A9是操作码，表示取一个数装入累加器A中，第二个字节中的10，就是这个立即数。这条指令执行后，累加器A中的值就是A9后边的这个10。

这条指令用汇编语言写出，是

LDA　#$10

立即寻址方式的指令，长度为两个字节。

2.绝对寻址——ABSOLUTE

绝对寻址方式是指：操作码后边跟的是操作数的地址。

例如指令及其汇编为：

AD　12　3A　　LDA　$3A12

这条指令的功能是把内存中 $3A12 号单元中的数装入累加器A中。

可以看出，前面举的那条二字节指令和这条指令的功能都是往累加器A中置数，所以它们的汇编符都是LDA，但它们的操作码却不同。正是因为操作码的不同，寻找地址的方式才不一样，从而成为不同的寻址方式。

另外，在这里又看到了，凡用两个字节表示一个地址时，总是先给出低位字节，再给出高位字节。通常称作高位在后，低位在前。

绝对寻址方式的指令，长度都是三字节。

3.零页寻址——ZERO　PAGE

内存中的$0000到$00FF，称为0 页，类似地把$0100到$01FF称为 1 页，其余类推。零页寻址方式指的是：操作数在0 页范围内，因此高位地址部分不再给出。

如指令

A5　23　　LDA　$23

这条指令等同于指令

AD　23　00　　LDA　$0023

但使用零页寻址方式，少占用一个字节。

零页寻址方式的指令，长度均为二字节。

4.累加器寻址——ACCUM

累加器寻址方式指的是：操作数就在累加器A中。

比如指令

2A　　ROL

其功能是将A中的数连同进位标志位C一起循环左移一位。

累加器寻址方式的指令，长度为单字节。

5.隐含寻址——IMPLIED

所谓隐含寻址，指的是地址已为操作码所隐含，不用再指明。从广义上讲，累加器寻址也是一种隐含寻址。但这里的隐含地址是专指除累加器以外的其它寄存器或标志寄存器中的各标志位。

例如指令

CA　　DEX

其功能是将变址寄存器X中的值减1后，再存入变址寄存器 X 中。就是使变址器X作减 1 操作。

再如指令

18　　CLC

其功能是置标志位寄存器中的C 标志位为 0 （可简单地说置C标志位为 0 ）。

隐含寻址方式的指令，长度也都是单字节。

6.绝对X变址——ABS,X

绝对X变址是指：将绝对地址加上 X 变址器中的值构成操作数的有效地址。

例如指令

BD　3A　2F　　LDA　$2F3A,X

它把 $2F3A加上X变址器中的当前值，作为操作数的绝对地址。如果X变址器中的值为 $20，那么这条指令将把内存中 $2F5A中的数送入累加器A中。若是X中的数为 $31，那么这条指令送入A中的将是 $2F6B中的数。

可见，同样一条指令，因X 中的数不同，操作数的地址就改变了，这就是把 X 寄存器称为变址器的原因。对于Y寄存器，也

具有改变一条指令的地址的功能，所以也是变址器。

绝对X寻址方式的指令，长度都是三字节。

7.绝对Y变址——ABS,Y

绝对Y变址和绝对X变址非常类似，只不过它的变址值取自Y变址器就是了。

比如指令

B9 26 3B　　LDA $3B26,Y

若Y中的数为$23，那么这条指令将把内存中$3B49号单元的内容送入累加器A中。

绝对Y变址寻址方式的指令，长度都是三字节。

8.零页X变址——Z.PAGE,X

这类寻址方式是将给出的零页地址加上X变址器中的数作为操作数的地址。

如指令

B5 A3　　LDA $A3,X

假定X中的值为$04，这条指令将把$A7中的数送入A中。

请注意，零页变址只能产生零页地址。还拿这条指令来说，当X变址器中的值为$62时，操作数的有效地址不是$105，而是$05，就是将$05号单元中的数送入A中。系统如此设计，目的是保证系统堆栈不会因变址操作而破坏。

零页X变址方式的指令，是二字节的。

9.零页Y变址——Z.PAGE,Y

同零页X变址类似，变址值取自Y的当前值。

例如指令

96 B0　　STX $B0,Y

如果变址器Y中的值为$29，操作地址就是$D9，这条指令的功能是把变址器X中的数送入内存的$D9号单元保存。

遵从零页变址只能产生零页 地址的规则，当Y中的数为$73

时，这条指令的执行结果是，把X中的数送往$23号单元保存。

零页Y变址方式的指令也是两字节的。

10.间接寻址——INDIRECT

这种寻址方式的指令只有一条，共有三个字节，用于无条件转移。操作码是6C，后两个字节中存放一个地址，但它并不是转移的地址，真正转移的地址存放在这个地址指明的内存单元中。换言之，指令的后两个字节是地址的地址。如象日常生活中，你问我张三的住址，我这样告诉你："请你问李四，李四知道张三住何处。"并且我确实告诉了你李四的住处。这样你从我这里得到的不是直接地址，而是转了一道弯的间接地址，也就是地址的地址。

比如指令

6C 25 36 JMP ($3625)

这是一条无条件转移指令，间接地址为$3625。

然而$3625是一个单元号码，只能存放从$00到$FF共$100个数（十进制256个数），用它作地址显然是不够用的。所以象这类指明某个单元内的数为一个地址时，总是把指明的这个单元中的数，作为地址的低位部分，而把后续单元中的数，作为地址的高位部分（这正符合低位在前、高位在后的约定）。

现在回过头来再说那条无条件转移指令。如果$3625号单元中的数是$7F，而$3626号单元中的数是$6D，那么那条转移指令的执行结果是使程序转移到$6D7F去继续运行。实际上这条指令是向指令计数器PC中填入$6D7F这样一个数（即把$3625中的数送PC_L，把$3626中的数送$PC_H$）。

11.先变址的间接寻址——（IND，X）

这种寻址方式是，先对指令中的地址部分进行X变址，然后再作为间接地址使用。

这类指令的长度都是双字节，因此地址部分给的是一个零页地址，做X变址时也只能是零页X变址，遵从零页X变址的方法。

比如指令

81 35　　STA ($81,X)

要先做零页X变址，就要看X变址器为何值。现假定 X 中的数为$23，则变址后为$58（零页地址）。

再做间址，这又要看$58和$59中各为何值。若$58中为$32，$59中为$3F。那么间址后的地址为$3F32。

于是这条指令的功能就是将累加器A中的数送往内存的 $3F32号单元中保存。

若此时X中的数为$FE，X零页变址后为$33，假定$33 号单元中为$20，$34号单元中的数为$6E，这条指令是将 A中的数送内存的$6E20号单元保存。

12.后变址的间接寻址——(IND),Y

这种寻址方式是先间接寻址，后做Y变址。

这类指令也都是双字节指令，所以给出的地址为零页地址，且只是低位字节，高位字节为后续地址号。

比如指令

91 3A　　STA ($91), Y

执行这条指令时先间址。假定$3A号单元中的数是$46，$3B号单元中的数为$7F，间接寻址后的地址为$7F46。

下一步再做Y变址。若Y变址器中的数为$54，那么Y变址后的地址为$7F9A。

于是所给指令的执行结果是把累加器A 中的数，存入内存的$7F9A号单元中。

13.相对寻址——RELATIVE

这种寻址方式，仅用于条件转移指令。指令长度为双字节。首字节为操作码，次字节为相对转移地址，也就是相对位移量。其值可正可负，符号由第7 位（最左边那位）明指（1 为负；0 为正）。

相对地址是由指令计数器（PC）中的数 和相对位 移量相加而得到有效地址的。

假如有下列连续三条指令：

```
AD 00 20 LDA $2000
D0 02    BNE
A9 20    LDA #$20
8D 00 25 STA $2500
00       BRK
```

这是一个不完整的程序段。要想运行它，必须把它们送入内存中一片连续的单元中。假定把这十一个字节依次送入内存的$300至$30A 号单元中，那么各条指令的首地址将分别在$300，$303，$305，$307和$30A号单元中。为了阅读方便，通常总是把一条指令写作一行，把操作码（即指令首地址）所在的单元号码先列出来，其后用一“-”号，再将整条指令写出。于是上列十一个字节构成的五条指令，可以写成如下格式：

```
0300-  AD 00 20 LDA $2000
0303-  D0 02    BNE $0307
0305-  A9 20    LDA #$20
0307-  8D 00 25 STA $2500
030A-  00       BRK
```

这里的第一条指令是把$2000号单元中的数送入累加器A中，第三条指令是把数$20送入累加器A中，第四条则是把累加器A中的数送入$2500号单元中，最后一条是产生中断，它会把标志位寄存器中的B标志位（第4位）置成1。

我们这里要介绍的是第二条指令。它是一个条件转移指令，条件是非0，其条件成立与否取决于第一条指令执行的结果。当第一条指令从$2000号单元取出的数为0时，非0条件不满足，不产生转移动作，程序接着执行第三条指令。当$2000取出一个

非 0 值时，非 0 条件满足，程序进行转移。

这条指令给出的相对位移量为02，意思是向前跳过两个字节，转到＄307号单元去执行第四条指令。

它是如何判别这种转移呢？转移地址又是如何算出来呢？这离不开指令计数器的作用。

当运行这段程序时，开始要设法将程序的首地址 ＄300 置入PC中（以后会见到这类命令），于是机器依PC中的值，控制执行＄300中的指令，同时将该指令的长度（字节数）与PC 中的值相加，再送入PC中，这时PC中应为＄303，正好是第二条指令的首地址。第一条指令执行完后，依PC 中的值又去执行＄303号单元开始的第二条指令，同时把第二条指令的长度加入 PC 中，使PC变成了＄305。第二条指令要求检查Z标志位的状态（第一条指令正好会改变这个标志位：＄2000号单元的数为 0 时，置 Z 标志为 1；＄2000号单元的数为非 0 值时，置Z标志位为 0 ）。当Z为1时，程序下滑，即按正常步骤执行＄ 305 号单元开始的第三条指令。而当Z为0时，非0条件满足，于是将PC中的值（＄305）加上相对位移量（＄02）置入PC中，变成了＄307。因此，做完 ＄303 的指令后，直接转＄307执行第四条指令。

这种条件转移指令与BASIC中的IF语句很类似。

上述程序的框图如图 11—2 所示。其运行结果是在＄2500号单元中存了一个数，它可能是＄2000号单元中的数(当它非0时)，或者是＄20（当＄2000号单元中的数为 0 时)。

现在再来看另外一个程序段：

```
0300-  CE  00  20        DEC  $2000
0303-  AD  00  20        LDA  $2000
0306-  D0  F8            BNE  $0300
0308-  A9  20            LDA  #$20
030A-  8D  00  25        STA  $2500
```

030D－　00　　BRK

这个程序中的第一条指令的意思是对＄2000号单元中的数作减1操作。其它指令同前面那个程序段无异。

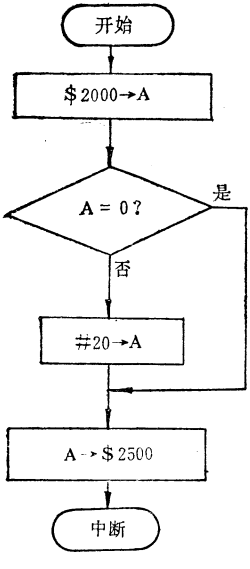


图11—2

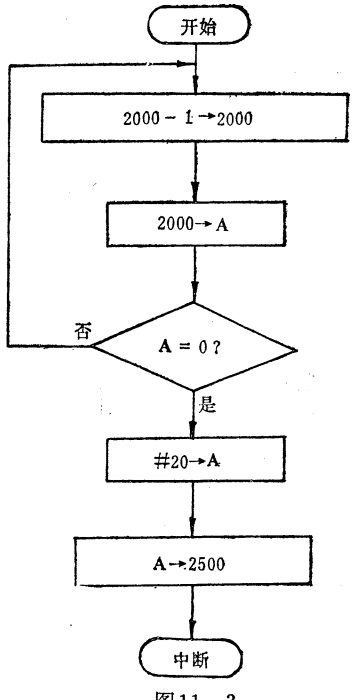


图11—3

这个程序中的条件转移指令（操作码在＄306号单元中)的相对位移量给出了一个负数（＄F8：最高位为1），这时如果转移条件满足，PC中的值(此时应为＄308）与F8相加实际上是使PC减去8（因为F8的补码是08)，因此＄308－＄08＝＄300。于是非0条件满足时转至＄300去。

这段程序的框图如图11—3所示。

在画机器语言程序框图时，鉴于机器语言中用的数都是十六

进制的，我们不再特地用$标明。数字前带井的都指立即数。需间址时用小括号括住间址部分；需变址时用逗号后跟变址器标志。

§4. 补充说明

一、无条件转移指令的执行细节

无条件转移指令的执行细节分别是：

JMP　(PC+1)→PC_L,(PC+2)→PC_H

JSR　PC+2↓,(PC+1)→PC_L,(PC+2)→PC_H

RTS　PC↑,PC+1→PC

RTI　P↑,PC↑

BRK　强迫中断,PC+2↓,P↓

1.关于JMP

JMP只有两个操作码，一为6C，用于间址方式，上节已介绍；另一为4C，为绝对寻址方式。现从下例看起：

```
2010-  A9  08          LDA  #$08
2012-  4C  17  20      JMP   $2017
2015-  A9  09          LDA  #$09
2017-  8D  01  30      STA  $3001
201A-  00
```

执行此程序时，应先置PC的数为$2010；

执行2010开始的指令时，PC中为$2012；

执行2012开始的指令时，由于PC为$2012，PC+1则为$2013；PC+2为$2014，于是依执行细节，将$2013中的数$17送$CP_L$，$2014中的数$20送$PC_H$。所以PC中的数为$2017。这是执行第二条指令的结果。

执行完第二条指令后，由PC控制，转去执行$2017开始的另一条指令。

这就是说，操作码4C之后的地址，指明程序转往那里继续运行。

这很类似于BASIC中的GOTO语句。

2.JSR和RTS

JSR只有绝对寻址方式，操作码为20。RTS只有隐含寻址方式，操作码为60。

请看下例：

```
53AB－20    AF    53      JSR  $53AF
53AE－00                  BRK
53AF－A9    AF            LDA  #$AF
53B1－85    3C            STA  $3C
53B3－60                  RTS
```

运行此程序，先置PC中为$53AB。

执行第一条指令时，操作码为$20，是JSR指令，依执行细节，将PC中的数加2成$53AD（PC中仍为$53AB）进栈（注意此为二字节数，即进栈两个数），使栈状态由图11—4(*a*)变为(*b*)，再将$53AC($53AB+1)中的数送PC_L，将$53AD($53AB+2)中的数送PC_H，使PC中的数变为$53AF。

执行完第一条指令后，依PC的控制，使程序转$53AF去运行一个子程序。

当执行到$53B3的指令时（RTS），依执行细节，将栈顶的$53AD退栈进入PC中，（栈状态变为图11—4(*c*)）再使PC中的加1，遂使PC中的数为$53AE。

执行完$53B3指令后，依PC的值，使程序执行$53AE的BRK。

可见JSR和RTS很象BASIC中的GOSUB和RETURN。$53AF为子程序的入口地址。

3.BRK和RTI

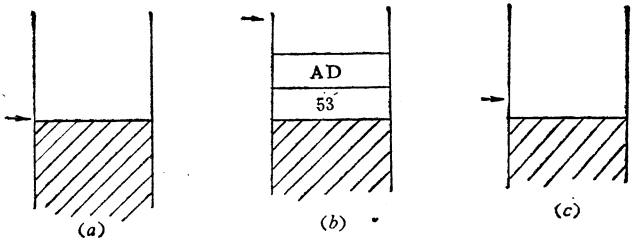


图11—4

请看下例：

```
3000 - A9  03          LDA  #$03
3002 - 00              BRK
3003 - 00              BRK
3004 - 85  35          STA  $35
  ⋮
```

程序运行时，先置PC为 $ 3000。

执行 $ 3000开始的指令时，PC变为 $ 3002。

执行 $ 3002开始的指令时，遇BRK，置B标志位为1。PC + 2 (即 $ 3004)进栈，使栈状态见380页图11—5—(*a*)变为(*b*)，P进栈，又使栈状态为 (*c*)。由于B 标志升起 (为1)，使程序转去执行中断处理程序。程序 遇到中断返回指令RTI 时，首先使P退栈，即恢复中断时各标志位状态，栈状态成 (*b*) 态。再使PC 出栈，这时PC中为 $ 3004，栈状态恢复 (*a*) 态。依PC的值，程序从 $ 3004处继续往下执行。

二、如何看指令表

指令表幅面较大，不易制版，也不好装订，这里把它拆成几个表，格式不尽一致，并不难看。读者也可以根据需要合成一个表。表中给出各条指令在不同寻址方式下的操作码、指令长度(字节数) 和执行周期，并且在最右一栏中给出了每条指令可以改变

表11--1

		相对			隐含			
助记符	功能	操作码	长度	周期	操作码	长度	周期	NV·BDIZC
BCC	C=0转	90	2	2				·····0
BCS	C=1转	B0	2	2				·····1
BNE	Z=0转	D0	2	2				····0··
BEQ	Z=1转	F0	2	2				····1··
BPL	N=0转	10	2	2				·0·····
BMI	N=1转	30	2	2				·1·····
BVC	V=0转	50	2	2				0·····
BVS	V=1转	70	2	2				1·····
BRK	中断				00	1	7	···1·1··
CLC	0→C				18	1	2	·····0
SEC	1→C				38	1	2	·····1
CLI	0→I				58	1	2	····0··
SEI	1→I				78	1	2	····1··
CLV	0→V				B8	1	2	·0·····
CLD	0→D				D8	1	2	····0···
SED	1→D				F8	1	2	····1···
TXA	X→A				8A	1	2	N····Z·

续表

		相对			隐含			
助记符	功能	操作码	长度	周期	操作码	长度	周期	N V · B D I Z C
TXS	X→S				9A	1	2	· · · · · · · ·
TAX	A→X				AA	1	2	N · · · · · Z ·
TSX	S→X				BA	1	2	N · · · · · Z ·
TYA	Y→A				98	1	2	N · · · · · Z ·
TAY	A→Y				A8	1	2	N · · · · · Z ·
INX	X+1→X				E8	1	2	N · · · · · Z ·
INY	Y+1→Y				C8	1	2	N · · · · · Z ·
DEX	X−1→X				CA	1	2	N · · · · · Z ·
DEY	Y−1→Y				88	1	2	N · · · · · Z ·
NOP	空操作				EA	1	2	· · · · · · · ·
PHA	A↓				48	1	3	· · · · · · · ·
PHP	P↓				08	1	3	· · · · · · · ·
PLA	A↑				68	1	4	N · · · · · Z ·
PLP	P↑				28	1	4	依栈中内容
RTI	中断返回				40	1	6	依栈中内容
RTS	返主				60	1	6	· · · · · · · ·

* 相对转移指令执行周期：不转移时按表中的数，跳转在本页内加1；跳转至另一页加2。

表11—2

助记符	功能	操作码 立即	零页	零页,X	绝对	绝对,X	绝对,Y	(间接,X)	(间接),Y	标志位变化 N V · B D I Z C
ORA	$A \vee M \to A$	09	05	15	0D	1D	19	01	11	N · · · · · Z ·
AND	$A \wedge M \to A$	29	25	35	2D	3D	39	21	31	N · · · · · Z ·
EOR	$A \forall M \to A$	49	45	55	4D	5D	59	41	51	N · · · · · Z ·
ADC	$A + M + C \to A, C$	69	65	75	6D	7D	79	61	71	N V · · · · Z C
LDA	$M \to A$	A9	A5	B5	AD	BD	B9	A1	B1	N · · · · · ZC
CMP	$A - M$	C9	C5	D5	CD	DD	D9	C1	D1	N · · · · · Z C
SBC	$A - M - \overline{C} \to A$	E9	E5	F5	ED	FD	F9	E1	F1	N V · · · · · Z
BIT	$A \wedge M, M_6 \to V, M_7 \to N$		24		2C					M_7 M_6 · · · · Z ·
LDY	$M \to Y$	A0	A4	B4	AC	BC				N · · · · · Z ·
CPY	$Y - M$	C0	C4		CC					N · · · · · Z C
CPX	$X - M$	E0	E4		EC					N · · · · · Z C
指令长度(字节数)		2	2	2	3	3	3	2	2	
执行周期		2	3	4	4	4*	4*	6	5*	

* 当跨出本页时，加1

表11—3

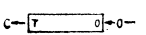
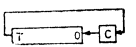
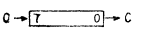
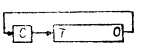
助记符	功能	操作码 零页	零页,X	绝对	绝对,X	绝对,Y	(间接,X)	(间接),Y	零页,Y	标志位 N V·B D I Z C
STA	A→M	85	95	8D	9D	99	81	91		······
STX	X→M	86		8E					96	······
STY	Y→M	84	94	8C						······
指令长度(字节数)		2	2	3	3	3	2	2	2	
执行周期		3	4	4	5	5	6	6	4	

表11—4

助记符	功能	操作码 立即	零页	零页,Y	绝对	绝对,Y	间接	标志位 N V·B D I Z C
LDX	M→X	A2	A6	B6	AE	BE		N····Z·
JMP	转移				4C		6C	·····
指令长度(字节数)		2	2	2	3	3	3	
执行周期		2	3	4	4△	4•	5	

• 跨页时加1；　△ 4C为3

表 11—5

助记符	功能	累加器	零页	零页,X	绝对	绝对,X	N V · B D I Z C
ASL		0A	06	16	0E	1E	N • • • • • Z C
ROL		2A	26	36	2E	3E	N • • • • • Z C
LSR		4A	46	56	4E	5E	0 • • • • • Z C
ROR		6A	66	76	6E	7E	N • • • • • Z C
DEC	M − 1→M		C6	D6	CE	DE	N • • • • • Z •
INC	M + 1→M		E6	F6	EE	FE	N • • • • • Z •
JSR	转子程序				20		• • • • • • • •
指令长度(字节数)		1	2	2	3	3	
执行周期		2	5	6	6	7	

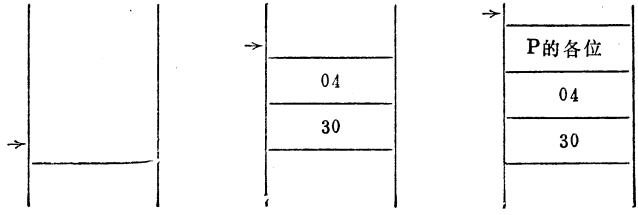


图11—5

哪些标志位（不变的标志位以圆点标出）。

现举一例：

ADC

这是一条加法指令，有8种寻址方式：

立即方式的操作码为$69，执行周期为2，指令长度为2字节。

绝对寻址方式的操作码为$65，执行周期为4，指令长为3字节。

……

绝对Y变址寻址方式，操作码为$79，执行周期为4，指令长为3字节。

§5. 几个简单的程序例

例1. 将$1000～$10FF中的数传送至$1100～$11FF中。

```
0300-  A2  00        LDX  #$00
0302-  BD  00  10     LDA  $1000,X
0305-  9D  00  11     STA  $1100,X
0308-  E8             INX
0309-  D0  F7         BNE  $0302
030B-  00             BRK
```

程序框图如下（图11—6）

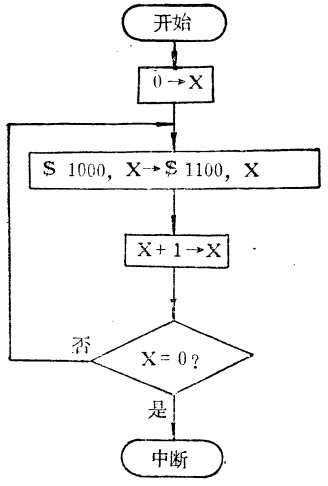


图11—6

例2. 等待键盘输入的程序

```
356-  2C  00  C0        BIT  $C000
359-  10  FB            BPL  $0356
35B-  AD  00  C0        LDA  $C000
35E-  2C  10  C0        BIT  $C010
361-  00                BRK
```

程序框图如下页所示(左右框图等效)：

这段程序类似于BASIC中的下列程序（但执行速度快得多）：

```
1000  P=PEEK (-16384)
1010  IF  P<128  THEN  1000
1020  POKE  -16368, 0
```

例3. 把第九章中的发音子程序用机器码和汇编码写出。如下：

```
302-  AD  30  C0              LDA  $C030
```

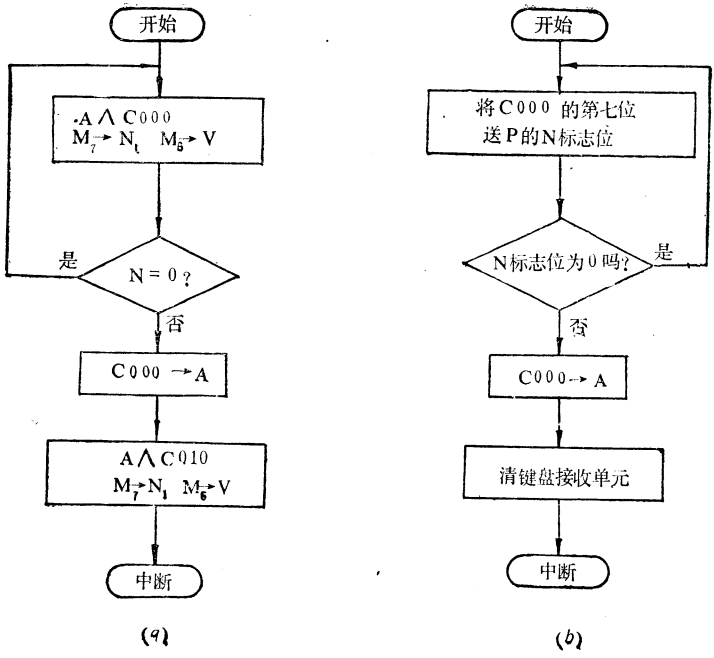


图11—7

```
305-  88          DEY
306-  D0  05      BNE  $030D
308-  CE  01  03  DEC  $0301
30B-  F0  09      BEQ  $0316
30D-  CA          DEX
30E-  D0  F5      BNE  $0305
310-  AE  00  03  LDX  $0300
313-  4C  02  03  JMP  $0302
316-  60          RTS
```

请读者对照第九章中的框图，再读一下此子程序。

§6. 监控命令简介

上述我们介绍的一些机器语言程序或汇编程序，只有在监控状态下，才能较快地输入与查看。

监控程序是常驻内存的，固化在ROM上，一开机它就存在。监控程序中有许多很有用的子程序，能为BASIC解释程序、操作系统甚至BASIC语言直接使用。

一、监控状态的进入与退出

从BASIC状态进入监控状态，用命令CALL－151。

在监控状态下，提示符变为一个星号“＊”。

退出监控状态命令有如下几个：

3D0G（在有DOS时方可使用）。

CTRL-C键；

RESET键等。

二、监控命令

在监控状态下，下列命令均可使用。

为了叙述方便，在具体描述每个命令时，我们在其左部画出一个＊号提示符。它是机器自动给出的，用以表明当前处于监控状态下。

另外，在每个命令结束后，总要敲入一次RETURN，作为命令的结束标志。我们用↙号来代表它。

在下边描述中，凡‹地址›，或‹数›，表明真正使用此命令时，在相应的位置上应敲地址号码或具体数。凡必须输入一个空格的地方，我们用“␣”代替。

在监控中，数和地址都用十六进制表示，一律不得再给$标志！

1.检查（察看）内存贮器的内容

(1) ＊〈地址〉↙　显示指定地址中的内容。

(2) ＊·〈地址〉↙　显示从当前地址到给定地址所有内存单元中的内容。

(3) ＊〈地址1〉·〈地址2〉↙　显示从地址 1 到地址 2 的所有单元中的内容。

(4) ＊〈地址〉↙　显示给定地址中的内容。

＊↙　显示后续地址到地址尾数为 7 或F 的内容（最多八个单元）。

＊↙　显示后续 8 个地址中的内容。

说明：机器显示内存中内容时，如满 8 个字节，每 8 个字节占一行。不满 8 个字节时，从左部排起列完为止。字节与字节中间有一个空格。

注意；(1)、(2)、……标记是命令序号，不是命令中的内容。勿误会。

2.修改内存单元中的数

(1) ＊〈地址〉↙　显示给定地址的内容。

＊：〈数〉↙　把给定的数写入该地址。

(2) ＊〈地址〉:〈数〉↙把给定的数写入给定的地址中。

(3) ＊〈地址〉:〈数1〉␣〈数2〉␣…␣〈数n〉↙ 把给出的n个数依次写入从给定地始开始的n个单元中。

(4) ＊:〈数1〉␣〈数2〉␣…␣〈数n〉↙ 把给定的n个数依次写入从当前地址开始的n个单元中。

说明：这里的数，从 $00到 $FF。前边一位（高位）为0时，可以省去。

3.将内存中一段单元的内容移到另一段单元中（成片搬家）

＊〈地址1〉<〈地址2〉·〈地址3〉M↙　把从地址 2 到地址3的内容移到从地址 1 开始的连续地址中。

说明：这样移动后，从地址 2 到地址 3 的内容并未改变。

4.比较两段内存单元中的内容

＊〈地址1〉＜〈地址2〉·〈地址3〉V↙　比较从地址2到地址3的内容是否和从地址1开始的单元的内容一致。一致时什么也不显示。不一致时，显示不同的内容和两片内存中的相应地址。

5.GO命令

＊〈地址〉G↙　从给定地址开始，执行机器语言程序（也就是置给定地址于PC中）。

6.LIST命令

(1) ＊〈地址〉L↙　把给定地址起的各条指令及其汇编显示出来（20行）。

(2) ＊L↙　从当前地址开始输出20行指令及汇编。

(3) ＊LL…L↙　如为n个L，从当前地址起输出n个24行指令及汇编。

7.检查和改变各寄存器的内容

＊CTRL-E↙　显示A、X、Y、P、S各寄存器中的内容。

＊:〈数1〉␣〈数2〉␣，…␣〈数n〉↙

n最大为5，将数1送A，数2送X，…

8.将内存单元的一段内容存入磁带

＊〈首地址〉·〈末地址〉W↙

把指定的一段内存单元中的内容存入磁带。

9.从磁带读一段内容到内存

＊〈首地址〉·〈末地址〉R↙

将磁带上一段内容读入内存的指定单元中。

10.其它命令

(1) 改变显示方式命令

①＊I↙　使屏幕显示字符为反方式（同BASIC中的INVERSE命令类似）。

②＊N↙　使屏幕显示字符为正常方式（同BASIC中 NORMAL命令类似）。

（2）开关打印机命令

设打印机与 1 号槽口接连。

①＊1CTRL-P　接通打印机。

②＊0CTRL-P　关闭打印机。

（3）强迫转移命令

＊CTRL-Y↙　强行使监控程序转到 $3F8去。

为使这条命令得以执行，$3F8 中应放入一条适当的转移指令，转移到你所需要的地方。

（4）十六进制加减运算

①＊〈数1〉+〈数2〉↙　按十六进制将二数求和并显示。

②＊〈数1〉-〈数2〉↙　求二数差并显示（均十六进制数）

（5）多重命令

比如：

＊〈地址1〉L〈地址2〉G↙　先将地址1开始的20条指令及汇编列出，再从地址 2 开始执行机器语言程序。

三、关于小汇编

一个用机器语言写出的程序，无疑可以用修改内存数值的办法将机器码一一敲入。但由于151个操作码很难记住，用机器语言写程序较困难。用汇编语言来写当然要容易得多。

汇编语言如何输入机器中呢？

苹果机带有一块能用于编辑和汇编的磁盘，它有一套操作方法和命令，比较麻烦，但功能较强。限于篇幅，不介绍它。

这里介绍一下小汇编。

在整数BASIC解释程序的后部，附有一个小汇编，比较简单实用。

只要你进入了整数 BASIC 系统（详见十三章），再进入监控

状态。这时出现＊提示符。接着打入F666G命令。如下：

＊F666G↙

!

!号是小汇编提示符。这时就可以开始输入汇编程序。

输入第一条指令时，一定要给出地址，然后跟着输入一个冒号。随后就可输入一条汇编指令。每输完一条，要敲一次RETURN键以示一条汇编指令结束，机器立即显示该条指令的机器码及汇编码。如果输入有错，将在出错处的下方显示一个∧号，并“嘟”一声，你可以重新输入正确内容。

输入后续指令时，一般可不再给出地址，在提示符“!”之后，敲入一个空格，再敲入汇编指令即可，其它过程同上。这样输入的一条条指令，被装入给定地址开始的一块内存中，并且不再具有汇编形式，而变成机器语言指令序列。

比如有下面三条指令：

```
300-  A9  10          LDA  #$10
302-  8D  00  10       STA  $1000
305-  60               RTS
```

用小汇编输入时过程如下：

```
! 300  :  LDA#10↙
```

机器立即显示

```
300-  A9  10        LDA  #$10
```

然后在“!”之后你可敲入第二条指令：

```
! ␣  STA  1000↙
```

这时机器显示：

```
302-  8D  00  10    STA  $1000
```

然后在“!”之后输入第三条指令：

```
! ␣  STR↙
```

屏幕上又显示：

```
305-  60          RTS
```

这样，你的三条指令，便以机器语言的形式，存放在内存$300开始的区域中。

注意：上述过程中的!是机器给出的，不是输入时敲入的。

退出小汇编的命令是：$FF69G。需退出时，在! 之后敲入：

!$FF69G↙

这里的$字符不可省略。

从小汇编退出后，又回到了监控状态。再从监控返回BASIC，可用：

*3D0G↙

上述命令F666G，$FF69G，3D0G，其中的G，就是前面提到的GO命令。也即把这些地址分别送入PC。

四、专门的控制字符与编辑字符

1.控制字符

控制字符是指CTRL-A～CTRL-Z，都是不可显示字符。监控为它们中的某些字符定义了一些功能。RESET也是被定义的特殊功能键。

(1) RESET键：在任何状态下，它立即中断任何命令或程序的执行（但不破坏程序及各变量的当前值），给出BASIC提示符，等待用户输入。

(2) CTRL-B字符：仅在监控状态下具有规定功能。敲入此字符再敲回车键，会使机器从监控状态返回进入监控前的BASIC状态，并破坏当前的BASIC程序。

(3) CTRL-C字符：在BASIC状态下，它会中止程序的运行。这样停下的程序可由CONT(在整数BASIC中为CON) 命令继续程序的运行。

在监控状态下，CTRL-C命令会在不破坏当前程序的情况下返回BASIC状态。

(4) CTRL-G字符：可驱动喇叭发出“嘟”声。

(5) CTRL-H字符：光标回退一格，相当于←键。

(6) CTRL-U字符：光标右移一格，相当于→键。

(7) CTRL-M字符：相当于RETURN键。

(8) CTRL-J字符：发出换行信号。

(9) CTRL-X字符：使刚敲入的字符行无效(作废)。

(10) CTRL-S字符：暂停输出。再敲任一键能使输出继续进行。

从(4)到(10)，可用于监控和BASIC的任一状态。

2.编辑字符

编辑字符可用于任何状态下。任何编辑字符都由ESC键导入。即先敲入ESC键，再敲相应字母键。

它编辑字符功能是：

ESC A：使光标右移一格。

ESC B：使光标左移一格。

ESC C：使光标下移一行。

ESC D：使光标上移一行。

ESC E：从光标处开始清除原文到本行尾。

ESC F：从光标处开始清除原文到屏幕页末。

ESC @：将光标置屏幕左上角，并清除整幅屏幕（相当于HOME命令)。

ESC I：使光标上移一行。

ESC J：使光标左移一格。

ESC K：使光标右移一格。

ESC M：使光标下移一行。

注意：后四个命令有一特点：敲入一次ESC后，可连续使用I, J, K, M 中的任何键使光标移动，只有敲了别的键，ESC 的作用才消失。

用后四个键可使光标在屏上随处移动，在移动过程中，凡经→键控制扫描过的字符(除非再被←键控制擦除)，包括从键盘上敲入的字符，都被当作是从键盘上输入的一样被接收。

因此，用这四个键控制修改一个含有错误的程序行很方便。首先将这个程序行显示于屏上，然后用这四个键将光标移至该行首，把需保留的字符用右移键一一扫过，需要删除的字符由空格键移过(删去)。需要插入字符时，用上述四编辑字符将光标移至被编辑行外，再一一敲入需插入的字符，插入完结后，仍可由四个编辑字符将光标移至适当位置，再由→键继续扫过所需字符，直到本行所需字符结束，敲入回车键。其实，屏上看到的所有字符中，只要有你所需要的字符序列，你都可以用上述四个编辑字符将光标移至相应位置，由→键录入你正在编辑的程序行中。

五、操作实例

1.把唱歌用的子程序，用修改内存中数据的办法输入机器中。步骤如下：

] CALL-151↙

*302：AD␣30␣C0␣88␣D0␣5␣CE␣1␣3␣F0␣9␣CA␣D0␣F5␣AE␣0␣3␣4C␣2␣3␣60↙

再用LIST命令列出：

*302L↙	(以下为机器输出的)
302- AD 30 C0	LDA $C030
305- 88	DEY
306- D0 05	BNE $030D
308- CE 01 03	DEC $0301
30B- F0 09	BEQ $0316
30D- CA	DEX
30E- D0 F5	BNE $0305
310- AE 00 03	LDX $0300
313- 4C 02 03	JMP $0302

```
316-  60                          RTS
:     (补足20行)
```

如果将$98和$46分别送入内存的$300和$301（即十进制的768和769），可用命令：

```
*300: 98␣46↙
```

这时运行这段程序可发出半拍“咪”（3)的声音。这可用下一命令实现：

```
·302G↙
```

若是把上述程序作如下扩充：

```
317-  A2  00                    LDX  #$00
319-  BD  40 03                 LDA  $0340,X
31C   20  30 03                 JSR  $0330
31F-  E8                        INX
320-  E0  08                    CPX  #$08
322-  D0  F5                    BNE  $0319
324-  CA                        DEX
325-  30  F0                    BMI  $0317
327-  BD  40 03                 LDA  $0340,X
32A-  20  30 03                 JSR  $0330
32D-  4C  24 03                 JMP  $0324
330-  86  2A                    STX  $2A
332-  8D  00 03                 STA  $0300
335-  A9  46                    LDA  #$46
337-  8D  01 03                 STA  #$0301
33A-  20  02 03                 JSR  $0302
33D-  A6  2A                    LDX  $2A
33F-  60
340-  C0  AC  98  90  80  72  66  60
```

程序中$33B～$342分别存放1，2，3，4，5，6，7，i七个音的音阶控制值，主程序在 $317～$33A中，子程序在$302～

$316中。

主程序框图如图11—8所示：

这个程序可重复演奏：

1 2 3 4 5 6 7 i

i 7 6 5 4 3 2 1

用修改内存单元内容的方法输入时，最好分段输入。

运行这个程序可用命令：

*317 G↙

2.把扩展后的程序用小汇编输入。

操作如下：

] INT↙

> CALL-151↙

*F666G↙

! 302: LDA ␣C0␣ 30↙

! ␣ DEY↙

! ␣ BNE ␣30D↙

! ␣ DEC ␣301↙

! ␣ BEQ ␣316↙

! ␣ DEX↙

! ␣ BNE ␣305↙

! ␣ LDX ␣300↙

! ␣ JMP ␣302↙

! RTS↙

! LDX #0↙

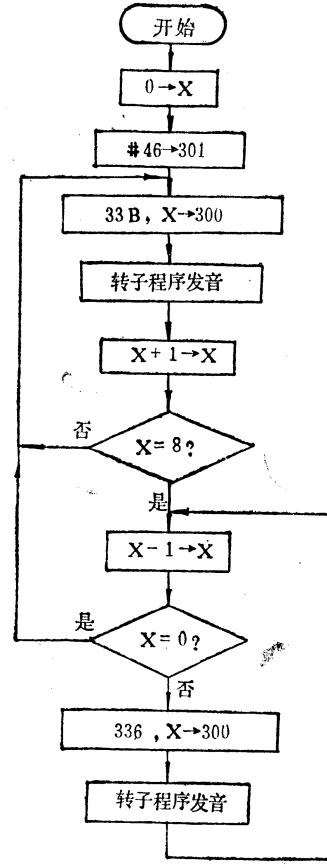


图11—8

```
! LDA 340, x↙
! JSR 330↙
! 31F: INX↙
! CPX *8↙
! BNE 319↙
! DEX↙
! BMI 317↙
! LDA 340, X↙
! JSR 330↙
! JMP 324↙
! STX 2A↙
! STA 300↙LDA *46↙
! LDA *46↙
! STA 301↙
! JSR 302↙
! LDX 2A↙
! RTS↙
! $FF69G↙
*340: C0  AC  98  90  80  72  66  60↙
```

这时用GO命令可运行它。也可用LIST命列示它。

习　题

(一)写一个机器语言程序，使内存$2000至$3FFF 全部变为0。要求程序和使用的数据在$300～3CF范围内。

(二)现有下列汇编程序，初始地址为$3000，请把它改写成机器语言，并说明它的功能是什么：

```
SEC
PHA
SBC  *$1
BNE  $302
```

```
PLA
SBC  #$1
BNE  $300
RTS
```

(三)有下列程序，请写出其汇编程序。并说明在监控状态下，敲入命令F273G和命令F277G时，分别出现什么现象。

```
F273- A9 FF
F275- D0 02
F277- A9 3F
F279- A2 00
F27B- 85 32
F27D- 86 F3
F27F  60
```

(四) 有下列一段子程序，请画出它的框图来，并指明这段程序的功能是什么。

```
300  A4 EF      LDX  $EF
302  A9 00      LDA  #$00
304  9D 00 02   STA  $0200, X
307  8A         TXA
308  F0 0B      BEQ  $0313
30A  BD FF 01   LDA  $01FF, X
30B  29 7F      AND  #$7F
30D- 9D FF 01   STA  $01FF, X
310- CA         DEX
311- D0 F5      BNE  $030A
313- 60
```

(五)已知$300，$301中放一被加数（低位在前、高位在后），$302和$303中放一加数(低位在前、高位在后)。写一个程序，求二数之和，将和数置$304和$305中(低位在前、高位在后)。

(六)有人希望把$2000号开始的$1000个数依次向后移动$100个单元

(即$2000→$2100，$2001→$2101，…，$2FFF→$30FF)。他采用了两种方法：

1.运行下列程序

```
300- A0 00      LDY #$00
302- 84 2A      STY $2A
304- 84 28      STY $28
306- A9 20      LDA #$20
308- 85 29      STA $29
30A- A9 21      LDA #$21
30C- 85 2B      STA $2B
30E- B1 28      LDA ($28),Y
310- 91 2A      STA ($2A),Y
312- C8         INY
313- D0 F9      BNE $030E
315- F6 29      INC $29
317- E6 2B      INC $2B
319- A5 29      LDA $29
31B- C9 30      CMP #$30
31D- D0 EF      BNE $030E
31F- 00         BRK
```

2.用下列命令：

```
] CALL-151
*2100<2000.2FFFM↙
*3D0G↙
]
```

请指出哪种方法是正确的，哪种方法是错误的，并改正其错误。

(七)有下列程序，请给出当A为0、1、2、3、4时，$28和$29号单元各应是什么数。

```
FBC1- 48        PHA
FBC2- 4A        LSR
```

```
FBC3- 29 03     AND #$03
FBC5- 09 04     ORA #$04
FBC7- 85 29     STA $29
FBC9- 68        PLA
FBCA- 29 18     AND #$18
FBCC- 90 02     BCC $FBD0
FBCE- 69 7F     ADC #$7F
FBD0- 85 28     STA $28
FBD2- 0A        ASL
FBD3- 0A        ASL
FBD4- 05 28     ORA $28
FBD6- 85 28     STA $28
FBD8- 60        RTS
```

(八)上机熟习监控命令的使用。

(九)上机练习编辑字符使用方法。

第十二章　深入的讨论

这一章打算从系统软件的角度给出一些必要的说明。目的是希望读者读完这一章之后，能更好地使用苹果机。

§1. DOS 自举与内存分配

把磁盘操作系统DOS3.3从磁盘装入内存中，称为DOS自举。

DOS 自举有两种方法。一种是开机时将装有 DOS的磁盘(系统主盘或用INIT命令格式化过的磁盘都装有DOS) 插入一号盘机内，打开主机电源后，磁盘机被启动。等到盘机上的小红灯熄灭，DOS就装入内存中了。第二种方法是使用IN#或 PR#命令 (#后面敲入磁盘控制器所在槽口号)，同样可完成自举动作。

自举后的 DOS，总是装在RAM 的最高位置。对于 48KRAM 的机器来说，它装在内存的 $9600～$BFFF(即38400～49151)，占用 $2A00个内存单元。它除了DOS自身的信息外，也包括了供文件使用的三个缓冲区。DOS在内存中所占的最小地址号码，称为 HIMEM值，字符串缓冲区就紧挨着它，是用户可以使用的最高地址。

显然，RAM大小不同时，HIMEM的值是不一样的。下表列出了几种情况：

系统容量	最高RAM地址		DOS自举的HIMEM	
RAM容量	十进制	十六进制	十进制	十六进制
16K	16383	$ 3FFF	6532	$1600
20K	20479	$ 4FFF	9728	$2600

24K	24575	$ 5FFF	13824	$3600
32K	32767	$ 7FFF	22016	$5600
36K	36863	$ 8FFF	26112	$6600
48K	49151	$ BFFF	38400	$9600

可见，当系统小于16K时，也就无法使用DOS了。

在不装入DOS的情况下，BASIC 同样可以工作，只是无法和磁盘打交道就是了。这时，HIMEM就是最高的RAM地址加1。对于48K的机器而言，就是 $ C000。

在内存中，除了DOS之外，还有监控程序、解释程序，用户源程序、屏幕区等，系统是这样分别给以分配内存空间的：

$ 0000～$ 00FF：即0页，为系统所用，主要用来设置一些指针和一些工作单元。

$ 0100～$ 01FF：即1页，为系统堆栈。

$ 0200～$ 02FF：即2页，为键盘输入缓冲区。由键盘输入的信息，先存放在这里，当接收到RETURN时再行处理。

$ 0300～$ 03FF：即3页，这一页是留给用户的。用来写一些机器语言子程序或存放一些数据。这个区，非经用户指明，系统是不会访问它们的，所以较安全。但由于需要，系统也占用了其中后 $ 30个单元。因此用户只可用 $ 0300～$ 03CF。

$ 0400～$ 07FF：为文本和低清晰度图形第一页存贮区。

$ 0800～$ 0BFF：为文本和低清晰度图形第二页存贮区（这一区的使用方法，后文会述及）。

$ 0800～HIMEM-1：用户区，其中：

$ 2000～$ 3FFF：高清晰度图形第一页存贮区；

$ 4000～$ 5FFF：高清晰度图形第二页存贮区；

$ C000～$ CFFF：I/O区。留外设接口用；

$ D000～$ F7FF：BASIC解释程序；

$ F800～$ FFFF：监控程序。

内存分配情况，示意于图12—1中。

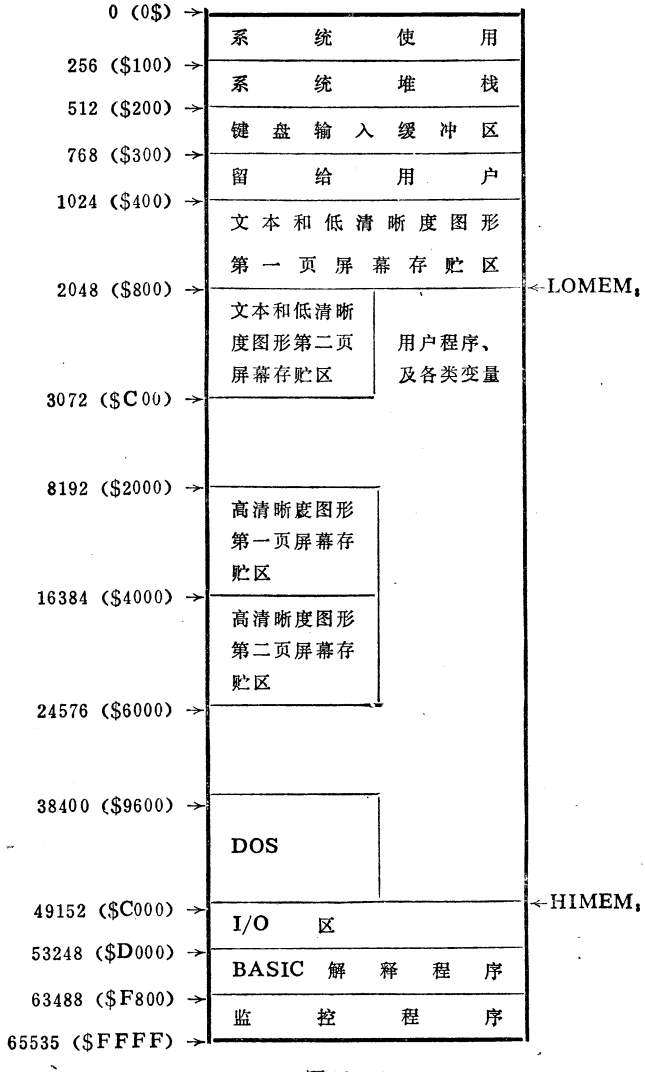


图12—1

在此我们约定，以后的叙述中，如不特别指明，我们总是按

DOS已经自举的情况下(即按上述分配)，来加以处理的。

现在，我们准备稍微全面、而又概括地介绍一下，从打开机器电源到敲入一个程序以至把它运行结束的全过程。它的粗框图在图12—2中给出。具体实现方法，在以下各节中作出简要说明。

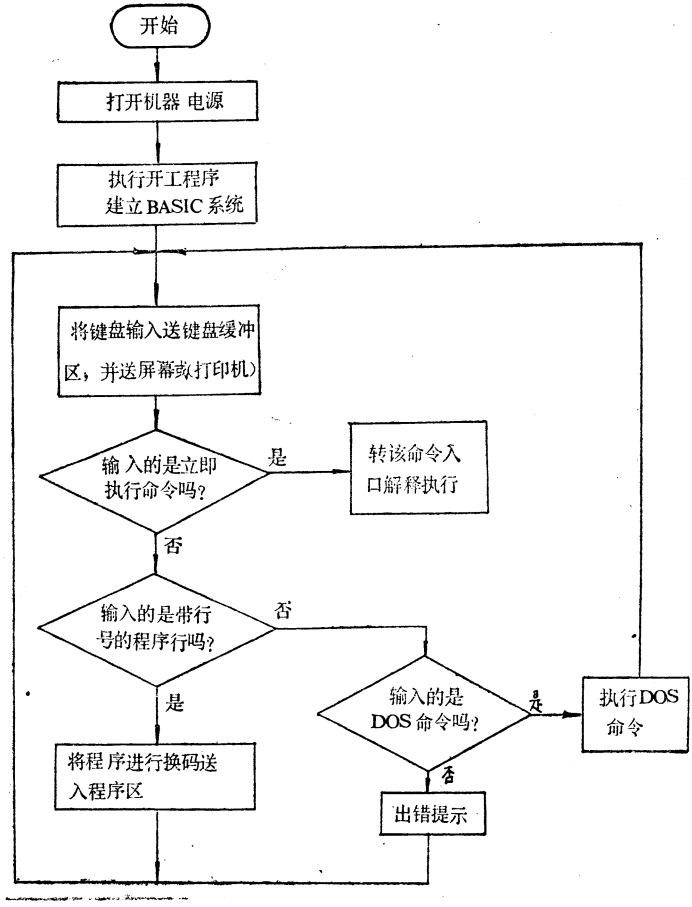


图12—2

§2. 系统的建立

一、RESET过程

我们知道'计算机之所以能一步一步工作起来，是由于 PC 中不断给出一些地址，也就是说，PC中各位的高低电平不同，可以将某个内存单元中的数码(这就是操作码)送入控制电路，不同的数码可控制产生不同的动作(这就是执行指令的功能)，同时还会送给 PC 一个新的数字（下条指令操作码的地址)，这些动作完结之后(即执行完一条指令)，又依PC中的状态找到新的地址去执行新的动作。这是计算机不休止地工作的基本方法。

问题是开机以后PC中的第一个数是怎样建立的。

原来6502微处理器上有一个被称为$\overline{RES}$的复位信号，它是当电源接通时产生的。这一复位信号自动将 $FFFC中的内容 送 入 PC_L，将 $FFFD 中的内容送入 PC_H。 $FFFC 和 $FFFD 在监控中，是固化的，一接通电源就可读出来。这两个单元中的数分别是 $62和 $FA，于是使PC最先得到的数是 $FA62。这样机器就从 $FA62开始执行程序。 $FA62 也在监控中，所以开机时这段程序是存在的。

这段程序称为RESET程序，也可称为开工程序。

RES 复位信号也可由RESET键产生。所以任何时刻(只要机器接通了电源) 敲击RESET键，都能进入开工程序。

RESET 程序的流程粗框图，示于图12—3中。有兴趣的读者可以打印出来对照阅读。打印的方法是：进入监控，启动打印机(也可先启动打印机、后进监控)， 给出地址 FA62，跟上一个或几个 L，回车即可。这段程序之结尾在 $FAD4。其中用到几个子程序，可先略过不读，(因为它又可能引出一系列子程序来，使问题复杂化)。

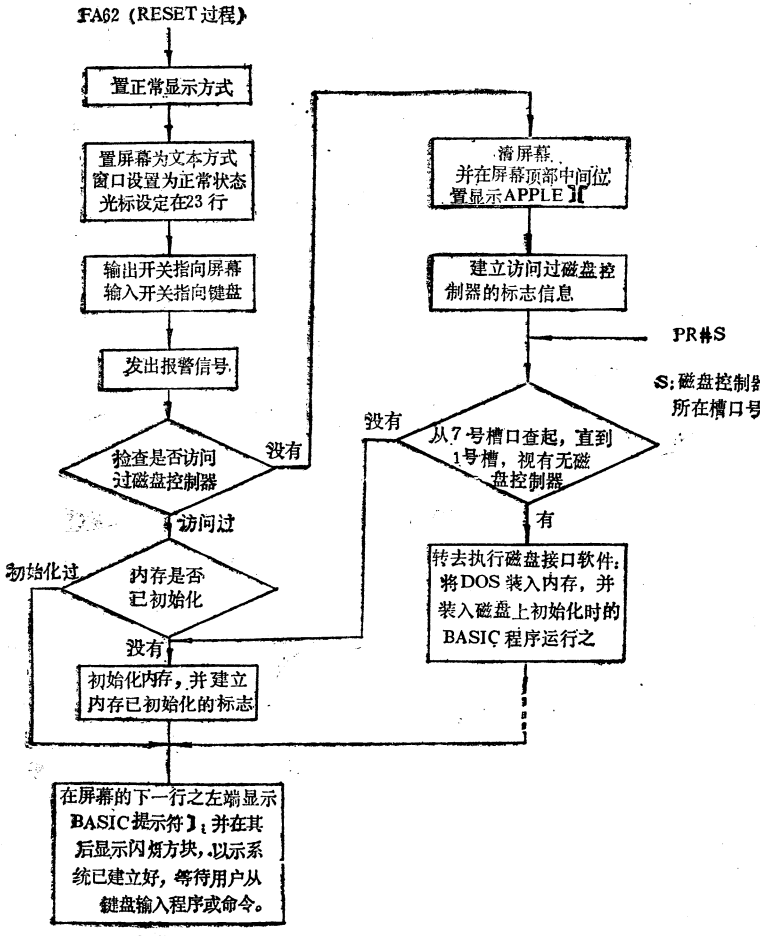


图12—3

从框图中不难看出以下几点：

1.如果系统未有配置磁盘机，那么从7号至1号槽口上将找不

到磁盘控制器，只能建立无DOS的BASIC系统。

2.如果系统配置有磁盘机，RESET 过程将会把 DOS 装入内存，并运行磁盘上的第一个BASIC程序。此后，内存中的程序就可以和磁盘交换信息了。

3.如果你虽有磁盘机，但却没有把磁盘插入盘机内，磁盘机会没完没了地转下去。这时敲入 RESET 键可以使磁盘机停止工作，同时也建立了一个无DOS的BASIC状态。

4.BASIC状态一经建立(不论有无DOS)，任何一次RESET将会使机器处于等待用户输入状态。但它不再访问磁盘控制器，也不再进行内存的初始工作。

二、内存的初始化

内存的初始化，大体上有以下几件工作：

1.调一个扫描用的子程序到0页的 $ B1～$ C8。这个子程序经常被用到。比如键盘输入结束后，输入的内容都在键盘缓冲区中。解释程序要一个个扫描其中的字符，检查是些什么信息：可能是行号，可能是变量，可能是保留字，……通常要扫过几个字符，甚至扫描几遍，才能弄清一个意思。对于程序区的内容也需作这种扫描工作。由于被检查的字符所在的地址码，在扫描过程中总是以不断加1的方式出现的，也就是说地址是改变的，从而使这个扫描程序中某些字节的内容要不断变化。这就使得它无法出现在BASIC解释程序中(因为那里是固化的)。为此，初始化时，把它转入0页，这样BASIC解释程序就能使用它了。

2.检测内存的大小

检测内存的目的，是确定用户可用的最高地址，即 HIMEM 值。

检测内存的方法很简单。它从 $ 0800号单元开始检测，先从中读出内容，全字节取反(0变1，1变0)后，再存入该单元，然后读出比较。如果该单元原来的数和现在的数不同，说明它是RAM

的一部分，再如此检测下一页(即从＄900开始的＄100个字节)，每页只检测页首一个字节。这样一页一页检测下去，直到发现某页首地址原来的内容和取反后存入的内容一致时为止。这时这一页实际上是不存在的或是固化了的。HIMEM 就取这页的页首地址号码。

DOS自举是不进行初始化的。它的HIMEM值，是对存有DOS的那块盘格式化时写入的（比这个值要低，因为除DOS外还留出了三个文件缓冲区的位置)。

3.建立系统指针

HIMEM 本身就是一个指针，它的数值记录在＄73和＄74号单元中，指明用户可用的最大内存地址（实际上比它小1)。同时把这个指针的值送入＄6F、＄70号单元中，它指明字符串数据存贮区的首地址。当有一个字符进入字符串存贮区时，＄6F、＄70中的数减1，把那个字符存入＄6F、＄70指向的单元。

第三个指针是 LOMEM。它指明用户区的首地址。这个指针的值存放在＄67和＄68中。为了以后运行检查时的方便，实际上它是用户区首地址加1，通常取＄801。

第四个指针是变量表首指针，初始化时取 LOMEM＋2，即803，它被存放在＄69、＄6A中。

变量表接在程序区之后，所以通常变量表首指针的值也被当作程序尾指针。程序尾指针放在＄AF 和＄B0 中，初始化时其值为＄803，这是第五个指针。

第六个指针为数组表首指针，它接在变量表之后。其值存放在＄6B、＄6C中，初始化时也为＄803。

第七个指针为数组表尾指针。其值存放在＄6D、＄6E 中，初始化时为＄803。

这七个指针归纳如表12—1所示：

内存初始化程序，从＄F128～＄F1D4。

表12—1

指针名称	指针地址	初始化值
HIMEM（最高可用地址）	73	00
	74	96
字符串数据存贮区首址	6F	00
	70	96
LOMEM（用户区首址）	67	01
	68	08
变量表首址	69	03
	6A	08
程序尾地址	AF	03
	B0	08
数组表首地址	6B	03
	6C	08
数组表尾地址	6D	03
	6E	08

§3. 程序(或命令)的输入

一个程序行(或命令)的输入，总是以敲入某一字符开始，而以RETURN结束。RETURN以前敲入的字符，依次存入键盘缓冲区。由于敲入的字符可以是多种多样的，处理也较复杂。

每接收一个字符，总要在屏上显示出来，所以必须有显示字符的子程序与它配合。

接收程序行(或命令)的子程序，入口为 $FD6A，出口由显示字符子程序退出。它在遇到输入为RETURN后，由显示字符子

程序产生一个回车换行动作，再退出来。然后转键缓区处理程序。

一、接收一个程序行(或命令)进键盘缓冲区的过程

这是一段子程序。它的框图如图12—4所示。

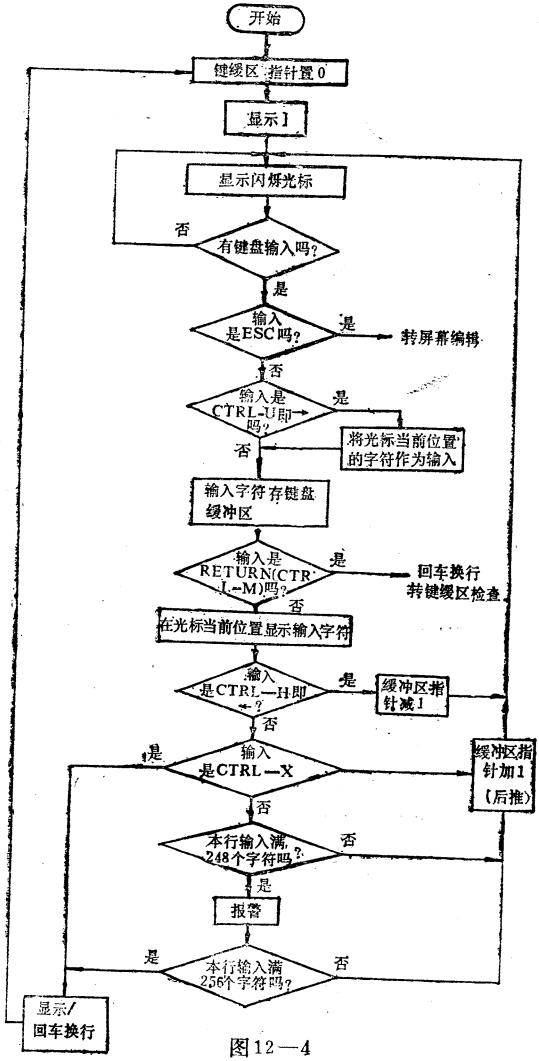


图12—4

程序中以X变址值代表当前输入字符在键缓区的序号。开始时置0。当敲入CTRL-X或键缓区装满(虽经报警仍继续输入时出现的)256个符时，会给出字符"＼"提示，然后再使X置0。这就是说，在这种情况下，刚才的输入作废了。

正常输入时，总是每接收一个字符，立即依X指示位置装入键缓区，并显示出来。然后X的值增1（使下次输入能进入缓冲区的下一个位置），再给出闪烁光标等待新的输入。当遇左移键时X作减1操作；遇右移键时把屏上字符当作键入，X作加1操作（输出位置右移）。

遇ESC输入时，处理稍复杂些，无非是将键入字符和屏上被扫描字符收入键缓区而已。这里不再作详细介绍。

只要发现敲入RETURN时，便结束此子程序。

二、显示字符子程序

把需要显示的字符送入累加器A中，转入此子程序即可在屏上显示出来。

子程序的入口为$FDF0，其中主要用到一个关键的子程序，被称为COUT1。其入口为$FB78，退出口有几个，地址号最大的为$FCA7。这个子程序如图12—5所示。

这个子程序是很重要的。因为任何输入输出都要调用它，包括回车、换行，输出任何一个提示符等。

程序中要查询输出是否为特殊字符，因为特殊字符要产生特殊动作。如CTRL-H要左移输出位置，CTRL-J要使光标上推一个文本行位置，CTRL-G要发"嘟"声，RETURN要回车换行等等，程序都作了相应的处理。对于输出CTRL-S，会使输出停下来。这是以程序走一个有条件的循环来实现的，它仍然在执行指令，并非静止不动。当输入一个别的字符后，它就不在那里循环了。当敲入CTRL-C时，键盘接收单元不予清除，待程序转入有关段落后再依据它的存在而中断程序的运行。还需交待的是，只有遇到

RETURN才检查CTRL-S 和 CTRL-C的输入，是为了输出的完整性。换言之，只有输出RETURN时才处理这两个字符的功能。

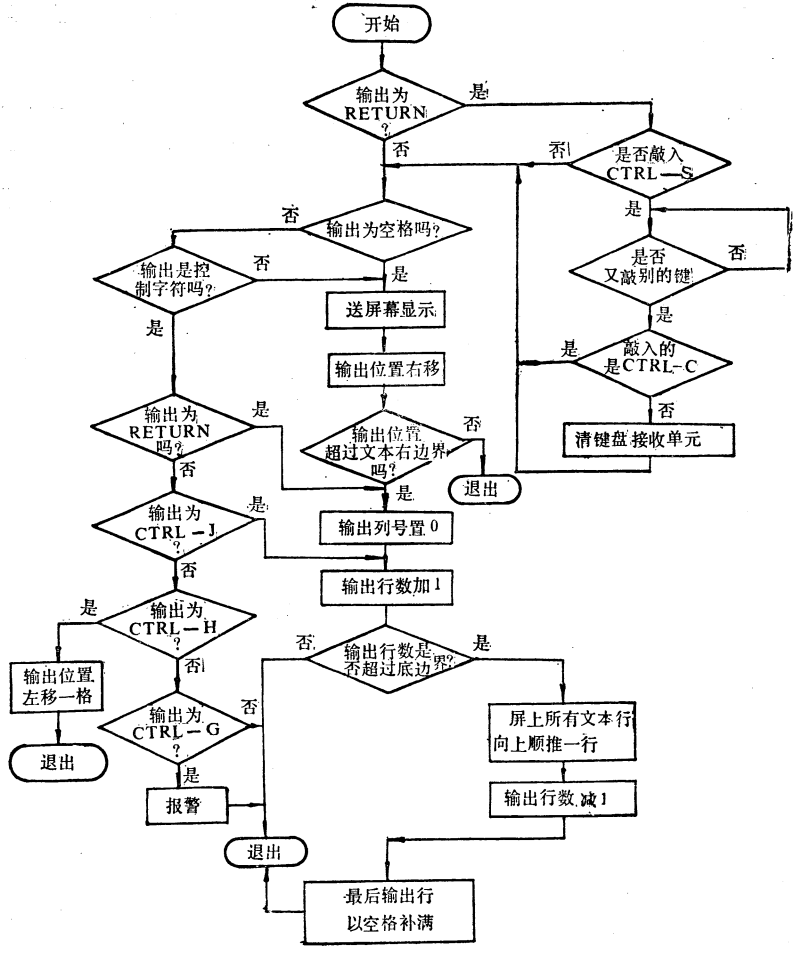


图12—5

三、键盘缓冲区处理

键盘缓冲区的处理，大体上有三项任务：

1.检查行长与换码

一个程序最多不能超过249个字符。当一行输入结束后，发现该行长度超过249个字符时，其后的字符略去不管。

一个程序行（或命令）送入缓冲区时，是存放着各字符的键盘输入 ASCII码。这时要作换码处理。换码方法很简单：把各字符的最高位置0。换码的目的是为以后使用的方便，这是系统的约定。

2.后续处理判定

后续处理要依据键盘缓冲区的内容来判定：

首先，判断刚刚的输入是否有字符，因为有时你可能什么字符也未敲入，只是敲了一个回车符(RETURN)，这时，控制程序作回车换行处理，并输出]提示符，重新等待输入。如果有输入字符，做下一步。

第二步，检查键缓区首字符是否为数字。如果是数字，则认为刚才输入的是一个延迟执行的程序行，控制程序去执行这样一种处理：将缓冲区的程序行送入程序区，然后再等待键盘输入。如果键缓区首字符非数字，则认为是一个立即执行命令，转相应命令入口。

无论在那种情况下，都还要完成另外一种换码工作，这就是把行号的十进制数换成二进制码，并且把键缓区的保留字换成相应的内部代码，这是为后续处理作准备的。

3.将键缓区的信息送程序区

程序行在程序区存放，是按行号大小顺序排列的。当一个键盘缓冲区的程序行移入程序区时，要拿缓冲区的程序行号与程序区的一条条程序的行号作比较，以便把它插入程序区的适当位置。当键缓区程序行号大于当前程序区所有行号时，键缓区的程序行应当排在程序区尾。当键缓区程序行号介于程序区两个行号之间时，键缓区程序行就插入程序区这两个程序行之间。这时大于新入行

号的程序行，都要向后推移一段距离。在上两种情况下，程序尾指针、变量表头指针、数组表头指针、数组表尾指针都要增大一个相应的数字。当键缓区程序行号同于程序区中某一程序行号时，则以键入新程序行代替旧程序行（如果新入行号后，无语句，则删去这一行）。这时，只要新入程序行与被取代的程序行长度不同，其后的程序区部分及各个指针都要作适当的变化。

以上这些工作，说起来容易，程序却是很烦杂的。我们不打算对这些处理程序作更多细节上的介绍，读者明确上面的描述也就够了。

四、关于 ASCII码的进一步说明

我们在这里讲述这个小题目，一方面是只有这时才能说清楚，同时，也为了使读者以后读、写程序方便些。特别是在程序区，ASCII 码是大量用到的。

本书附录1给出了两个ASCII 码表。我们先看第2个表：显示ASCII码表。并事先声明，我们按十六进制编码来描述。

最左边四列，从字符"@"到字符"？"，编码＄00到＄3F（某一字符的编码，为其所在列上方的编码与所在行左方的编码之和），它们出现在文本显示区时，屏上呈反方式(白底黑字)。5列到8列，从字符"@"到字符"？"，编码为＄40 到＄7F，它们出现在文本显示区内时，屏上呈闪烁状态。11列到14列，从空格符(表上就是一个空格）到字符"－"，编码为＄A0到＄DF，它们出现在文本显示区时，屏上为正常方式下的字符（黑底白字）。第9列和第10列的字符，编码为＄80到＄9F，为控制字符，在屏上不显示。最后两列编码为＄E0到＄FF，这两列在屏上显示的字符，与所用的机器的字符发生器有关。有一种字符发生器不产生小写字，它们显示的就是表中最后两列所示字符。有一种字符发生器可以产生小写英文字母，那么，这些编码依次对应：

＼*a b c d e f g h i j k l m n o p q r s t u v w x y z*

{ | } ~

所以在该表的上方标有“小写”二字。

键盘输入时，使用正常显示方式下的编码(输入控制字符时，用控制字字符编码)。因此，我们通查询键盘输入专用单元(－16384)时，就是与这一部分ASCII 进行比较的。而从键盘进入键盘接收缓冲区经过处理后，这些ASCII码的最高位变成了0。于是，CTRL－D由＄84变成＄04，RETURN由＄8D变成＄0D，空格由＄A0变成＄20，字符 A由＄C1 变成＄41，数字0由＄B0变成＄30，……在BASIC程序中使用ASC函数求得的ASCII码或通过 CHR＄函数将一个ASCII码转为相应的字符，都指的是最高位为0的ASCII 码。程序区中字符的 ASCII码，也是这类编码。这类编码正是附录1中表1的机内 ASCII码表所列出的编码。

五、程序区结构

程序在程序区的存放格式，是一种链式结构。换言之，找到一行程序，就可以知道下一行程序在什么地方。这种结构使查找行号的工作大大加快。因为这时不需要扫描程序中的语句。这对于插入程序行、删除程序行以及执行GOTO、GOSUB、LIST等等语句，是非常方便的。

程序区是以一个程序行作为基本单位存放的。这个基本单位开始的两个字节，存放一个链指针，指明下一个程序行从内存中的那个单元开始（也就是下一个链指针的首地址在内存中的地址单元号码)。基本单位的随后两个字节是本程序行的行号。接下去就是程序行中的内容，除了保留字用内部代码外，其余均是各字符的ASCII码。一个程序结束时敲入的 RETURN，在此用0代替。

任何一个程序行从键缓区转入程序区之后，程序区总存在一个末程序行，它是没有后续程序行的。所以通常把它的链指针指向的地址及其后续地址（本应放下一个程序行的链指针的两个单元)，填入0，作为程序结束标志。

程序区的这种链式结构，示意于图12—6中。

现在假定有如下一个程序：

```
10 DIM A (15, 12)
20 DIM B (20)
30 A = 10
40 B = A + B
50 A$ = "ABCDE"
60 INPUT B$
70 INPUT B$
90 C = A + B
100 INPUT B$
110 A = 5 + C
120 B$ = A$ + B$
130 A% = 5
140 FOR I = 1 TO 10
150 B(I) = 1
155 A(I,0) = I * I
160 NEXT
170 PRINT "BYE"
```

这一段程序运行情况暂且不管，我们目前想说明的是，它在内存程序区是如何存放的。

用监控命令，可以看到程序区内容如下：

```
800-  00  0F  08  0A  00  86  41  28
808-  31  35  2C  31  32  29  00  1A
810-  08  14  00  86  42  28  32  30
818-  29  00  23  08  1E  00  41  D0
820-  31  30  00  2D  08  28  00  42
828-  D0  41  C8  42  00  3C  08  32
830-  00  41  24  D0  22  41  42  43
838-  44  45  22  00  44  08  3C  00
```

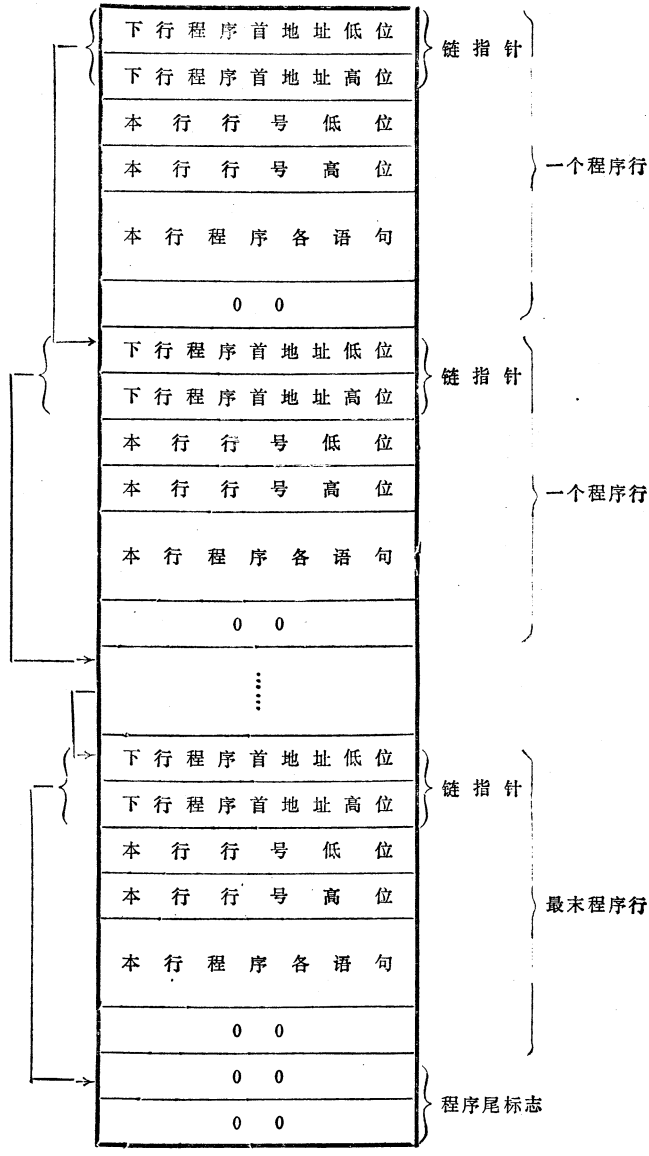

下行程序首地址低位
下行程序首地址高位
本行行号低位
本行行号高位
本行程序各语句
0 0
链指针
一个程序行
下行程序首地址低位
下行程序首地址高位
本行行号低位
本行行号高位
本行程序各语句
0 0
链指针
一个程序行
下行程序首地址低位
下行程序首地址高位
本行行号低位
本行行号高位
本行程序各语句
0 0
0 0
0 0
链指针
最末程序行
程序尾标志

图12—6

```
840-  84  42  24  00  4C  08  46  00
848-  84  42  24  00  56  08  5A  00
850-  43  D0  41  C8  42  00  5E  08
858-  64  00  84  42  24  00  68  08
860-  6E  00  41  D0  35  C8  43  00
868-  75  08  78  00  42  24  D0  41
870-  24  C8  42  24  00  7E  08  82
878-  00  41  25  D0  35  00  8A  08
880-  8C  00  81  49  D0  31  C1  31
888-  30  00  95  08  96  00  42  28
890-  49  29  D0  49  00  A4  08  9B
898-  00  41  28  49  2C  30  29  D0
8A0-  49  CA  49  00  AA  08  A0  00
8A8-  82  00  B5  08  AA  00  BA  22
8B0   42  59  45  22  00  00  00
```

这时如果你查看所有指针，可得到：

```
67-  01
68-  08 B8 08 B8 08 B8 08
6F-  00
70-  96 00 96 00 96
AF-  B8
B0-  08
```

可见几个指针的值改变了：变量表首址（在$69，$6A中）、数组表首址（在$6B、$6C中）、数组表尾地址（在$6D、$6E中）和程序尾指针（在$AF、$B0），都指向了$8B8。

程序区存放的第一个程序行在$801～$80E中。$801和$802放的是链指针，它指明下一个程序行从内存地址$80F号单元开始存放（同样$80F和$810又是一个链指针，指明它的下一个程序从内存$81A号单元放起。余类推）。$803和$804放的是该行程序的行号，在此为$000A，正是十进制的10。$805的$86

是保留字DIM的内部代码。$806的$41，是“A”的ASCII码。$807的$28和$80D的$29，分别是“(”和“)”的ASCII码。$808的$31和$809的$35以及$80B的$31和$80C的$32分别是数字1、5、1、2的ASCII码。$80A的$2C是逗号的ASCII码。可见从$803～$80E的内容，正是原来的第10程序行。

以下的程序段，读者不妨再读几行。

关于字符的ASCII码，可参阅附录1。各保留字的内部代码，见附录2。

§4. 程序(或命令)的运行

在对键盘输入缓冲区作检查时，只要发现缓冲区的首字符非数字，便按立即执行的命令进行处理。

一、执行过程概述

首先要把保留字换为内部代码。它是一个逐字扫描过程，只要连续几个字符能结合成一个保留字，便将这几个字符换成相应的一个内部代码，填入这几个字符的首字符在键缓区的位置，同时将后续内容（从构成这个保留字的最后一个字符的下一个字符开始）依次排在这个代码位置之后，凡不能构成保留字的字符，不再作换码处理。如此，一直遍历完缓冲区的内容。

这样处理过的缓冲区的内容，除了没有链指针和标号外，和一个程序行在程序区的存放格式是完全一样的。

这时就该执行键缓区的语句或命令了。

执行键缓区的语句或命令，实际上又是一次对键缓区的扫描，因为键缓区的内容是依次一个一个被提供解释的。

执行键缓区内容时，可能遇到以下几种情况：

1.键缓区的首字符若非保留字内部代码，这就要检查是否为DOS命令。如果是DOS命令，由DOS去处理，如果不是DOS命令

或DOS不在内存中，按出错处理。

2.若键缓区是BASIC保留字的内部代码，转该保留字入口解释执行。

3.特别地，当键缓区首字符是RUN的代码时，转RUN的入口。

在执行RUN命令时，它首先将数组表首址与尾址指针都设置在变量表首址所指的地址，这就是所谓的清除所有变量和数组的功能。

其次，设置一个用于对程序区进行扫描的指针，使它先指向程序区首地址。

接着，检查程序区扫描指针是否指向程序尾。如已到程序尾，说明程序运行结束，转出至等待键盘输入的程序段。如果程序区指针未达到程序尾处，取该行程序的语句执行，也就是转相应的保留字入口。当一个程序行执行完毕，程序区扫描指针便指向程序区的下一个程序行，作同样处理，直到执行完全部BASIC程序。

每个保留字都有自己的内部代码，根据这些代码，都可以找到各自的解释入口。各保留字的入口，见附录2。

二、各种表的建立与使用

程序在运行过程中，总要遇到各种量，有常量、有变量、有数组（实际上是下标变量），而且数据类型又可能不同，有实型量、整型量与字符串型量。程序是如何保存和使用这些数据呢？

基本思想是：所有常量采自程序行本身；所有变量，构造一个变量表；对于数组，由于其结构的特殊性，另辟一个数组表。程序中需要保存某个变量时，就在变量表中登记它。需要使用某个变量的数据时，就到变量表按名查找它（数组也按同样的方法处理）。一个变量（或数组元素）在需要使用它而从表中查不到时，就在相应表尾上登记它，并给它以0值（对字符串变量稍不同）。一个变量（或数组元素）得到新的数据时，从表中找到它的名字并以新数取代旧数。表中无此变量名（或数组名）时，同样在相

应表尾上登记它。

变量表通常接在程序区之后。数组表总是紧跟在变量表之后。各表中的名字（变量名、数组名）总是以程序中运行时出现的次序为次序的。

1.变量表

变量表中每个变量占七个字节，前两个字节为变量名的ASCII码。变量名多于两个字符时，从第三个字符开始略去不管；变量名只有一个字符时，变量表名字栏的第二个字节暂填入0值。

变量名因数据类不同，也加入不同标志。对实型变量名，变量名栏内两个字节的最高位都置为0；对整型变量名，两个字节的最高位都置为1；而对字符串变量名，第一个字节的最高位置0，第二个字节的最高位置1（由此也可看出：两个相同的变量名，只要有不同的尾符，程序就能加以区分）。

变量名之后的五个字节存放该变量的数据。

实型数据用一个字节存放指数，后四个字节存放尾数。

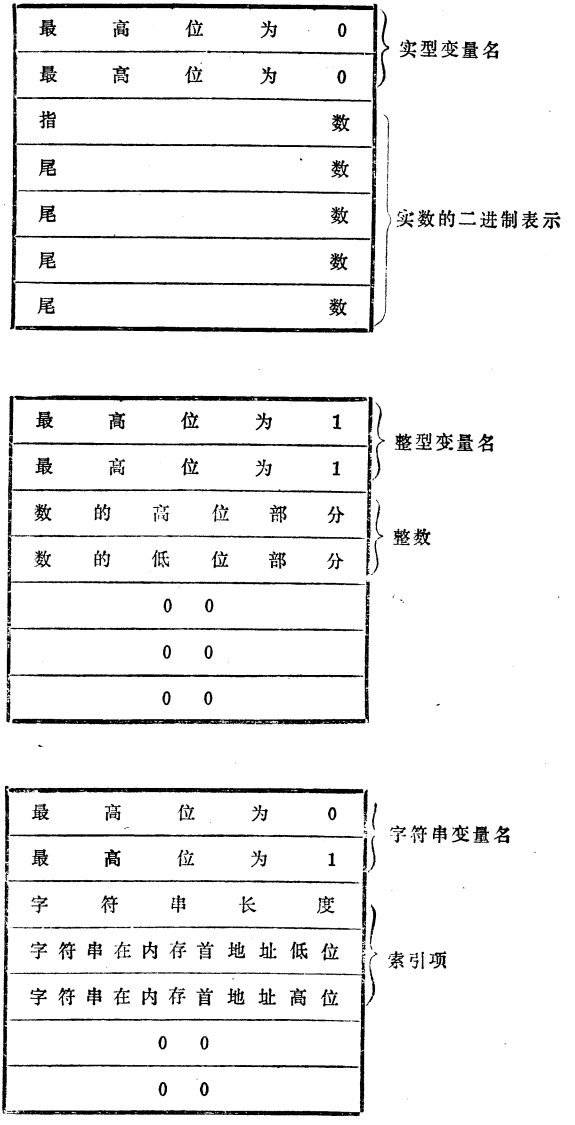
整型数据，第一个字节放该数的高位部分，第二个字节放该数的低位部分(高位在前低位在后)。后三个字节不用，填入0。

字符串型数据，在这里只存索引项。字符串的值则存放在程序区或字符串数据存贮区中。索引项占用三个字节：第一个字节存放该字符串的长度（由此可知，为何一个字符串最多不得超过255个字符），接下去两个字节存放该字符串值在内存中存放位置的首地址。最后两个字节不用，填入0。

图 12-7 是三种数据类型的变量在变量表中存放格式的示意图。它们在内存中的存放顺序，是按程序运行时出现的先后顺序，一个挨着一个存放的。

现将上节存入程序区的那个程序运行后的变量表列出如下：

```
8B8-   41 00 85 48 00 00 00 42
8C0-   00 84 20 00 00 00 41 80
```



各变量名栏，后七位存变量用的字符的ASCII码

图12—7

```
8C8-   05 35 08 00 00 42 80 05
8D0-   F4 95 00 00 43 00 85 20
8D8-   00 00 00 C1 80 00 05 00
8E0-   00 00 49 00 84 30 00 00
8E8    00
```

其中每七个字节中的前两个字节是变量名：

```
41  00  —A
42  00  —B
41  80  —A$
42  80  —B$
43  00  —C
C1  80  —A%
49  00  —I
```

程序中变量A经过两次赋值，执行30行时为浮点数5，执行110行后变为25。在$8BA～$8BE中是它的浮点表示。

执行130行时，A%取5。它存放在$8DD和$8DE中。

关于字符串型变量A$，B$的值，因情况不同，处理也不一样。

程序50行是为A$赋值的。赋值号之后是以字符串常量出现的，所以在变量表中登记A$的数值时，索引地址指向程序区该串值存放的第一个单元，即$835。长度为5。程序继续运行下去，其值未曾修改过，索引地址与长度也均未改变。

程序中的B$是经过几次改动的，而且有三个INPUT B$。它们的执行过程如下：

执行60行的INPUT B$时，假定你输入3个字母：ACE，它们依次排列在键缓输入区中。当敲入RETURN键后，对缓冲区内的信息进行处理（注意在此对INPUT语句的处理，不同于对键入程序或命令的处理）。它首先在变量表中查找INPUT语句带的变量名。如找不到就在变量表尾填入它，并修改数组首地址和尾地

址的指针(各加7)接着填入数据。若变量表中有此变量名，把数据填入。填入数据时，依字符串数据区指针指明的位置，存放在它前面（号码小）的一个区域中。然后，字符串数据区首指针又指向新入字符串之首。同时，将索引项(长度、索引地址)填入变量表相应字符串变量名下。执行60行之后，各有关单元如图12-8从(*a*)变成(*b*)。其中41，43，45分别为A,B,C的ABCII码。

执行70行，与执行60行完全类似。这时除了字符串存贮区指针起始位置有所改变外，你完全可以认为前面那个INPUT B$已不存在。事实上执行 70 行时，是不管原来 B $的值的。这是比较容易处理的一种方法。如果对70行的INPUT B$输入四个字符B，各有关单元的变化如图12-8-(*c*) 。同理执行100行时也一样处理，设执行100行时输入为两个字母C。各有关单元应如图 12-8-(*d*)。

在执行120行时，A $仍为原来的“ABCDE”（A $的索引项未改动过)。而B$，从变量表(*d*)查得它指向 $95F7 开始的两个字节，因而其值为CC。这样A$ + B$的结果是“ABCDECC”，共7个字节。由于120行是将A$ + B$之和赋值给B$的，“ABCDECC”这七个字符需送字符串存贮区，并让B $指向它。送入时，先将字符串存贮区首地址指针减 7，变成$95F0，然后将结果存入它所指向的地址开始的7个单元。并在变量区的B $变量名下填入相应的索引项。从而变成如图12-8(*e*)的样子。

字符串型量与数型量处理上的这种不同，道理是很简单的，因为串型值没有固定的长度，五个字节往往不够使用。

最后说一点，尽管整型变量和字符串型变量在变量表中分别有三个和两个字节置0不用，显得有些浪费，但这种每7个字节代表一个变量的整齐结构，为查找变量名提供了很大方便。

2.数组表

同变量表一样，数组表也是在程序执行过程中建立和使用的。

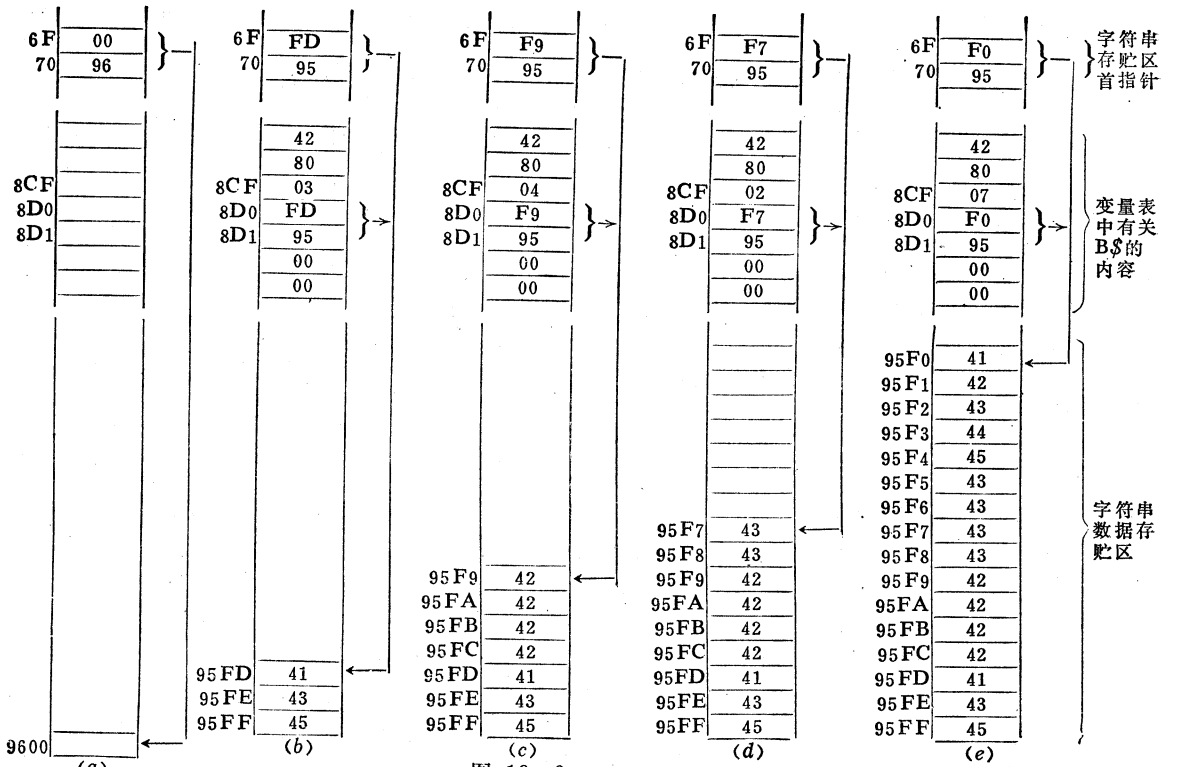

6F 00
70 96
8CF
8D0
8D1
9600
(a)
6F FD
70 95
42
80
8CF 03
8D0 FD
8D1 95
00
00
95FD 41
95FE 43
95FF 45
(b)
6F F9
70 95
42
80
8CF 04
8D0 F9
8D1 95
00
00
95F9 42
95FA 42
95FB 42
95FC 42
95FD 41
95FE 43
95FF 45
(c)
6F F7
70 95
42
80
8CF 02
8D0 F7
8D1 95
00
00
95F7 43
95F8 43
95F9 42
95FA 42
95FB 42
95FC 42
95FD 41
95FE 43
95FF 45
(d)
6F F0
70 95
42
80
8CF 07
8D0 F0
8D1 95
00
00
95F0 41
95F1 42
95F2 43
95F3 44
95F4 45
95F5 43
95F6 43
95F7 43
95F8 43
95F9 42
95FA 42
95FB 42
95FC 42
95FD 41
95FE 43
95FF 45
(e)
字符串存贮区首指针
变量表中有关B$的内容
字符串数据存贮区

图 12—8

尽管你可能在程序一开始就用了数组说明语句DIM，但它只能建立一个表头，留出一块空间。当着变量表的变量逐步增多，这个变量表的扩充而后推。不过，通过数组表的首尾指针，总能找到它们数组表随每个分量的位置，这就为建立与使用它作了保证。

数组表是以一个数组为一个单位依序存放的，最先出现的放在前面，最后出现的放在后面。数组名总是放在一个数组的最先两个字节，同变名量一样标出数据类型标志。接下去是若干个字节的索引项(它与数组名构成表头)。再后为该数组元素的数据。一个数据存放格式因类型不同而异：实型量仍用一个字节记指数，四个字节记尾数；整型量只用两个字节表示(先高位后低位)；字符串型量用一个字节记长度，两个字节为索引地址。不管那种数据类型，总是按一定顺序从0号元素一个接一个排放下去的。一维数组从0号元素开始，接下是1号元素，…二维数组是先行后列。这些前面数组一章已介绍过，不再重述。

现在介绍一下数组表头中的索引项。

索引项有三栏：第一栏用两个字节，记载该数组总长度（从数组名占用的第一个字节起到最后一个元素占用的最后一个字节止)。它主要用来给出下一个数组的位置。第二栏一个字节，记载本数组的维数（其范围从0～$58)。第三栏的字节数是第二栏给出的维数的2倍，所以允许每个数组不等长。它是两个字节作一组，给出一个最大下标值加1的值。它的排列顺序：最先给出的是最后一个最大下标值加1的数（两字节)，最后给出的是第一个最大下标值加1的数。

数组表的结构方式，显示于图12-9中。它也是以程序运行时先后出现的顺序，一个接着一个排放下去的。

下面我们把前面那个程序运行完后的两个数组表头列出如下：(那个程序运行后数组表首地址为$8E9,数组表尾为$D72)

第一个，A数组：

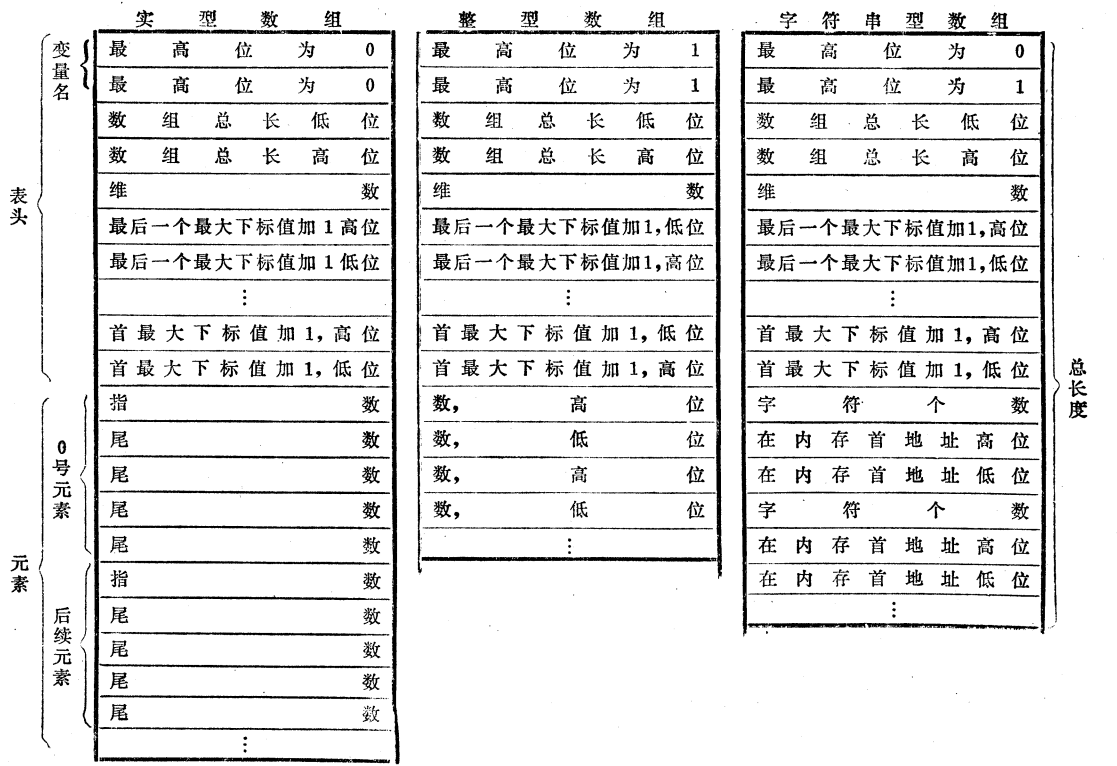

实型数组
整型数组
字符串型数组
表头
元素
变量名
0号元素
后续元素
总长度
最高位为0
最高位为1
最高位为0
最高位为1
数组总长低位
数组总长高位
维数
最后一个最大下标值加1高位
最后一个最大下标值加1低位
最后一个最大下标值加1,低位
最后一个最大下标值加1,高位
首最大下标值加1,高位
首最大下标值加1,低位
指数
尾数
数,高位
数,低位
字符个数
在内存首地址高位
在内存首地址低位

图 12—9

8E9－41 00　19 04　02　00 0D

数组名　总长度　维数　第二个最大下标值加1

8F0－ 00 10

第一个最大下标值加1

第二个，B数组：

D02－42 00　70 00　01　00 15

数组名　总长度　维数　最大下标值加1

注意，在给出最大下标值加1的那个栏中，不仅下标的顺序是倒排的，而且对数的表示也同常规不同，它是先放高位，后放低位。整型数组的元素，也用这种方法。这是为了使用时的方便。

§5. 特殊技巧举例

有了前面这些知识，对于使用好你的苹果机是格外有用的。一些特殊的技巧也就会应运而生，现简略介绍几个设计技巧。

一、节约内存和加快运行的常规方法

以下一些方法，可以减少对内存空间的占用和加速程序的运行：

1.一个程序行中，多写几个语句。

2.少用REM语句。

3.尽量用变量来代替常量。

4.尽量使用相同的变量名。

5.经常使用的变量，应尽早在程序中出现（以便让它处在变量表的开头，查找时能尽快查到）。

6.需由GOTO或GOSUB转入的程序段，尽量使用小行号。因为对它们所带的行号，要扫描程序区来查找的，行号越小，找到得越快。

7.重复使用的程序段，要用GOSUB来做。

8.NEXT后跟的变量名，尽量不用。

9.要求精度不高，且数值范围不大时，尽量使用整型量。

10.尽可能用一些机器语言程序来代替某些高级语言功能。

二、如何恢复一个被破坏了的程序

一个NEW命令，可以清除内存现行程序及其变量和数组。但这样清除了的程序是可以恢复的。

NEW命令的核心，是摘开程序区的头一个链指针并把它置为0（作为程序尾标志），同时将程序尾、变量首、数组首、尾指针都改成LOMEM＋2的值。程序区的其它部分以及变量表、数组表中的内容等根本没有改变。只要把程序区的链头接上，被NEW掉的程序立即可再现出来。若重新运行它，需把变量表首指针改过来，否则会把程序冲掉。如欲敲入新的程序行（包括修改）或将程序存盘，程序尾指针也得改过来。它们的值，在监控状态下可以设法找到。

现举一例，设有程序：

```
10  A = 10
20  B = A * A
30  PRINT A
40  PRINT B
50  FOR I = 1 TO 10
60  A(I) = I
70  NEXT
80  END
```

它在程序区存放形式是：

```
800－  00 0A 08 0A 00 41 D0 31
808－  30 00 14 08 14 00 42 D0
810－  41 CA 41 00 1B 08 1E 00
818－  BA 41 00 22 08 28 00 BA
820－  42 00 2E 08 32 00 81 49
```

```
828-  D0 31 C1 31 30 00 39 08
830-  3C 00 41 28 49 29 D0 49
838-  00 3F 08 46 00 82 00 45
840-  08 50 00 80 00 00 00
```

如果该程序运行过，$847和$848中分别为$41和$00，这是变量A的名字；$69和$6A中分别为$47和$08，它指明变量表的首地址；$6B和$6C中分别为$5C和$08，它指明数组表首地址；$6D和$6E中分别为$9A和08，它指的是数组表尾。而$AF和$B0中分别为$47和$08，指的是程序尾（如果程序没运行，上述几个指针同时指向$0847）。

现在假定用了NEW，再用LIST命令时，什么也列不出来。这时察看程序区，头几个字节是：

```
800-  00 00 00 0A 00 41 D0 31
```

以下的内容没有任何改动。这里可看出它把$801和$802中都改为0了。

这时其它几个指针也恢复到内存初始化时的状态。

读一下程序区的内容不难恢复好使用NEW命令前的程序：

跳过$801至$804（它们是链指针和行号），从$805开始往后查找第一次出现的00字节，这是程序行的终结标志。下一个字节为另一个程序行的头(链指针)，它的位置为$80A，这时把$801中改写为$A，把$802中改写为$8。这样程序就链接上去了，可以LIST出来。

如果想运行它，再在监控状态下继续往后扫描程序区，以找出连续的三个全0字节，这就是程序尾。第一个0字节，是程序末行的终结符，接下两个0字节本应是链指针，而$0不属程序区，用以标志程序结束。这个结束标志要保其完整性，所以一般把变量头指向三个全0字节之后的一个字节。从程序区内容可查出，三个0字节的下一个地址为$847，于是把$69号单元改为$47，而

把$6A号单元改为$8。这样恢复了的程序即可投入运行。

若还想修改程序或存盘，把程序尾指针也指向三个全0字节后的单元就行了。也就是把$AF中改为$47，把$B0中改为$8。

人为地破坏链头，往往可用于程序的保密。

比如，不修改任何指针，程序调试结束后，把$801号单元改为0，这个程序将不能再LIST出来。但它可以存到磁盘上，下次也可以调出来。调到内存中还是LIST不出来。但只要把链头接上，程序可立即使用。

事实上，将链值作任何修改都可从表面上破坏一个程序。比如针对刚刚列出的那个程序，把$80A号单元改为数字1，那么LIST它时，屏上连续显示第一行程序（无休止）。而将$80A号单元改为别的数，情况又不同，屏上可能显示一些乱七八糟的东西。

用这种办法，可以欺骗一些人。

另一方面，由于干扰或使用不当，使内存中的程序遭到破坏时，可用类似的办法修复它。因为这类破坏不大可能是大面积的，往往仅是程序区的某一两个字节。当它仅破坏了程序行的内容时，一般改写那个错误行即可（重新敲入或经过编辑），若是破坏了链指针或行号，正常的办法是无效的。这时可以通过查看程序区内容，找到那个被破坏了的链指针或行号，把它改正过来。

这种修复工作，比你重新敲入那个程序要快得多（特别是较长的程序）。

三、修改某些指针，可以更好地组织你的程序

一个BASIC程序可以配几个机器语言子程序，只要用CALL命令，这些子程序就可以执行。通常的办法是把这些机器语言子程序一个个调试好，并分别冠以程序名，用BSAVE命令存入磁盘。在使用它们的BASIC程序中，再把它们调出来。

这样组织的程序，在运行时需要几次磁盘操作，它不仅花时间，而且增加对磁盘的磨损。

现在我们可以考虑另一种较好的组织程序的方法。

首先找到你的BASIC程序的结尾处。在三个全0字节之后接着存放你的机器语言子程序（为应付必要的修改，最好留出一些空间来）。而把程序尾指针和变量首指针设置于机器语言子程序之后（需修改BASIC程序时，仍需将程序尾指针移到那三个0字节之后）。这样，用SAVE命令存入磁盘时，能把BASIC程序和机器语言子程序一并存入。再用LOAD命令时，就可以一起装入内存。运行时变量表将建立在机器语言子程序之后。这样就减少了访问磁盘的次数。

这样做，要下一些功夫，对一般程序必要性不大，而对于程序较为固定，又经常使用的，作这种处理还是有用的。有些编写的较好的游戏程序，常采用此种处理方法。这里的介绍，也是为了给读者阅读这类程序提供一点线索的。

四、文本和低清晰度图形第二页的使用

在介绍图形绘制时，我们没有提及文本和低清晰度图形第二页的使用方法，通常这种方式是不使用的。这是因为这个页面的存贮区为$800～$BFF，它刚好为BASIC程序区复盖。如果不采取特殊处理，使用这个页面的结果，将会破坏你的程序。

明确了各个指针的功效后，就可以得心应手地处理了。方法很简单。在输入新的程序前，你可以先将$68号单元的数改为$C，而将$C00号单元的数改为0，再用一次NEW命令，目的是根据$67和$68的值修改其它指针。这样，程序区可以改由$C00开始，而将$800～$BFF留作文本和低清晰度图形二页使用。

上述处置方法，可以在BASIC状态下执行下列三个命令实现：

```
POKE  104, 12
```

POKE 3072, 0

NEW

这样处理之后，在你的BASIC程序中，通过屏幕软开关的设置，文本和低清晰度图形都可以有两页用于显示。不仅图形和文本可以交替显示，而且两幅文本页或两幅图形页也可以交替出现。这就增加了屏幕显示的多样化。

五、系统中子程序的直接使用

系统中有些子程序是可以直接用CALL 调用的，它相当于执行某个命令。比如

CALL －936 相当于HOME

CALL －868 相当于编辑字符ESC－F

CALL －922 相当于CTRL－C

CALL －3475 相当于TRACE

CALL －3473 相当于NOTRACE

CALL －384 相当于INVERSE

CALL －380 相当于NORMAL

CALL －3082 可将当前屏幕设置成刚用过的HCOLOR设定的颜色。

等等。

六、在高清晰度图形页内显示字符的方法

在一幅图形页面内显示字符的方法，至少有三种选择：

1.将所需显示的字符，在相应的位置上用HPLOT语句画出。

较灵活的办法是按字符的 ASCII码顺序，对每个字符写一段画点画线的子程序，需要画某一字符时，用ON GOSUB语句转去画出。HPLOT语句中每对坐标点用带参数的相对数字给出，比如画数字1，可用

```
HPLOT X+3, Y+1 : HPLOT X+4, Y TO X+4, Y
+7 : HPLOT X+2, Y+7 TO X+6, Y+7
```

这样，只要给出不同的X，Y参数，这个“1”就可出现在不同的位置上。

这种方法速度较慢，但简单实用。

2.用图形表的办法。

也就是把需要显示的字符做成一个个图形定义，组成一个图形表。使用时调出相应号图形即可。由于调用图形时可以给出起笔坐标，所以能在屏幕的任意位置显示。

以上两种方法，除可显示字符外，当然也可以显示汉字或其它图块。它们的缺点都是速度稍慢。

3.用机器语言来实现。

下面提供出一个方案来，它可以使任何PRINT语句输出的内容，在图形页上显示出来。

它的技巧较高，但明确以下三点也就容易理解了。明确这三点对阅读系统程序或研制质量高的程序都是有帮助的：

首先，要知道一个字节的各位，在高清晰度图形屏幕上是如何排放的。

一个字节是八位，最高位是色选位，右边的七位正好对应屏幕上七个点的位置。它按倒排顺序对应：最右边一位是屏上左边一点，从右倒数第二位是屏上左边第二点…。这样正好40个字节对应屏上一行点的位置。这40个字节是连续的。

为计算方便，我们把想要显示的每个字符的点阵，限制在8×8的范围内。这样我们可在另外一个内存区中，用连续八个字节构造一个点阵，以便把它移到屏上的上下并排的连续八行的位置上，能显示出需要的字符来。下面列出的$1E00开始的八个字节，就是这样构造的“@”字符。$1E08开始是“A”字符，$1E10开始是“B”字符，…这样按ASCII码顺序依次把键盘上的各字符构造出一个表(我们没全列出)。

其次，如果在高清晰度图形一页方式下，屏上第0行对应的

是＄2000开始的40个(＄28个) 字节，第1行对应的并不是＄2029开始的40个字节，而是＄2400开始的40个字节，第3行对应的又是＄2800开始的40个字节。所以对Y轴上的坐标值需要经过计算以后，才能找出该行上最左边的七个点对应于内存中哪个字节。在下面的程序中，从＄1D8D开始的子程序，就是用来完成这种计算的。它从＄11中取出一个行数 (Y 坐标值),运行后则将该行首字节的地址号，存于＄14 (低位) 和＄15(高位)中。比如＄11号单元是0时，＄15中为＄20，＄14中为＄0，构成＄2000地址。

最后还需要知道输出开关单元。所有的PRINT输出，可能送向屏幕，也可能送向打印机，还可能送向磁盘。这就要在执行PRINT时，通过输出开关来控制。输出开关为＄36 (十进制54) 和＄37 (十进制55) 号单元。在这两个字节中设置相应设备的程序入口，它就可以转去。所以它实际上是个转移地址。现在需要它向屏幕上输出我们构造的字符点阵，那就让它转到我们专门为此设计的程序中来。

我们这段程序的入口是＄1DB3，＄1D就是十进制的29，＄B3是十进制的179。

当运行下列程序时：

```
10  HGR
20  HCOLOR = 3
30  HPLOT 100, 10 TO 200 TO 150
40 POKE 54, 179 : POKE 55, 29
50  VTAB 5 : HTAB 10
60  PRINT "ABCD"
70
 ⋮
```

前三行好理解。40行就是改变输出开关位置，使有输出时转入＄1DB3,这样60行的“ABCD”字符就显示在高清晰度图形第了。

一页屏幕上程序不算太长，有兴趣的读者不妨仔细读一下，并试一试。程序清单如下：

```
1D8D- A5 11     LDA  $11
1D8F- 0A        ASL
1D90- 0A        ASL
1D91- 29 1C     AND  #$1C
1D93- 85 15     STA  $15
1D95- A5 11     LDA  $11
1D97- 6A        ROR
1D98- 6A        ROR
1D99- 6A        ROR
1D9A- 6A        ROR
1D9B- 29 03     AND  #$03
1D9D- 05 15     ORA  $15
1D9F- 09 20     ORA  #$20
1DA1- 85 15     STA  $15
1DA3- A5 11     LDA  $11
1DA5- 6A        ROR
1DA6- 29 E0     AND  #$E0
1DA8- 85 14     STA  $14
1DAA- 6A        ROR
1DAB- 6A        ROR
1DAC- 29 18     AND  #$18
1DAE- 05 14     ORA  $14
1DB0- 85 14     STA  $14
1DB2- 60        RTS
1DB3- 48        PHA
1DB4- 8D 50 C0  STA  $C050
1DB7- 8D 52 C0  STA  $C052
1DBA- 8D 57 C0  STA  $C057
```

```
1DBD-  C9 8D     CMP  #$8D
1DBF-  F0 3B     BEQ  $1DFC
1DC1-  29 3F     AND  #$3F
1DC3-  0A        ASL
1DC4-  0A        ASL
1DC5-  0A        ASL
1DC6-  85 0E     STA  $0E
1DC8-  A9 1E     LDA  #$1E
1DCA-  69 00     ADC  #$00
1DCC-  85 0F     STA  $0F
1DCE-  A5 25     LDA  $25
1DD0-  0A        ASL
1DD1-  0A        ASL
1DD2-  0A        ASL
1DD3-  85 11     STA  $11
1DD5-  8A        XTA
1DD6-  48        PHA
1DD7-  98        TYA
1DD8-  48        PHA
1DD9-  A2 08     LDX  #$08
1DDB-  A0 00     LDY  #$00
1DDD-  20 8D 1D  JSR  $1D8D
1DE0-  18        CLC
1DE1-  A5 14     LDA  $14
1DE3-  65 24     ADC  $24
1DE5-  85 14     STA  $14
1DE7-  A5 15     LDA  $15
1DE9-  69 00     ADC  #$00
1DEB-  85 15     STA  $15
1DED-  B1 0E     LDA  ($0E),Y
```

```
1DEF-  91 14          STA  ($14),Y
1DF1-  E6 11          INC  $11
1DF3-  E6 0E          INC  $0E
1DF5-  CA             DEX
1DF6-  D0 E5          BNE  $1DDD
1DF8-  68             PLA
1DF9-  A8             TAY
1DFA-  68             PLA
1DFB-  AA             TAX
1DFC-  68             PLA
1DFD-  4C F0 FD       JMP  $FDF0
1F00-  00 1C 22 2A 3A 1A 02 3C
1F08-  00 08 14 22 22 3E 22 22
1E10-  00 1E 22 22 1E 22 22 1E
1E18-  00 1C 22 02 02 02 22 1C
1E20-  00 1E 22 22 22 22 22 1E
  ⋮
```

程序中用到的$24，$25号单元，为当前光标所在位置寄存单元，$24号存放X方向上列号，$25号存放Y方向上行号(均按文本方式计算)。程序尾的 JMP $FDF0，是转向正常的屏幕输出。因此使用此程序时，不仅有图形中的字符，还产生文本字符。

最后交代一下。程序中若要终止此种特殊输出，可用命令

POKE 54, 189 : POKE 55, 158 : PRINT

返回正常输出开关位置。

七、用机器语言程序绘制高清晰度图形

用机器语言程序绘制高清晰度图形的最大好处是速度快、效率高。写成子程序形式，由BASIC程序通过CALL语句调用它，能满足BASIC语言不容易实现的要求。

上面的在图形屏幕上显示字符的例子，实际上就是一个用机器语言在高清晰度图形屏幕上绘图的实例，只不过它的图块较小(8×8个点阵)，且是字符图形罢了。把图块安排得大一些，把字符点阵改为你所要求的图形点阵，将程序作相应的修改，就可以用来实现高速画图。

现在我们来举一个例子。下列程序可以使屏幕上一个点在第二行上快速地从左边跑到右边。

```
302-  A9 00       LDA #$00
304-  8D 17 03    STA $317
307-  A9 24       LDA #$24
309-  8D 18 03    STA $318
30C-  A9 01       LDA #$01
30E-  8D 01 03    STA $301
311-  A2 07       LDX #$07
313-  AD 01 03    LDA $301
316-  8D 00 24    STA $2400
319-  CA          DEX
31A-  F0 0B       BEQ $327
31C-  0E 01 03    ASL $301
31F-  E0 01       CPX #$01
321-  D0 F0       BNE $313
323-  A9 00       LDA #$00
325-  F0 EF       BEQ $316
327-  EE 17 03    INC $317
32A-  AD 17 03    LDA $317
32D-  C9 28       CMP #$28
32F-  D0 DB       BNE $30C
331-  60          RTS
```

这是一段子程序，框图如图12-10。

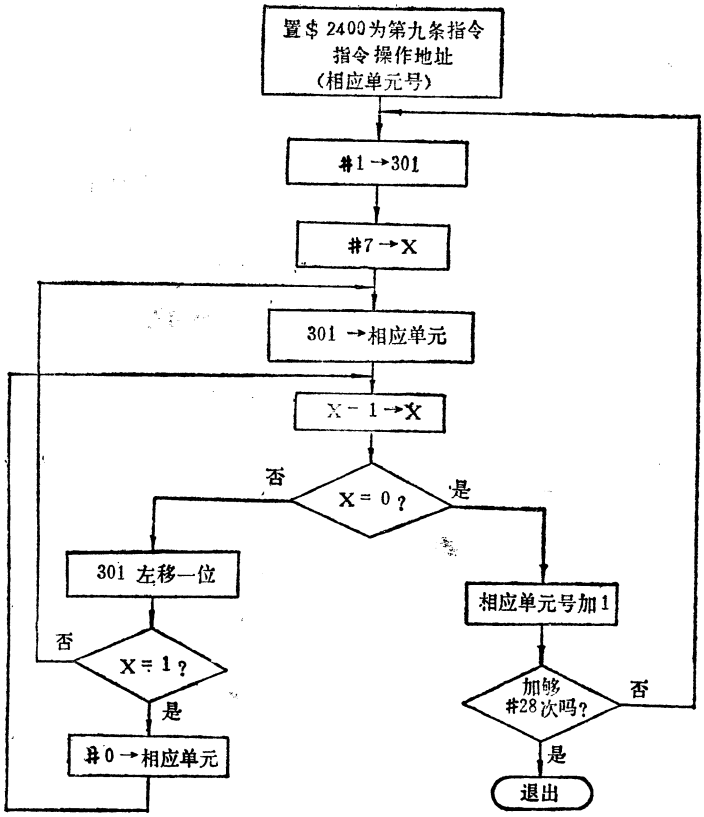


图12—10

第九条指令的操作数地址，图中也称为相应单元。单元号码的初值为＄2400，正好对应高清晰度图形第一页的第二行上的左边七个点的位置。当＄301的初值#1送入＄2400时，屏上第二行最左边出现一个点。当＄301左移一位时，变成了02，把它送＄2400后，屏上第二行左边一点消失，其右侧出现第二点。当显示完7个点之后，要想再向右移动屏上的点，则需将＄2400清除

变成全零，而让$2401得到1。如此直到屏上第二行40（$28)个点的位置都经过，相当于那个点跑到屏的右侧了。这时退出此子程序。

这个子程序的入口是$302，即十进制770。如果运行下面的BASIC程序：

```
10  HGR
20  HCOLOR=3
30  CALL 770
40  END
```

将会在高清晰度图形第一页方式下，使一个白点从第二行的左侧跑到右侧，其速度之快，可以使你根本看不到。

为了证实它的真实存在，可以在上述程序中增加一个延时子程序。监控程序有一段子程序，入口地址为$FCA8，专用于延时的，不妨把它插入上述程序中。

为了不大幅度修改程序，建议将第八条指令加以修改，让其转$332的子程序，从$332开始，除加入延时外，把原第八条指令也写入。这样从$332开始的子程序结束后，返回来再做第九条指令。下面把这些修改部分列出：

```
313-  20 32 03      JSR  $332
332-  A9 80         LDA  #$80
334-  20 A8 FC      JSR  $FCA8
337-  AD 01 03      LDA  $301
33A-  60            RTS
```

$332开始的第一条指令，是使A中置$80，这个值的大小可控制延迟时间。取得大，迟时长；取得小，延时短。$80是一个不大不小的中间值(A中的数可从0～$FF)。

经过这样修改之后，你可以再运行刚才那个BASIC程序，将会清楚地在高清晰度屏幕上看到一个点从左向右的运动情况。

我们这里仅仅给出了一个点的水平运动，多个点的，任意方向上的运动，都可以实现，只是编写程序要麻烦得多。特别要注意的是，Y轴参数是不能直接使用的，必须经过机器语言子程序的计算，才能给出Y轴坐标对应的单元地址。用于这种计算的子程序，在第(六)中已给出，可参阅。

§6. 同系统有关的几个命令和函数

下列几个命令与函数，同系统关系较密切，知道了前面一些内容，使用它们并不困难。

1.HIMEM：命令

格式是：

〔行号〕 HIMEM：算术表达式

它用来设置程序运行时在RAM中可用到的最高地址。对48 RAM的系统而言，从这个值到49152的内存中可放置图形、数据以及机器语言程序，它们不会被运行程序时破坏掉。

算术表达式的值不得大于内存初始化时的HIMEM，而又要比数组尾地址大(否则无法运行程序)。

2.LOMEM：命令

其格式是：

〔行号〕 LOMEM：算术表达式

这个命令用以给出变量表的首地址。

算术表达式的值应大于程序尾地址而小于HIMEM的值。

这个命令常用于保护高清晰度图形。因为一个长的程序，或变量较多或数组较大时，运行中往往能把变量表或数组表延伸到高清晰度图形区内，从而把图形破坏掉；或者画图时把延伸到图形区的变量表或数组表破坏掉。用这个命令可把这些表设置在图形区之后，就都安全了。

3.CLEAR命令

格式是：

〔行号〕 CLEAR

这个命令的功能是不清除程序，但能清除所有变量及数组。

实际上它除了保留程序尾指针不动外，将其它指针恢复成初始化时的状态。

这在某些特殊情况下很有用。比如当一个程序运行的某一时刻，前面用过的所有变量和数组已不再需要了，可用此命令将其清除掉，以腾出内存空间。

4.FRE函数

FRE函数格式是：

FRE(算术表达式)

算术表达式是个哑元，给任一变量名、数字、字母等都行。

这个函数有两个功能：一是清理字符串存贮区的内容。前面介绍字符串变量时，我们见到过：有三个经 INPUT读入的B $，它们都存放在字符串存贮区中，但实际上最先进来的两个串值已无效，但它仍呆在那里。这个函数可以把已作废的内容去掉，留出的空位让存放在较小单元号的内容顺序递补。这样也可腾出一部分空间来。

这个函数的第二个作用是给出一个值，它表明自由空间还有多大。实际上就是指数组尾到字符串存贮区之间的字节个数。这会使你知道你的程序是否还可以扩充或大致能扩充多少。

5.WAIT命令

这个命令的格式是：

〔行号〕 WAIT 算术表达式1，算术表达式2，〔算术表达式3〕

这个命令允许用户在程序中插入有条件的暂停。

算术表达式1，指明一个内存单元地址，它只能是－65535～

65535之间的一个数。

算术表达式2和算术表达式3都是0～255之间的数。执行此命令时，分别把它们换成八位二进制数。

执行这个命令时，把算术表达式1指明的单元中的数取出，也化为八位二进制数。假定我们叫这个数为数1，另外那两个给定的数换成二进制数后分别叫数2、数3。这个命令先作数1与数3的位对位的异或运算（对应位相同时，结果位为0；对应位不同时，结果位为1），这个结果也是八位二进制数，设它为数4，然后数4与数2作位对位的逻辑与操作。如果这一结果为0，则重复上面的工作（相当于程序暂停在那里）；否则WAIT结束，执行后续语句。

不取算术表达式3可选项时，把它当作0来做。

适当选择三个算术表达式的值，可以有条件地使程序暂停。把它用于处理实时数据采集还是很有用的。

6.USR函数

这个函数的格式是：

USR（算术表达式）

这个函数的功能和使用方法比较特殊，没有读过BASIC解释程序时是很难使用的。这里只作简单的交代。

这个函数被调用时，先把算术表达式的值保存在浮点累加器中（不是A寄存器，而是解释程序设置的，为$9D～$A3号单元），然后转$0A去。所以在$0A，$0B，$0C中应设置一条转移指令，或转子指令。如系转子指令，它还会返回来，你还得设一子程序返回指令，才能结束此函数的功能。退出时，它把浮点累加器的当前值作为函数的值。由于$0A开始的字节中可以转某个机器语言程序或子程序，在那里做什么事，SUR函数不管，因而函数带出的值也许与给定的自变量无关。至于从$0A转何处，一般转解释程序的某个入口，这要读过解释程序才能使用。

7.&命令

这个命令可用于两种执行方式，但它多用于立即执行方式。

这也是一个特殊命令，没有机器语言的配合无法使用。

使用这个命令时，必须从$3F5起给出一条转移指令，转至一个机器语言子程序的入口。它虽与CALL命令很相似，但它不需给出任何参数，而且只需击一次键，使用很方便。

苹果机系统主盘上有一RENUMBER程序，功能很强。它可以重排一个程序的行号，可以将一些程序行插入某两个程序行之间，还可将两个程序合并，甚至将若干个程序编辑为一个程序。在这个程序中，较好地使用了&命令。

它的功能是由一个机器语言子程序实现的，把转入这个机器语言的入口地址的指令填入$3F5开始的单元中。当你敲入&命令时，就转入此子程序。这时你按它规定的命令字符敲入，它可完成一种功能，然后退出子程序。这时如果再敲&命令，又可转入该子程序，依你敲入的后续命令完成某种工作。然后再退出来。假如你不再敲&，而给出正常的命令，它一样可以执行。这样，由于&的作用，你的机器就有两套命令随你使用：一套是BASIC的，一套是&的，这就大大提高了机器的功能。

8.SHLOAD命令

此命令的格式是：

〔行号〕 SHLOAD

这个命令的功能是把录在录音带上的图形表，读入内存中紧挨着HIMEM：的那个区域中，然后再把HIMEM：指针自动移到图形表首地址那里，以便把图形表保护起来。

图形表存入磁带的方法，可以用监控命令。需要存入磁带的是两组数：图形表长度与图形表。通常把图形表长度送入$0与$1号单元中。

像我们在第八章第四节中举过的那个图形表的例子，图形表首地址为$6000，末地址为$600D，长度为D，如果想把那个图

形表存入磁带，可以在监控中使用命令：

*0：D␣0↵

*0.1W 6000.600DW↵

第一个命令是向0～1单元中送入图形表长度，第二个命令是两个往带上写数据的连续命令。

这样存于磁带上的图形表，可以用SHLOAD命令调出。

当然要使用这个图形表，还需要给232，233号单元送入图形表首址。232、233就是$E7，$E8，图形表首址可通过查HIMEM：找到（查$73，$74号单元的内容即是），在监控中一样可以向$E7，$E8写入图形表首地址（针对上例，在监控中用命令*E7：00␣96↵就可以了）。

习　题

（一）现有下列程序：

```
10 FOR I=1 TO 3
20 A(I)=I
30 NEXT I
40 END
```

请给出程序区各字节的数值和有关指针单元的数值。

（二）内存中有一个BASIC程序，用NEW命令将其消除掉了。现从程序区读出的内容如下：

```
800—00 00 00 0A 00 41 D0 31
808—30 00 14 08 14 00 42 D0
810—41 CA 41 00 1B 03 1E 00
818—BA 42 00 21 08 28 00 80
820—00 00 00
```

请指出如何恢复这一程序。你能直接读此程序区给出程序清单吗？

（三）有下列一个子程序，请读懂它，并指出它的作用何在？

```
0300—  A9 00      LDA #$00
0302—  8D 65 03   STA $0365
0305—  A2 05      LDX #$05
0307—  BD 00 08   LDA $0800, X
030A—  F0 03      BEQ $030F
030C—  E8         INX
030D—  D0 F8      BNE $0307
030F—  18         CLC
0310—  8A         TXA
0311—  69 01      ADC #$01
0313—  8D 01 08   STA $0801
0316—  8D 22 03   STA $0322
0319—  A9 08      LDA #$08
031B—  8D 02 08   STA $0802
031E—  8D 23 03   STA $0323
0321—  AD 00 00   LDA $0000
0324—  F0 19      BEQ $033F
0326—  A9 00      LDA #$00
0328—  8D 65 03   STA $0365
032B—  18         CLC
032C—  AD 22 03   LDA $0322
032F—  69 01      ADC #$01
0331—  8D 22 03   STA $0322
0334—  AD 23 03   LDA $0323
0337—  69 00      ADC #$00
0339—  8D 23 03   STA $0323
033C—  4C 21 03   JMP $0321
033F—  EE 65 03   INC $0365
0342—  AD 65 03   LDA $0365
0345—  C9 03      CMP #$03
```

```
0347—  D0 E2      BNE $032B
0349—  18         CLC
034A—  AD 22 03   LDA $0322
034D—  69 01      ADC #$01
034F—  85 69      STA $69
0351—  85 6B      STA $6B
0353—  85 CD      STA $6D
0355—  85 AF      STA $6F
0357—  AD 23 03   LDA $0323
035A—  69 00      ADC #$00
035C—  85 6A      STA $6A
035E—  85 6C      STA $6C
0360—  85 6E      STA $6E
0362—  85 B0      STA $B0
0364—  60         RTS
```

（四） 上题寻找程序尾时是逐字节扫描程序区找到的。请改写上题程序，以便通过一个个链指针找到程序尾。

（五） 在输入程序之前使用HIMEM：命令或LOMEM：命令，与在程序一开始使用HIMEM：命令或LOMEM：命令有什么区别？在程序的中部或尾部使用HIMEM：命令或LOMEM：命令，会出现什么情况？

第十三章　整数BASIC简介

整数BASIC(INTBASIC) 虽有固化在 ROM 中的，但目前绝大部分机器是存贮在系统软磁盘上，开机时由HELLO程序将其装入语言扩展板中。

在浮点BASIC状态下，用INT命令可以进入整数BASIC状态，其提示符为“>”。在整数BASIC 状态下，用 FP 可以再转回浮点BASIC状态。

由于我们介绍的是从磁盘上装入内存的整数 BASIC 解释程序，它总是处在有DOS的系统状态下。事实上，没有 DOS支持，整数BASIC一样可以工作。整数BASIC程序在磁盘上的类型标记为I。

整数BASIC的主要优点是：占用内存空间小，运行速度快。

§1. 整数BASIC的特点

整数BASIC在以下几个方面与浮点BASIC不同：

一、数据类型

整数BASIC只有整型量与字符串型量两种数据类型。

1.整型量

整型量的数值范围只能是－32767到32767之间的整数。

可以由下标变量构成数组，但其维数只能是一维的。换言之，下标变量只允许有一个。任何一个数组，必须先说明后使用。一个数组名不得被说明两次。

两个数相除时，如果不能整除，其商仅取整数部分。

2.字符串型量

使用字符串型量时，必须先定义字符串的长度。

字符串不能构成数组。

二、常量

整常量只能以允许范围内的整数形式出现，不得带小数点，也不得用科学计数法表示。

字符串常量须用引号对括住。为INPUT语句输入字符串常量时，如无领头的或拖后的空格及嵌入的逗号，引号对可以略去。

三、变量

变量名必须以字母开头，其后的字符，可以是字母，也可以是数字，字符总数不超过100个就行。但下列任意一个字不得包含于变量名中：AND、AT、MOD、OR、STEP或THEN；同时下列字不得用于变量名的开头：END、LET、或REM。

这里没有象浮点BASIC那样只对变量名的前两个字符加以区分。在整数BASIC中，两个变量名即使左边几十个字符对应一致，只要最后有一个字符不一样，都能区分为两个变量名。这不仅可以使变量名能用一些有意义的单词(不只是一个代号)，而且大大扩充了变量名的可取范围。

字符串变量名，必须以$结束。

整数型变量名，不带任何尾符。

四、程序行

一个程序行不得超过150个字符，只要在这个范围内，一个程序行允许写多个语句，但每两个语句之间，同样得用冒号分隔开。

五、运算符

整数BASIC的运算符如下：

小括号 （ ）

单边运算符 +、－、NOT（逻辑非）

算术运算符：

↑

*、／ MOD

+、－

= （赋值号）

关系运算符：

=、≠或<>、>、<、>=、<=

逻辑运算符：

AND

OR

各运算符的含义，与在浮点BASIC中无差异，但要注意下列几点：

1.MOD是模运算，它给出两数相除的余数。如

3 MOD 2=1，9 MOD 5=4

2.字符串只有=和≠（不等）两种比较运算，而无别的运算功能。对字符串作不等比较时，只能≠号不能用<>号。

3.<>号不得写成><

<=不得写成=<

>=不得写成=>

六、函数

由于整数BASIC只能进行整数运算，所以函数的使用受到了限制。浮点BASIC中的 SIN、COS、TAN、ATN、LOG、SQR、EXP，都不再能使用；INT函数也没有必要。所以，整数BASIC的算术函数只有三个：

ABS(X)—给出X的绝对值。

SGN(X)——给X的符号：当X<0时，为负1；X=0时，为0；X>0时，为正1。

RND(X)——给出0到X（不含X）之间的一个随机整数。

和字符串有关的函数也只有两个：

ASC(X)——给出指定字符串的首字符的ASCII码的十进制数。此处X为字符串变量名或用引号对括起来的字符串。

LEN(X)—给出X的字符个数。此处X也是字符串变量名或用引号对括起来的字符串。

和低清晰度图形有关的函数，仍是：

SCRN(X，Y)—给出坐标点（X，Y）处的颜色号码。

检测游戏控制器旋钮转动值的函数，仍是：

PDL(X)—给出相对于X号旋钮旋转角度的一个正整数(0～255)。X必须在允许范围内。

和系统有关的函数，只有：

PEEK（地址）——给出指定地址内的数的十进制表示。地址号码大于32767时，必须用负地址的形式给出(因为整数BASIC不认识大于32767的数)。

打印格式函数：

TAB（X)

同浮点BASIC中一样，它只能跟在保留字PRINT之后，用来指定打印头和光标的位置，X只能取1到40的整数。

整数BASIC不允许用户自己定义函数。

七、图形方式

整数BASIC只有低清晰度图形方式，没有高清晰度图形方式。

八、语句和命令

整数BASIC的语句和命令，比浮点BASIC的语句和命令要少，同时也增加了一些新的语句和命令。有些语句和命令虽同浮点BASIC的定义符一样，但使用时有某些区别。下节将详细介绍这些内容。

九、没有READ/DATA语句

取得数据的方法少一种。

§2. 整数BASIC的语句和命令

为了叙述方便，分作几类加以介绍。

一、和浮点BASIC基本相同的语句和命令

和浮点BASIC基本相同的语句和命令有：

GR

COLOR =

PLOT

HLIN

VLIN

TEXT

PR#

IN#

LIST

LOAD

SAVE

RUN

NEW

REM

LET

VTAB

NOTRACE

二、和浮点BASIC稍有差异的语句和命令

1.INPUT语句

在执行INPUT语句时，如果要求用户输入字符串值时，不给出问号提示，仅有一个闪烁光标；且任一个字符串值输入后，必

须紧跟着回车键。INPUT语句要求输入数值时，只给出单一问号提示。

此外，INPUT语句可以带提示字符串，但提示字符串之后应紧跟着逗号而不能用分号。

2.PRINT语句

它不能用输盘上输入“？”代替。

PRINT的输出项跟有逗号时，仍为分区输出，但每区为8个字符位置，一行共分5区。

其它同浮点BASIC的PRINT。

3.END

必须由END结束程序运行。否则给出出错提示。

4.FOR…TO…〔STEP…〕…NEXT

除了要求初值、终值和步长必须是整型量外，循环语句在整数BASIC中不能用于立即执行方式。此外，STEP跟的算术表达式，不得为负数，即增量只能为正；NEXT后跟的变量名不得省缺。

5.IF…THEN

IF语句执行方式与浮点BASIC不同：当条件不满足时，不是直接转下一个程序行，而是跳过THEN带的语句，执行后续语句。比如：

```
⋮
100  IF A=15  THEN 120 : B=10
110  C=A+B
120  PRINT C
⋮
```

当变量A为15时，执行这段程序中的所有语句。而变量A取值不是15时，仅跳过THEN 120，后续的B=10还是要执行的。

IF语句的这种执行方式，允许在一个程序行中写几个IF语句。

6.GOSUB…RETURN 和 GOTO

GOSUB 和 GOTO 所带的行号，不仅可以由整数给出，而且可以由算术表达式给出。

比如：

GOSUB N+3

GOTO M*5

等都是合法的，只要程序行中有这些行号就行。

7.CALL

CALL所带的地址号，小于 32768 时，必须用正地址；大于32767时，必须用负地址。

比如：

CALL 770，不能用 CALL－64766

CALL－936，不能用CALL 64600

8.POKE语句

往一个小于32768的地址中送一个数时，必须用正地址。而往一个大于32767的地址中送一个数时，也必须用正地址。

比如：

POKE 50，255，不能用 POKE－65486，255

POKE－16384，0，不能用 POKE 49152，0

9.DIM语句

对于整型量，它可以说明一个一维数组。即通过这个语句，给出一个一维数组的最大下标值，系统按最大下标的2倍给该数组各元素留出空间来(因为任一整型常量，只需两个字节即可表示)。不给出DIM说明，任何数组都不能使用（不象浮点BASIC，给每个下标值以10预留空间)。

对于字符串型量，必须先用DIM语句说明该字符串的长度，系统以一个字符占一个字节留出相应的空间来。一个字符串的长度不得大于255。没有用DIM说明过的任何字符串型量，都无法使用。

甚至连下面的语句都是非法的：

```
10 A$ = "GOOD"
   10. POP
```

只能用于延迟执行方式。

11.LOMEM：和HIMEM：

这两个命令，只能用于立即执行方式。即不得写在程序行中。

12.DEL

也只能用于立即执行方式。

13.TRACE

当一个程序行内有多个语句时，它只显示一个行号（浮点BASIC是每执行一句，都要显示这一个语句所在的行号的）。

三、专为整数BASIC设计的语句和命令

1.AUTO命令

这个命令的格式是：

AUTO 数1〔，数2〕

这个命令用来置自动行号方式。当使用这个命令之后，输入一个程序时，用户可以不要输入一个个行号，而由机器自动以等间隔的方式提供使用。

数1，指明程序的起始行号；数2指明行号间隔。当可选项省缺时，机器自动以数2为10处理。

2.MAN命令

MAN命令格式是：

MAN

这是一个撤销AUTO的命令，它必须跟在CTRL－X之后。这个命令敲入后，机器不再自动提供行号。如果这时要输入新的程序行，只能由用户输入行号及语句。

3.DSP语句

DSP语句的格式是：

〔行号〕DSP 变量名

这是一个置检查方式语句。它的功能是：当给定的变量的取值发生变化时，连同行号一起显示出来。对数组元素，只能给数组名，机器自动按下标当时的值取相应元素显示。

RUN命令可以清除DSP方式。因此，立即执行的DSP使用之后，不要用RUN命令，而改由GOTO语句来运行程序，以免DSP的失效。

CON命令也不破坏DSP。

4.CON命令

CON命令的格式是：

CON

它相当于浮点BASIC中的CONT命令。用来继续运行被CTRL—C中止的程序(整数BASIC运行时的中止，只有CTRL—C命令。整数BASIC中，没有STOP语句)。

5.CLR命令

CLR命令格式是：

CLR

它相当于浮点BASIC中的 CLEAR 命令。它可以使数型变量清0、数组维数和字符串变量的长度清0（RUN也有这类功能)。

四、下列保留字只能用于浮点 BASIC 中，而不能用于整数 BASIC中

ATN、CHR $、COS、DATA、DEF FN、DRAW、EXP、FLASH、FN、FRE、GET、HCOLOR=、HGR、HGR2、HOME、HPLOT、INT、INVERSE、LEFT $、LOG、MID $、NORMAL、ON…GOSUB、ON…GOTO、ONERR… GOTO、POS、READ、RECALL、RESTORE、RESUME、RIGHT $、ROT=、SHLOAD、SIN、SPC、SPEED=、SQR、STOP、STR $、TAN、USR、VAL、WAIT、XDRAW、&。

五、代用命令

用CALL－936，可以起到HOME的作用。

用POKE 50，63，等同于浮点BASIC中的INVERSE命令。

POKE 50，127，等同于浮点BASIC中的FLASH命令。

POKE 50，255，等同于浮点BASIC中的NORMAL命令。

§3．整数BASIC程序例

例1． AUTO和DSP的使用方法

下列整数BASIC程序，可用来求某班学生期终考试各分数段人数。设这个班共有学生50名，每人考6门课。程序应该是这样：

```
10 DIM A  (10)
20 FOR I=0 TO 10
30 A (I)=0
40 NEXT I
50 FOR I=1 TO 50
60 B=0
70 FOR J=1 TO 6
80 INPUT C
90 B=B+C
100 NEXT J
110 D=B/6/10
120 A(D)=A(D)+1
130 NEXT I
140 FOR T=0 TO 9
150 PRINT I*10; "- -"; I*10+9; ":", A(I)
160 NEXT I
170 PRINT "100:", A(10)
180 END
```

在输入这个程序时，如果你不使用AUTO 命令，那就像在输入浮点BASIC程序时一样，先敲入行号，再敲入语句，最后敲回车符，这就算输入了一个程序行。后续的程序行照此一个个输入，直至输入结束。

如果你使用AUTO 命令，则应先敲入此命令。为表达清楚，在这里，我们把用户敲入的字符，都用下横线标出，凡未标下横线者，为机器自行提示的内容：

```
>AUTO 10 ↙
>10 DIM A (10) ↙
>20 FOR I=0 TO 10↙
>30  A(I)=0↙
>40  NEXT I↙
>50  FOR I=1 TO 50↙
>60  B=0↙
>70  FOR J=1 TO 6↙
>80  INPUT  C↙
>90  B=B+C↙
>100 NEXT J↙
>110 D=B/6/10↙
>120 A(D)=A(D)+1↙
>130 NEXT I↙
>140 FOR I=0 TO 9↙
>150 PRINT I*10;"--"; I*10+9; ":", A(I)↙
>160 NEXT I ↙
>170 PRINT "100 :", A(10)↙
>180 END↙
>190 CTRL-X\
>MAN↙
>
```

这时，退出了自动编行号方式。程序方可以运行。对初用者

要注意：不退出自动编行号方式，你敲入的任何内容（那怕是立即执行的命令），都当作延迟执行的程序行中的语句；如果敲入的不是语句，给出出错提示，仍等待你输入语句。换句话说，只有用CTRL－X、MAN命令退出自动编行号方式之后，才能再敲入别的立即执行命令。

对上面一个程序，如果想知道D的变化情况，加入下列程序行

```
115 DSP D
```

运行中只要D值有变化，就可显示改变值的语句所在行号（在此为#110）和D的新值（D = × × ×）。

例2.把任一个由键盘输入的正数转换为十六进制数，并显示出来。

程序如下：

```
10 DIM A$(1), B$(1), C$(1), D$(1), E$(1)
20 INPUT A
30 IF A<0 THEN 20 : C=A
40 K=0
50 IF K=4 THEN 500
60 GOSUB 100
70 GOTO 50
100 B=A MOD 16
110 IF B<10 THEN GOSUB 400+B*10
120 IF B>9 THEN GOSUB B*10+100 : GOSUB 300+K*10
130 K=K+1
140 A=A/16
150 RETURN
200 E$="A" : RETURN
210 E$="B" : RETURN
220 E$="C" : RETURN
```

```
230 E$ = "D" : RETURN
240 E$ = "E" : RETURN
250 E$ = "F" : RETURN
300 D$ = E$ : RETURN
310 C$ = E$ : RETURN
320 B$ = E$ : RETURN
330 A$ = E$ : BETURN
400 E$ = "O" : RETURN
410 E$ = "1" : RETURN
420 E$ = "2" : RETURN
430 E$ = "3" : RETURN
440 E$ = "4" : RETURN
450 E$ = "5" : RETURN
460 E$ = "6" : RETURN
470 E$ = "7" : RETURN
480 E$ = "8" : RETURN
490 E$ = "9" : RETURN
500 PRINT C; " = $"; A$; B$; C$; D$
510 END
```

这个程序运行时，只要输入一个正数，就会输出相应十六进制表示的数来。如：

输入：256

输出为：256 = $0100

输入：1025

输出为：1025 = $0401

程序中第30行的C = A，在浮点BASIC中是永远执行不到的。而在整数BASIC中，只要前一个语句的条件不成立，就会执行到。

程序 120 行第一个语句中的条件成立与否，第二个语句总是要执行的（在浮点BASIC中则不然）。

110行和120行的 GOSUB 语句，子程序入口行号的给出，都用了算术表达式，这在浮点BASIC中是不允许的。在这里，实际上相当于浮点BASIC中的ON…GOSUB语句。

程序中使用了算术运算符 MOD，它的取值为两个数相除的余数。对于一个十进制数而言，MOD 16，结果是该数用十六进制表示时的个位。在这里它仍用十进制的表示法。通过400～490行和200～250行的子程序，把它换成字符形式的十六进制表示法。对该数的十六进制数表示的其它各位(总共四位)，程序都作了类似处理。最后的结果，分别存放于A$，B$，C$，D$中。

由于整数BASIC中没有字符串加法运算（连接）功能，只好通过PRINT语句中分号的作用，形式上输出一个十六进制表示的数（在机内实际上并没有存放成一个数型量的十六进制数。那样的数，机器会自行转换的，而打印出来还是十进制数，这不是题目所要求的）。

§4．整数BASIC解释执行过程

整数BASIC解释程序大体上是按下面给出的粗框图中的步骤（图13—1）工作的。

现分别简述于下：

一、开工与接收输入

开工程序和浮点 BASIC 很类似。它的任务主要是检测内存RAM的大小，以确定用户可用的最高地址，即 HIMEM：值，并给出程序区和数据区的指针。

对于48KRAM 的系统而言，它的HIMEM：值同样是 $9600。

HIMEM：值，也被当作程序尾地址指针（实际上是程序尾地址加 1 ）。换句话说，程序区是紧贴着HIMEM：所指明的地址的(它相当于浮点BASIC中的字符串数据存贮区)。

整数BASIC 中，用户可用的最低地址通常为$800，它用

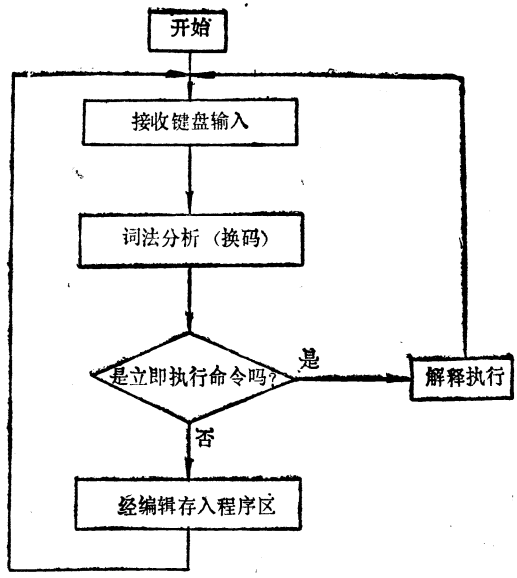


图1 —1

LOMEM：指明。整数BASIC 运行时建立的数据区（相当于浮点BASIC 的变量表、数组表），就从这里开始（相当于浮点 BASIC 的程序区）。

内存检测结束后，把 LOMEM：值存入$4A 和$4B 中，把HIMEM：值存入$4C和$4D中。也就是说，$4A、$4B和$4C、$4D分别为两个指针，一个指向数据区头，一个指向程序区尾。另外还建立了两个指针：一个在$CA和$CB 中，是指向程序区首的指针；另一个在$CC和$CD 中，是指向数据区尾的指针。由于开工时仍无 BASIC 程序及数据，所以先将程序区首指针指向HIMEM：，而将数据区尾指针指向 LOMEM：。这些工作，实际上就是内存的初始化。

对于RAM为48K的机器来说，初始化之后各指针单元的初始值，如下表所示：

指针名称	指针单元	初始化的值
LOMEM：数据区头指针	4A	00
	4B	08
HIMEM：程序区尾指针	4C	00
	4D	96
程序区首指针	CA	00
	CB	96
数据区尾指针	CC	00
	CD	08

开工程序结束后，给出整数BASIC提示符“>”及闪烁光标，等待用户的键盘输入并接收它们。

接收键盘输入的过程，和浮点BASIC状态下的情形基本类似。键盘缓冲区仍在$200～$2FF中，收入一个字符显示一个字符。当接收到RETURN键入时，结束接收工作，转词法分析程序。

以上开工和接收键盘输入的程序，入口为 $E000。

二、词法分析

词法分析就是对键盘缓冲区的数据进行处理。它的主要任务是将输入至键缓区的程序行或命令中的十进制数换成二进制数，把语句定义符或命令转换成内部代码，然后分析键缓区收入的是否立即执行命令。如果是立即执行命令，转相应入口解释执行。如果是延迟执行程序，编辑到程序区中。如果两者都不是，查是否DOS命令。如是DOS命令，由DOS控制处理；如果不是DOS命令，那只能认为是错误的输入，给出出错提示信息。

十进制换成二进制的程序，入口为 $E4DA。语句定义符换

为内部编码的程序，入口为 $E3FF。

整数BASIC各定义符的内部编码，参阅附录2。

出错信息表，见附录3。

三、将键缓区的源程序送入程序区

键盘缓冲区接收到的如果是带行号的程序行，将编辑入程序区中。

1.源程序在程序区的存放格式

从宏观上讲，源程序总是贴着HIMEM:指向的地址存放的。行号小的程序行在前（地址号码小，离HIMEM远），行号大的程序行靠后，按行号大小顺序排列的。

而就一个程序行来说，它的第一个字节存放这个程序行的长度，也就是这个程序行在内存程序区中占用的字节数。第二、三个字节存放这个程序行行号的十六进制数。从第三个字节起存放这个程序行的语句。一个程序行的结束，用一个字节为01的数作标志(它实际上是敲入键盘上的RETURN转化过来的)。

这种结构，示意于图13—2中。

比如，有下列一个整数BASIC程序：

```
10  ABCD = 5
20  ABC = ABCD + 6
30  DIM ABCDE(3)
40  FOR I = 1 TO 3
50  ABCDE (I) = I
60  NEXT I
70  DIM ABC$(4)
80  ABC$ = "AA"
90  DIM AAA$(4)
100 INPUT AAA$
110 INPUT ABC, S
120 END
```

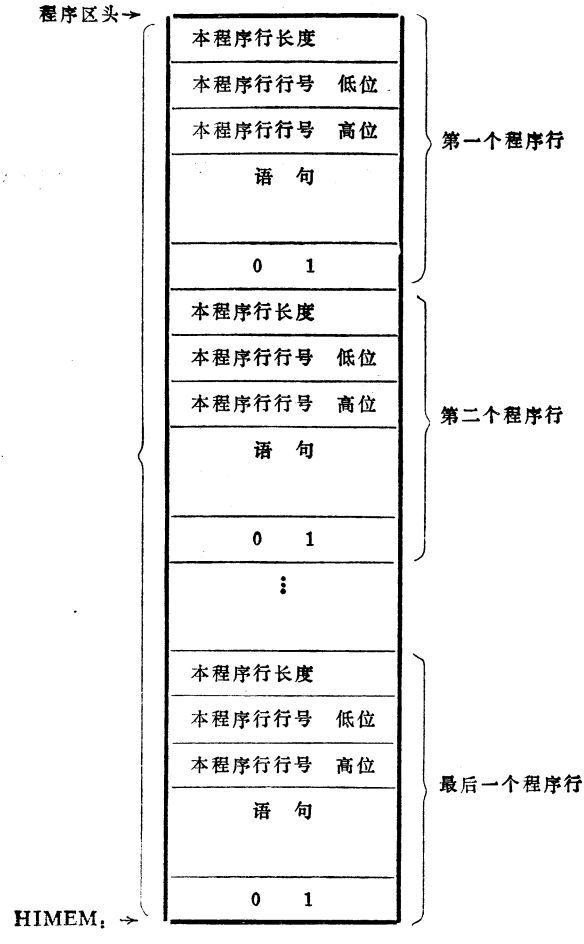


图13—2

将这个程序输入机器后，我们从监控中可以查到当前各指针的值：

4A－00
4B－08 } $800—数据区首

4C－00
4D－00 } $9600—程序区尾

CA－72
CB－95 } $9572—程序区首　　CC－00
CD－08 } $800—数据区尾

只要找到程序区首尾指针，就可以把程序区的内容列示出来。在监控中用下述命令：

```
*9593.72FF↙

9570- 01 01 0C 0A 00 C1 C2 C3
9578- C4 71 B5 05 00 01 10 14
9580- 00 C1 C2 C3 71 C1 C2 C3
9588- C4 12 B6 06 00 01 0F 1E
9590- 00 4F C1 C2 C3 C4 C5 34
9598- B3 03 00 72 01 0E 28 00
95A0- 55 C9 56 B0 00 00 57 B3
95A8- 03 00 01 0E 32 00 C1 C2
95B0- C3 C4 C5 2D C9 72 71 C9
95B8- 01 06 3C 00 59 C9 01 0E
95C0- 46 00 4E C1 C2 C3 40 22
95C8- B4 04 00 72 01 0D 50 00
95D0-  C1 C2 C3 40 70 28 C1 C1
95D8-  29 01 0E 5A 00 4E C1 C1
95E0-  C1 40 22 B4 04 00 72 01
95E8-  09 64 00 52 C1 C1 C1 40
95F0-  01 0A 6E 00 54 C1 C2 C3
95F8-  72 D3 01 05 78 00 51 01
```

试以第一个程序行为例：

$9572中的$C，表明这个程序行在程序区中占用$C(十进制12）个字节。

$9573和$9574中，是本程序行行号的十六进制表示$000A(即十进制的10)。

$9575～$9578中的$C1，$C2，$C3，$C4为A、B、C、D四个字符的ASCII码（它与浮点BASIC不同)。

$9579中的$71，为“=”（赋值号）的内部代码。

$957B和$957C中的$0005，为数字5的十六进制表示。而$957A中的B5，是换码后留下来的（键盘输入时，数字5的ASCII码为B5）这里后四位的5，没有任何用处（因为它已换为十六进制$0005存于$957B和$957C中），而前四位的B，将作为十六进制常数的标志，为以后的解释执行提供信息。

$957D的01，则是本程序行的结束标志。

其它程序行，读者可对照阅读。

整数BASIC程序区，没有程序结束标志。程序结束位置，由HIMEM:指明。

2.程序由键盘缓冲区往程序区编辑的简要过程。

这一过程和浮点BASIC的处理方法有类似之处，但也有很大不同，这是因为程序区排放格式不同所产生的。但整数BASIC比浮点BASIC简单些，因为它没有链指针，在插入一个新的程序行或删除一个程序行等情况下，需要将程序区的某些程序行上推下移时，无需一个个计算、修改链指针。

整数BASIC的一个程序行移入程序区时，首先看程序区有无程序。如果程序区首址和HIMEM等值，说明程序区尚无程序。这时将键缓区的程序行，按程序区的规定格式，移至紧贴HIMEM:的区域，并修改程序区首指针就行了。

如果程序区已有程序，那就取程序区的一个个行号从小到大，依次与键缓区的那个程序行号相比较。如果找到了一个相同的行号，再视键缓区的内容而定：若是删除这个程序行，则按程序区原来那个程序的长度，将比它小的所有程序行向后移动那么多个字节，同时修改程序区首指针。如果是修改原来那行程序，则比较新旧程序行的长度，使比它行号小的那些程序行，上推或下移适当字节位置，将键缓区程序行移入程序区（旧的程序行被冲掉了），并修改程序区首指针。

如果键缓区行号比程序区所有程序行号都小，则键缓区程序行移至程序区最顶部，修改程序首指针。

如果键缓区行号比程序区所有程序行号都大，则上推原程序区所有程序行，将键缓区程序行置于紧贴HIMEM:的部位，修改程序区首指针。

如果键缓区行号介于程序区两个程序行号之间，上推较小的所有程序行，将键缓区程序行插入，并修改程序区首指针。

总之，只要键缓区有带行号的输入，程序区总要变化，程序区首指针也要修改。但它总是指向程序区最小行号的那个程序行的首字节的。

四、解释执行

当从键盘输入命令时，转入解释执行阶段。不同的命令有不同的入口。RUN命令的特点是取程序区的程序行来解释执行的。

在解释执行过程中，所有的常量仍采自程序区，所有的变量(包括数组)则要通过访问数据区取得。

1.数据区

在整数BASIC中，变量和数组混合建立在同一数据区中。数型变量除了变量名的长度不同外，数据都是等长的（一个整常数总是由两个字节表示)。数组和字符串变量，都通过DIM语句预留了空间，这就有条件将它们放在一起。

它们在数据区存放的顺序，是以程序运行时出现的顺序为序的。

各种变量（包括数组）在数据区的格式都是一样的。它们分作四栏：

第一栏，名字（变量名，数组名）栏。一个字符占一个字节，名字有几个字符（包括符号“$”）就占用几个字节。都按各字符的ASCII码存放。

第二栏：名字结束标志。由于整数BASIC名字不等长，故需

设此标志。用一个字节，置此字节的内容为0。

第三栏：索引栏。用两个字节给出下一个名字的首地址。它实际上也是一个链指针，只不过这个指针不在本链环之头，但它总是指向下一个链环之首的。

第四栏：数据。数型变量的数据，占用两个字节；数组的每个元素也占用两个字节，全部数组元素，占用的内存空间为：

（最大下标值＋1）×2

顺序总是0按号元素、1号元素，…，依次排列下去；字符串型数据占用的内存空间为指定长度加1，多余的一个字节用以存放一个字符串值的终结符，其代码为$1E。结束标志设置的必要性在于：一个字符串变量名的长度被定义后，仅仅是留出了足够的存贮空间，在后续程序运行中，这个字符串取得的字符个数，只能是不大于给定的长度，但并不一定就等于那个长度。没有一个标志，将不知道这个字符串究竟是由多少个字符组成的。

数据区的格式，请参阅图13—3。

上面提到的那个整数BASIC程序运行后，可以再次查看它们的指针：

4A－00
4B－08
4C—00
4D—96
CA—72
CB—95
CC—45
CD—08

可以看出，除数据尾指针改变了之外（指向了$845），其它指针均未保持运行前的状态。

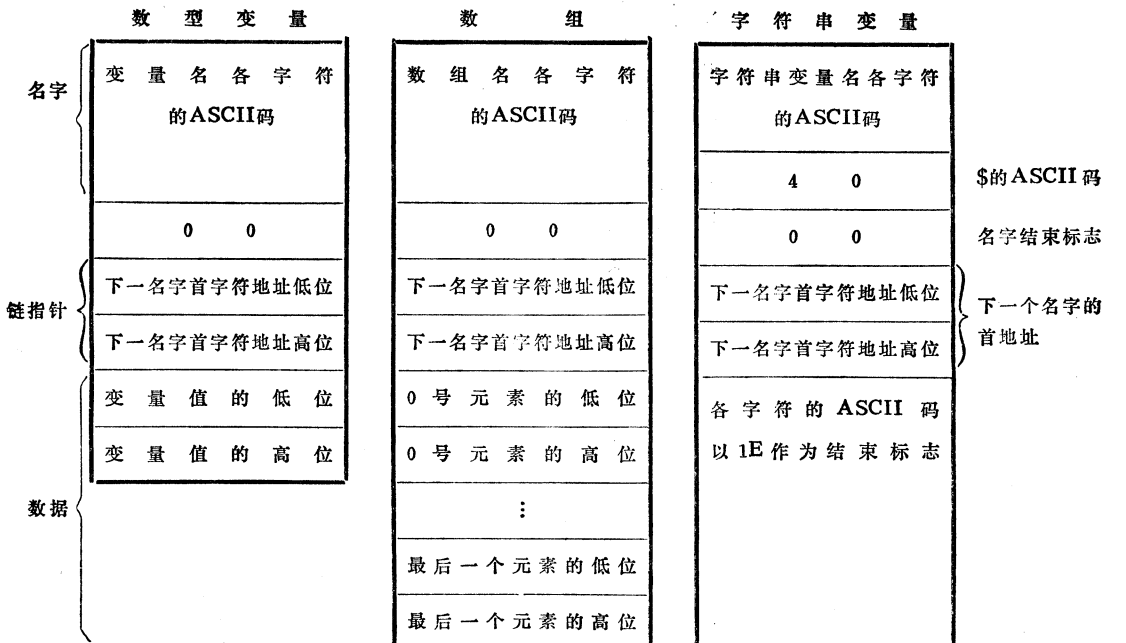

数型变量
数组
字符串变量
名字
变量名各字符的ASCII码
数组名各字符的ASCII码
字符串变量名各字符的ASCII码
4 0
$的ASCII码
0 0
名字结束标志
链指针
下一名字首字符地址低位
下一名字首字符地址高位
下一个名字的首地址
数据
变量值的低位
变量值的高位
0号元素的低位
0号元素的高位
最后一个元素的低位
最后一个元素的高位
各字符的ASCII码以1E作为结束标志

图 13—3

数据的内容如下：

```
800—  C1 C2 C3 C4 00 09 08 05
808—  00 C1 C2 C3 00 11 08 15
810—  00 C1 C2 C3 C4 C5 00 21
818—  08 00 00 01 00 02 00 03
820—  00 C9 00 27 08 04 00 C1
828—  C2 C3 40 00 33 08 C1 C1
830—  1E 20 20 C1 C1 C1 40 00
838—  3F 08 C1 C1 1E 1E 20 D3
840—  00 45 08 03 00
```

其中＄800～＄803，为变量名ABCD，其值为5。第二个名字在＄809～＄80B中，为ABC，其数据为15。第三个名字在＄811～＄815中，为ABCDE，其数据为0，1，2，3。实际上这是一个数组，个数是四个分量…。

2．数据区的建立和使用过程

当一个程序运行时，只要遇到变量名或数组名，总要访问数据区的。它的任务就是从数据区查找相同的名字，以便写入数据或读出数据。

访问数据区的方法很简单。先用数据区头指针给出的地址建立一个扫描指针，检查扫描指针是否和数据区尾指针同值（即指向同一地址）。如果数值不同，则从扫描指针指向的单元开始，逐一取出其中的内容（直到0字节）与程序中的变量名（或数组名）进行比较。如果名字完全一样（包括长度），这个名字就算找到了。把程序中得到的数据写入该名字的相应数据栏内（给这个变量或数组元素赋值时）或取出相应数据给程序中的变量（或数组元素）使用（使用时）。

如果上述比较发现名字不同，则将数据区该名字尾的全0字节之后的内容（链值）送入扫描指针，再查扫描指针是否与数据

区尾指针同值。如果不同值，继续上述的比较。

如果扫描指针与数据区尾指针同值，说明数据区尚无此名字。这时从数据区尾接下去登记它，留出规定空间，修改数据区尾指针值。然后根据需要写入数据或读出数据。

对DIM语句的处理稍有别于其它语句。如果是对某个名字的首次说明，仅仅登记下名字，留出空间、修改数据区尾指针就够了，它没有读、写动作。如果是对一个名字的有别于第一次说明的二次说明，则要当出错处理。

3．值得讨论的几个问题

(1) DIM没有初始化功能

从上面叙述中，不难看出一个现象：DIM只是为一个数组或字符串预留了空间，但并未对这些元素进行初始化。

请看下列程序：

```
10  AB=0
20  DIM A (2)
30  FOR I=0 TO 2
40  A (I)=A (I)+I
50  PRINT A (I),
60  NEXT I
70  AB=AB+1
80  PRINT AB
90  IF AB<3  THEN 20
100 END
```

这个程序的运行结果，有可能是：

```
0       1       2       1
0       2       4       2
0       3       6       3
```

在这个程序中，由于第10行和70行、90行的作用，20行～90可循环执行3次。每执行20行时，虽经DIM为数组A留出了空间，但由

于它没有对数组元素的初始化(清0)作用,所以这些元素被继续使用下去。于是第二次打印出的A(1)是1+1的结果,第三次打印出的A(1)是2+1的结果。A(2)也类似。

我们说,这种输出仅只是可能的,那是因为这个程序在首次执行30行到60行时,相应A(I)中的初始值不一定就是0。这要视当时的具体情况而定。事实上,你只要在30行之间增加几个用到变量或数组名的语句,或减少一些语句(如删去第10行),A(I)被分配的地址就会有变化,它们的初值也就有可能不相同了。

因此,在整数BASIC程序中,不要指望用DIM来对一个数组进行初始化(浮点BASIC的DIM具有这种功能),而应由程序来完成这种工作。

(2) 数组名与数型变量名同名时,将认为是同一名字(数据区数型变量名与数组名格式完全一样)。

一个程序中,对一个已用作数型变量的名字,再用DIM说明它,被认为是二次说明,给出出错信息。

一个程序中,把数组名当作变量名使用,是可以的。这时这个变量取数组的0号元素的数据。

例如程序:

```
10  DIM  A (3)
20  FOR  I=0  TO  3
30  A (I) = I+8
40  PRINT  A (I),
50  NEXT  I
60  PRINT
70  PRINT  A
80  END
```

运行后输出:

```
 8        9        10      11
 8
```

A(0)为8，A也为8

把上列程序改写为：

```
10  DIM  A (3)
20  FOR  I=0  TO 3
30  A (I)=I+8
40  NEXT  I
50  A=A+6
70  FOR  I=0  TO  3
80  PRINT  A (I),
90  NEXT  I
100 PRINT : PRINT A
110 END
```

运行后将输出：

```
14      9     10     11
14
```

可见A(0)取得A的值。也就是说，把数组名当作变量名使用时，这个变量名与数组的0号元素视为同一的。

其原因在于变量的数据为该名字下的第一个（也仅此一个）数据，而数组0号元素，也是该名字下的第一个元素，它们处于同一位置上。

(3) CLR和RUN的清0作用

CLR和RUN都有清除变量为0、数组维数为0和字符串长度为0的作用。实际上它们是把每个名字下的第一个数据清0（对数型变量或数组）或置为1E（对字符串变量）。

正是这个理由，使用CLR或RUN命令之后的数组，除0号元素为0外，其余元素仍留在数据区的数据栏内。

假定数据区初始状态为全0，输入下列程序：

```
10  DIM  A (3)
20  FOR  I=0  TO  3
```

```
30   A (I)=A (I)+5+I
40   PRINT  A (I),
50   NEXT  I
60   PRINT
70   END
```

这时运行这个程序，将输出：

```
5        6        7        8
```

再次敲入RUN命令（二次执行此程序），将输出：

```
5        12       14       16
```

再次敲入RUN命令（第三次运行此程序），将输出：

```
5        18       21       24
```

如果这时敲入 GOTO 10，则输出：

```
10       24       28       32
```

若再敲入GOTO 10，则输出

```
15       30       35       40
```

前三次输出的A(0)均为5，就是RUN清0的结果。而其它输出，每次都不一样，也是因为RUN和GOTO没有全清0功能造成的。

(4) NEW的作用

NEW可以清除当前内存中的BASIC程序。它实际上是把程序区首指针和数据区尾指针改为初值：即将

4C→CA

4D→CB

4A→CC

4B→CD

可见，它只是修改指针，并不改变数据区和程序区的内容。因此，修复一个被NEW破坏了的程序，只要修改程序区首和数据区尾的指针就可以了。寻找程序区的头和数据区的尾，要根据它们各自的结构逐个判断，这需要不断实践才能熟练。

习　题

（一）某人想计算当X＝1，2，3，4时，$X^2+1=$？他写了如下一个整数BASIC程序，指出有什么错误？应如何改写？

```
10  DEF FN A (X) = X↑2 + 1
20  READ A
30  IF A = 0 THEN 70
40  PRINT FNA (A)
50  GOTO 20
60  DATA 1, 2, 3, 4, 0
70  END
```

（二）说明下列程序有什么用途？

```
5   CALL -936 : PRINT : PRINT
10  A = RND (100) + 1
20  B = RND (100) + 1
30  C = RND (4)
40  GOTO 100 + C * 50
100 PRINT A; "+"; B; "="; A + B
110 GOTO 5
150 IF A > B THEN 170
160 T = A : A = B : B = T
170 PRINT A; "-"; B; "="; A - B
180 GOTO 5
200 PRINT A; "*"; B; "="; A * B
210 GOTO 5
250 PRINT A; "/"; B; "="; A/B; "..."; A MOD B
260 GOTO 5
```

如果把150行和160行合写为一个程序行：

```
150 IF A < B THEN T = A : A = B : B = T
```

是否可以？

（三）哥德巴赫猜想，任一偶数都可表示为两个素数之和。请编写一个整数BASIC程序，验证这个猜想对某一范围内（比如100到200之间）的数是正确的，并把两个素数打印出来。

（四）把下列浮点BASIC程序改写成整数BASIC程序：

```
10   A0 = -99999 : A1 = 0
20   B0 = 0 : B1 = 99999
30   READ A
40   IF A = -99999 THEN 400
50   ON SGN (A) + 2 GOSUB 100, 200, 300
60   GOTO 30
100  IF A<A1 THEN A1 = A
110  IF A>A0 THEN A0 = A
120  AA = AA + A
130  RETURN
200  N = N + 1
210  RETURN
300  IF A<B1 THEN B1 = A
310  IF A>B0 THEN B0 = A
320  BB = BB + A
330  RETURN
400  PRINT N, AA, BB,
410  PRINT A0, A1, B0, B1
420  IF -AA>BB THEN PRINT : PRINT "TOTAL<0"
500  DATA 2, -20, 35, -7, -43, 75, …, -99999
600  END
```

（五）把一个已输入机器内的浮点BASIC程序转化成一个整数BASIC程序或把一个整数BASIC程序转化成一个浮点BASIC程序，应采用什么方法，请予指明。

（六）在整数BASIC状态下，通过监控命令查得下列信息：

```
4A- 00 08 00 96
```

```
CA- E3 95 00 08
95E3- 09 0A 00 C1 71
95E8- B1 0A 00 01 09 14 00 C2
95F0- 71 C1 12 C2 01 06 1E 00
95F8- 62 C2 01 05 28 00 51 01
```

请列出这个BASIC程序清单。

（七）有下列BASIC程序：

```
10 DIM A (3)
20 FOR I=0 TO 3
30 A (I)=I
40 NEXT I
50 END
```

请指明程序区的各字节的内容是什么，各指针单元应为何值（按RAM为48K的系统）。

（八）上题程序运行后，数据各单元及各指针单元应为何值？

习题参考答案

第 一 章

(一)

(1) 5

(2) 12

(3) 217

(4) 2918

(二)

(1) 11010

(2) 110110

(3) 10000001

(4) 110001111011000

(三)

(1) F

(2) 11F

(3) 1926

(4) 9E04

(四)

(1) 11

(2) 28

(3) 2707

(4) 65535

(五)

(1) A6

(2) C6

（3） 2CE

（4） 52FA

(六)

（1） 10001111

（2） 1110100100

（3） 110110100110101

（4） 1111111111111111

第 二 章

(一)

"ABC 34"

32.001

(二)

AA

BBC%

(三)

（1） 823450

（2） .0000023456

（3） 1259000000

(四)

（1） PRINT (88－52＊63)/(18＋47＊3)

（2） PRINT (2＊SIN(45＊3.1416/180)↑4)
/(EXP(2)＊LOG(5))

（3） PRINT (6＊2＊3↑2)/(5↑3)

（4） PRINT ABS (SQR (SIN (65＊3.1416/180)↑2＋
COS(25＊3.1416/180)↑2))

(五)

（1） 3＊5＋2＊(68/(2＋3))
↑ ↑ ↑ ↑ ↑
③ ⑤ ④ ② ①

（2）

```
A+B*6/4+8  OR 65+2*8↑2
↑  ↑ ↑ ↑     ↑   ↑ ↑ ↑
⑤  ② ③ ⑥     ⑧   ⑦ ④ ①
```

（3）

```
A+B* SIN (25+4)-85 AND NOT 3
↑  ↑  ↑     ↑  ↑     ↑   ↑
⑤  ④  ②     ①  ⑥     ⑦   ③
```

第 三 章

（一）

（1）一个程序行中若放置两个语句时，应使用语句分隔符予以分隔。逗号不是语句分隔符。

（2）赋值语句的左部必须是变量名，而不得用带运算符的表达式。变量名中不得有运算符号。

（3）缺保留字PRINT。不是一个语句。

（4）INPUT语句不能用逗号结束、那样的逗号无分隔变量名的作用，不符合逗号使用规则。

（5）逗号是多余的，不符合逗号使用规则。

（6）用PRINT语句输出一个字符串常量时，字符串前后的引号对不能省缺。

（7）INPUT保留字之后只能跟变量名不能跟常量（以数字打头的字符串，绝不能作为变量名）。

（二）

```
    × × × × ×
  ×           ×
 (×   O   O   ×)
  ×     V     ×
  ×   <   >   ×
    ×         ×
      × × ×
```

(三)

15, 15　　0

5

(四)

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

(五)

```
10 READ A, B
20 A=A+B
30 PRINT A
40 READ B
50 A=A+B
60 PRINT A
70 READ B
80 A=A+B
90 PRINT A
100 READ B
110 A=A+B
120 PRINT A
130 DATA 12, 43, -54, 76, 15.7
```

(六)

```
10 INPUT R
20 S=3.1416*R*R
30 V=4*3.1416*R↑3/3
40 PRINT "S="; S, "V="; V
50 END
```

第 四 章

(一)

(1) 1

(2) 1

(3) 0

(4) 1

(5) 1

(二)

(1) < > 写成了<>

(2) >= 写成了≥

(3) 保留字THEN之左邻字符为A，使AT构成新的保留字。

(4) ABCD是浮点数变量名，不能与字符串型变量A$比较。

(5) 这行程序会永无休止地执行下去。

(三)

(1)

```
 1
 5
```

(2)

```
 1          2
 4          6
```

(四)

```
10 INPUT "X=? "; X
20 IF X<0 THEN 50
30 Y=(SIN (X)+COS (X))/2
40 GOTO 60
50 Y=(SIN (X)-COS (X))/2
60 PRINT "Y=";Y
70 END
```

(五)

```
10 INPUT "KEYIN THE LENGTH OF THE SIDE OF A
```

```
    TRIANGLE"; A, B, C
 20 P=(A+B+C)/2
 30 T1=P-A : T2=P-B : T3=P-C
 40 IF T1>0 AND T2>0 AND T3>0 THEN 70
 50 PRINT "DATA ERROR"
 60 GOTO 10
 70 S=SQR (P*T1*T2*T3)
 80 PRINT "TRIANGLE AREA: "; S
 90 END
```

(六)

设在DATA语句中，每4个数代表一个学生的情况，它们依次是：学生的学号、主课1考分、主课2考分、副课考分。4个数全为0作终止标志。

程序如下：

```
 10 READ A, B, C, D
 20 IF A=0 THEN 200
 30 IF B+C+D >= 280 THEN 70
 40 IF B >=95 AND C >=95 AND D >=80
    THEN 70
 50 IF (B=100 OR C=100) AND B+C+D >=260
    THEN 70
 60 PRINT A : GOTO 10
 70 PRINT A, B; ", "; C; ", "; D
 80 GOTO 10
 90 DATA……
200 END
```

(七)

```
 10 READ A
 20 IF A=0 THEN 60
 30 IF A<0 THEN PRINT A : GOTO 10
 40 C=C+A
```

```
50  GOTO  10
60  PRINT  C
70  DATA  -2, 3, 15, -86, -7.3, 90, -80, 7, 42, -8, 0
80  END
```

（八）

```
10  A = INT (RND (1) * 21)
20  IF  A = 15 THEN 50
30  PRINT  A
40  GOTO  10
50  END
```

（九）

原因是弄错了运算符的优先顺序。在苹果机上，NOT是单边运算符，＜是关系运算符。由于单边运算符优先于关系运算符，所以对程序20行中的表达式，总是先做NOT X运算，再将其值与5做小于比较。不管X为何值，NOT X的结果为逻辑值，它只能取1或0，而1或0是永远小于5的，因此整个关系式永远为真，于是永远转回第10行。这样不管你输入什么数，也不会打印出来。

修改的办法很简单，把X＜5用括号括起来，这就改变了运算顺序，得到正确的结果。

第五章

（一）

12　　6

（二）

（1）

```
5  A = 1
10  FOR  I = 1  TO  5
30  A = A * I
40  NEXT  I
50  PRINT "5! =", A
```

```
60 END
```

(2)

方法一：

```
20 INPUT "N=? "; N
25 V=1
30 FOR J=1 TO N
35 A=1
40 FOR I=1 TO J
50 A=A*I
70 NEXT I
75 V=V+1/A
80 NEXT J
90 PRINT V
100 END
```

方法二：

```
20 INPUT "N=? "; N
25 V=1 : A=1
40 FOR I=1 TO N
50 A=A*I
60 V=V+1/A
70 NEXT I
90 PRINT V
100 END
```

方法二的基本思想是：已知$n!$，可不必用公式$(n+1)!=1*2*3*\cdots*n*(n+1)$，而用$(n+1)!=n!*(n+1)$。

(三)

```
10 N=0 : B=0
20 FOR I=1 TO 100
30 INPUT A
40 IF A <= 0 THEN 70
```

```
  50  B=B+  SQR (A)
  60  N=N+1
  70  NEXT  I
  80  PRINT  B/N
  90  END
```

(四)

```
  10  FOR  I = 1  TO  397  STEP  2
  20  IF  I*(I+2)=399 THEN 60
  30  NEXT  I
  40  PRINT "NO! "
  50  GOTO  70
  60  PRINT  I, I+2
  70  END
```

(五)

```
  10  INPUT "M, N" ; M, N
  20  IF  M*4 >=N  AND M*2 <  =  N  AND
      INT (N/2)  =  N/2  THEN  50
  30  PRINT "DATA ERROR"
  40  FOR I  =  1 TO 100 : NEXT : GOTO 10
  50  FOR  I  =  0  TO  M
  60  IF  I*2+  (M-I)*4  =  N THEN 80
  70  NEXT  I
  80  PRINT  I, M-I
  90  END
```

(六)

```
  10  FOR  I=0  TO  100
  20  FOR  J  =  0  TO  100-I
  30  K=100-I-J
  40  IF  I*3+J*2+K/2 =100  THEN 70
  50  NEXT  J, I
```

```
  60 PRINT "NO" : END
  70 PRINT I, J, K
  80 END
```

(七)

```
  10 S=1
  20 FOR I=1 TO 19
  30 S= S+ 1/ (I*(I+1))
  40 IF 1/ (I*(I+1)) < 1E-3 THEN 60
  50 NEXT
  60 PRINT I, 1/ (I*(I+1)), S
  70 END
```

(八)

```
  10 INPUT M, N
  20 IF M<N THEN K=M : GOTO 40
  30 K=N
  40 FOR I=K TO 2 STEP -1
  50 IF INT (M/I) =M/I AND INT (N/I) =N/I THEN 70
  60 NEXT I
  70 PRINT I
  80 END
```

(九)

```
  10 T=0
  20 FOR I=0 TO 100 STEP 50
  30 FOR J=0 TO 100 STEP 20
  40 A=I+J
  50 FOR K=0 TO 100 STEP 10
  60 B=A+K
  70 FOR L=0 TO 100 STEP 5
  80 C=B+L
  90 FOR M=0 TO 100 STEP 2
```

```
100 D=C+M
110  FOR N=D TO 100
120 IF N=100 THEN T=T+1 : GOTO 140
130  NEXT  N
140  NEXT M, L, K, J, I
150 PRINT  T
160 END
```

(注意，此题上机运行时间较长。最后输出为4562)

第 六 章

(一)

执行顺序为*10 *100 *110 *120 *130 *20 *200 *210 *220 *230 *30 *300 *310 *320 *330 *40

运行结果输出

```
 35
 750
 .875
```

(二)

```
10  S=0
20  FOR I=1 TO 3
30  GOSUB  100
40  NEXT
50  PRINT S
60  END
100 READ A, B, C
110 P= (A+B+C) / 2
120 S1=SQR (P*(P-A)*(P-B)*(P-C))
130 S=S+S1
140 RETURN
200 DATA…
```

(三)

```
10  GOSUB  100
20  A=P
30  GOSUB  100
40  B=P
50  GOSUB 100
60  C=P
70  S=A+B+C
80  PRINT "=" ; S
90  END
100 P=1
110 READ M
120 FOR  I=1 TO M
130 P=P*I
140  NEXT
150 PRINT "+" ; M;  "! " ;
160 RETURN
200 DATA 3, 4, 5
RUN
 +3!+4!+5!=150
```

(四)

先打印10个数，它们的值只能是从1到7中的任一整数。

然后再打印10个数，全部为11。

(五)

```
10  M=0:S=0:N=0:K=0:L=-9999999:R=L
20  FOR I=1 TO 10000
30  ONERR GOTO 500
40  INPUT A
50  IF A=R THEN 90
60  T=SGN (A) +2
```

```
70  ON T GOSUB 200, 300, 400
80  NEXT
90  PRINTTN
100 PRINT S/M
110 PRINT K, L
120 END
200 IF K>A THEN K=A
210 IF L<A THEN L=A
220 RETURN
300 N=N+1
310 RETURN
400 S=S+A : M=M+1
410 RETURN
500 PRINT "DATA ERROR"
510 INPUT A
520 RESUME
```

（六） 这个程序运行时，执行的行号顺序是：

*10 *100 *110 *200 *210 *220 *100 *110 *200 *210 *220 *100…

这个程序将无限重复下去，成为死循环。

运行时输出为：

0 5 0 5 0 5 ……

（七） 这个程序将输出4的阶乘24。执行过程及栈状态，如图6—7所示：

（八） 本程序运行时，可以让用户在机器作加减乘除练习。究竟做那一种运算，由用户自选，这种运算做多少次，也由用户指定。参加运算的数据由机器随机产生。一个题目显示出来后，用户给出答案，如果回答对了，计算机自动给你记分（每答对一次给1分），回答不对，不给分。给定次数做完后，显示你的最后得分。如果你还想再作练习，敲入YES，否则敲入别的字母。

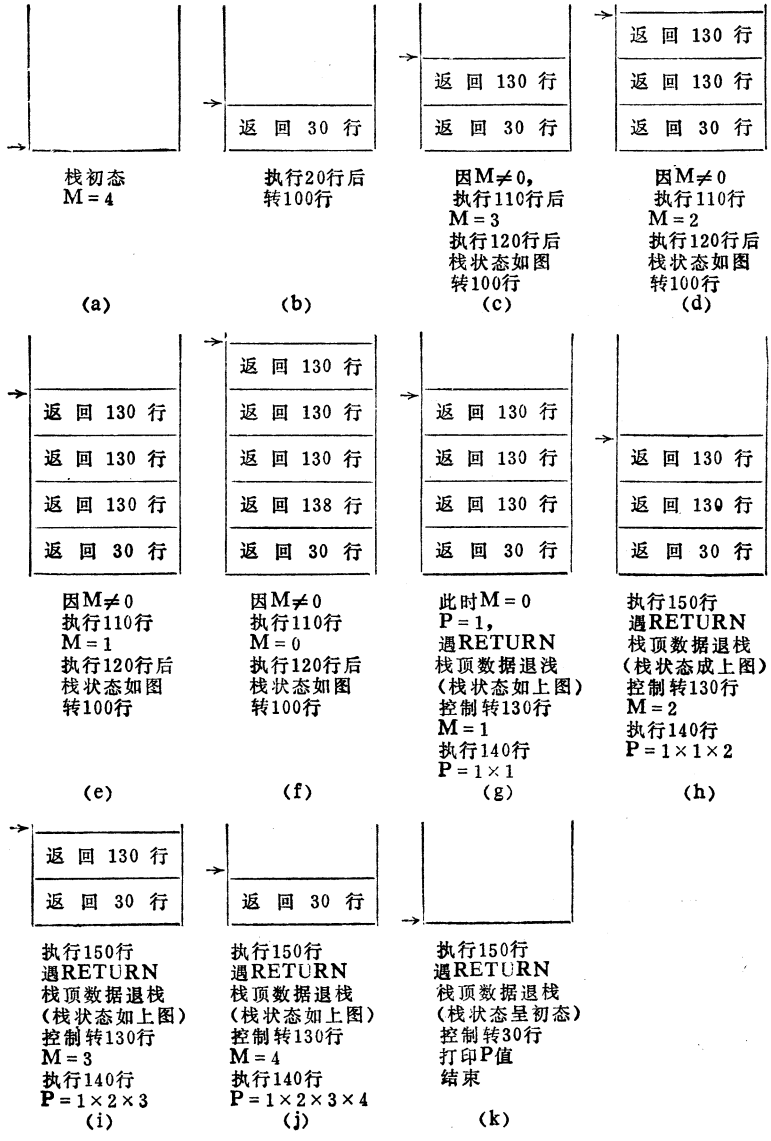

栈初态
M = 4
(a)
返 回 30 行
执行20行后
转100行
(b)
返 回 130 行
返 回 30 行
因M≠0,
执行110行后
M = 3
执行120行后
栈状态如图
转100行
(c)
返 回 130 行
返 回 130 行
返 回 30 行
因M≠0
执行110行
M = 2
执行120行后
栈状态如图
转100行
(d)
返 回 130 行
返 回 130 行
返 回 130 行
返 回 30 行
因M≠0
执行110行
M = 1
执行120行后
栈状态如图
转100行
(e)
返 回 130 行
返 回 130 行
返 回 130 行
返 回 138 行
返 回 30 行
因M≠0
执行110行
M = 0
执行120行后
栈状态如图
转100行
(f)
返 回 130 行
返 回 130 行
返 回 130 行
返 回 130 行
返 回 30 行
此时M = 0
P = 1,
遇RETURN
栈顶数据退浅
(栈状态如上图)
控制转130行
M = 1
执行140行
P = 1 × 1
(g)
返 回 130 行
返 回 130 行
返 回 130 行
返 回 30 行
执行150行
遇RETURN
栈顶数据退栈
(栈状态成上图)
控制转130行
M = 2
执行140行
P = 1 × 1 × 2
(h)
返 回 130 行
返 回 30 行
执行150行
遇RETURN
栈顶数据退栈
(栈状态如上图)
控制转130行
M = 3
执行140行
P = 1 × 2 × 3
(i)
返 回 30 行
执行150行
遇RETURN
栈顶数据退栈
(栈状态如上图)
控制转130行
M = 4
执行140行
P = 1 × 2 × 3 × 4
(j)
执行150行
遇RETURN
栈顶数据退栈
(栈状态呈初态)
控制转30行
打印P值
结束
(k)

图 6—7

第 七 章

(一)

```
10 READ N,
20 DIM A (N, N)
30 FOR I=1 TO N
40 FOR J=1 TO N
50 READ A (I, J)
60 NEXT J, I
70 FOR I=1 TO N-1
80 FOR J=I+1 TO N
90 B=A (J, I) : A (J, I) =A (I, J) : A (I, J) =B
100 NEXT J, I
110 FOR I=1 TO N
120 FOR J= 1 TO N
130 PRINT A (I, J),
140 NEXT J
150 PRINT
160 NEXT I
170 END
180 DATA 3, 1, 2, 3, 4, 5, 6, 7, 8, 9
```

(二)

```
10 READ N, M
20 DIM A (N, M), B (N, M), C (N, M)
30 FOR I = 1 TO N
40 FOR J = 1 TO M
50 READ A (I, J)
60 NEXT J, I
70 FOR I = 1 TO N
80 FOR J=1 TO M
```

```
 90  READ B (I, J)
 100 NEXT J, I
 110 FOR I = 1 TO N
 120 FOR J = 1 TO M
 130 C (I, J) =A(I, J) +B(I, J)
 140 PRINT C (I, J),
 150 NEXT J
 160 PRINT
 170 NEXT I
 200 DATA 2, 3
 210 DATA 12, 18, -50, -7, 325, 60
 220 DATA 58, -30, 6, 18, 72, -9
 300 END
```

(三)

```
 10  READ N
 20  DIM A (N) , B (N) , C (N)
 30  T1=0 : T2=0 : G=0 : J=0
 40  FOR I=1 TO N
 50  B (I) =0 : C (I) =0
 60  READ A (I)
 70  NEXT
 80  FOR I = 1 TO N
 90  IF INT (A(I)/2) * 2=A(I)/2 THEN 140
 100 G=G+1
 110 B (G) =A (I)
 120 T1=T1+A (I)
 130 GOTO 170
 140 J=J+1
 150 C (J) =A (I)
 160 T2=T2+A (I)
```

```
170 NEXT I
180 M=G
190 PRINT "ODD; "
200 GOSUB 400
210 M=J
220 FOR I=1 TO M
230 B (I) =C (I)
240 NEXT I
250 PRINT "EVEN; "
260 GOSUB 400
270 PRINT "THE TOTAL OF ALL ODD NUMBERS IS";
    T1
280 PRINT "THE TOTAL OF ALL EVEN UNMBERS IS";
    T2
290 END
400 FOR I=1 TO M-1
410 FOR L=I+1 TO M
420 IF B(I)<=B(L) THEN 460
430 Y=B(I)
440 B(I)=B(L)
450 B(L)=Y
460 NEXT L
470 PRINT B(I),
480 NEXT I
490 PRINT B(M)
500 PRINT
510 RETURN
600 DATA……
```

（四）这是一个查找职工工资的程序。DATA语句只记录了六个人的情况。

程序运行时情况如下：带下横的内容由键盘敲入：

```
TYPE NAME TO FIND PAY? ZHANG SI HAI↙
ZHANG SI HAI'S PAY IS 66 YUN
TYPE NAME TO FIND PAY? ZHAO YAN↙
ZHAO YAN'S PAY IS 98 YUN
TYPE NAME TO FIND PAY? JIA KUN↙
CHN'T FIND NAME
TYPE NAME TO FIND PAY? ? + - ↙

TYPE NAME TO FIND PAY? 100↙
TOTAL:585        MEAN:97.5
TYPE NAME TO FIND PAY? 03↙
BYE!
```

(五) 增加如下一些程序行：

```
301  INPUT "ENTER 1 OR 2 OR 3"; Q
302  IF Q<1 OR Q>3 THEN 301
303  ON Q GOTO 310, 330, 350
330  P=LEN (A$)
332  FOR I=1 TO K
334  IF A$ < > LEFT$(A$(C(I), 1), P)
     THEN 340
336  C(L+1)=C(I)
338  L=L+1
340  NEXT
342  K=L : L=0
344  RETURN
350  P=LEN (A$)
352  FOR I=1 TO K
354  IF A$< >RIGHT$(A$(C(I), 1), P) THEN 360
356  C(L+1)=C(I)
```

```
358  L=L+1
360  NEXT
362  K=L : L=0
364  RETURN
```

(六)

```
10   READ N, E
20   DIM A(N, N), B(N), C(N)
30   FOR I=1 TO N
40   FOR J=1 TO N
50   READ A(I, J)
60   NEXT J, I
70   FOR I=1 TO N
80   READ B(I)
90   C(I)=I
100  NEXT I
110  FOR I=1 TO N-1
120  C=ABS (A(I, I)) : L=I
130  FOR J=I+1 TO N
140  IF ABS (A(I, J))>C THEN C=ABS (A(I, J))
150  NEXT J
160  FOR J=1 TO N
170  T=A(J, L) : A(J, L)=A(J, I) : A(J, I)=T : L=J
180  NEXT J
190  T=C(I) : C(I)=C(L) : C(L)=T
200  IF C<E THEN STOP
210  C=1/C
220  FOR J=I TO N
230  A(I, J)=A(I, J)*C
240  NEXT J
250  B(I)=B(I)*C
```

```
260  FOR  J=I+1  TO  N
270  T=A(J, I)
280  FOR  K=I  TO  N
290  A(J, K)=A(J, K)-A(I, K)*T
300  NEXT  K
310  B(J)=B(J)-B(I)*T
320  NEXT  J,I
330  B(N)=B(N)/A(N, N) : A(N, N)=1
340  FOR  I=N-1  TO  1  STEP  -1
350  FOR  J=I+1  TO  N
360  B(I)=B(I)-A(I, J)*B(J)
380  NEXT  J, I
390  FOR  I=1  TO  N
400  A(0, C(I))=B(I)
410  NEXT  I
420  FOR  I=1  TO  N
430  B(I)=A(0, I)
440  PRINT "X"; I; "="; B(I)
450  NEXT  I
```

第 八 章

(一)

```
10  HOME
20  PRINT : PRINT TAB (20); "*"
30  FOR  I=1  TO 4
40  PRINT  TAB (20-I); "*"; TAB (20+I); "*"
50  NEXT  I
60  PRINT  TAB(15); "* * * * * * * * * * *"
70  END
```

(二)

```
10  FOR X=0 TO 2*3.1416 STEP 3.1416/18
20  Y1=SIN(X)
30  Y2=2*SIN(X+3.1416/6)
40  IF INT (10*Y1)=INT (10*Y2) THEN 90
50  IF INT (10*Y1)<INT (10*Y2) THEN 80
60  PRINT TAB ((25+10*Y2)/1.5); "!"; TAB ((25+10*
    Y1)/1.5); "*"
70  GOTO 100
80  PRINT TAB ((25+10*Y1)/1.5); "*";
90  PRINT TAB ((25+10*Y2)/1.5); "!"
100 NEXT
110 END
```

(三)

```
10  PRINT TAB(16); "CALENDAR"
20  PRINT : PRINT
30  B=4
40  FOR I=1 TO 12
50  GOSUB 210
60  READ N
70  FOR J=1 TO N
80  PRINT TAB (4+(B-1)*5); J;
90  B=B+1
100 IF B>7 THEN B=1 : PRINT
110 NEXT J
120 IF B< >1 THEN PRINT
130 GOSUB 230
140 PRINT : PRINT
150 NEXT I
160 PRINT
```

```
170 END
210 PRINT TAB(15); "* * "; I; "* * "
220 PRINT TAB(3); "SUN MON TUE WED THU FRI
    SAT"
230 PRINT "  ";
240 FOR J=1 TO 36
250 PRINT "-";
260 NEXT J
270 PRINT
280 RETURN
300 DATA 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31
```

（四）这个程序运行时，好象从屏幕底部的右端出现一只向左移动的蓝色杯子，当移到屏幕中间时不再移动了。接着好象往杯子里加入桔子水(桔红色)，直到装满杯子为止。

（五）程序运行时，会在屏幕产生低清晰度彩色图案，这个图案在不断改变某些部分的颜色，它的改变规律具有上下左右的对称性。

（六）

```
10  DIMX(90), Y(90)
20  FOR I=0 TO 90
30  C=4*I*3.1416/180
40  X=90*COS (C)
50  Y=40*SIN (C)
60  X(I)=X+139
70  Y(I)=Y+91
80  NEXT I
90  HGR 2
100 HCOLOR=3
110 FOR I=0 TO 89
120 HPLOT X(I), Y(I) TO X(I+1), Y(I+1)
130 NEXT I
```

```
140  END
```

（七）

```
10   S=24576
20   FOR  I=0  TO  15
30   READ  A
40   POKE  S+I, A
50   NEXT  I
60   DATA  1, 0, 4, 0, 45, 53, 54, 63, 39, 36, 36,
     44, 45, 54, 6, 0
70   POKE  232, 0: POKE  233, 96
80   HGR2
90   HCOLOR=3
100  FOR  I=1  TO  30
110  ROT=I* 2
120  SCALE  I
130  DRAW 1 AT  100, 40
140  FOR  J=1  TO  100 : NEXT  J
150  XDRAW 1 AT  100, 40
160  NEXT  I
170  ROT=0
180  DRAW 1 AT  100, 40
190  END
```

（八）程序运行时，先在屏上出现一个高清晰度的蓝色菱形，随后又出一个16×16的方格图，它几乎占满了屏幕。接着出现一幅低清晰度图形：屏的底色为浅绿色，中间出一个粉红色三角形。然后上述三幅图形又依次出现一次。最后屏上出现一片字符L，中间一块三角形区内为别的字符。

第 九 章

（一）

```
10   FOR  I=770  TO  790
```

```
20  READ A
30  POKE I, A
40  NEXT I
50  READ I, J
60  IF J=0 THEN END
70  POKE 768, I : POKE 769, J
80  CALL 770
90  GOTO 50
100 DATA 173, 48, 192, 136, 208, 5, 206, 1, 3, 240,
9, 202, 208, 245, 174, 0, 3, 76, 2, 3, 96
110 DATA 172, 70, 192, 160, 192, 30, 172, 30, 152, 110,
128, 30, 152, 70, 228, 70
120 DATA 172, 70, 192, 160, 192, 30, 172, 30, 152, 70,
192, 30, 172, 30, 152, 70, 228, 70
130 DATA 152, 30, 128, 30, 114, 160, 96, 70, 114, 160,
128, 70, 152, 70
140 DATA 114, 70, 152, 160, 172, 30, 152, 30, 192, 160,
192, 30, 172, 30, 152, 30, 128, 30
150 DATA 152, 160, 172, 70, 192, 70, 228, 70, 192,
160, 228, 30, 192, 30, 228, 160, 228, 160
160 DATA 0, 0
```

（二）这个程序可以辅助你谱写一首曲子。但程序中缺发音的机器语言子程序。没有这个子程序非但发不出声音来，而且可能运行不下去。

下面假定发音的子程序已送入内存，进一步说明一下程序功能。

程序供用户三个选择输入项：输入歌谱信息、完整地唱一遍你谱写的曲子和打印歌谱信息：

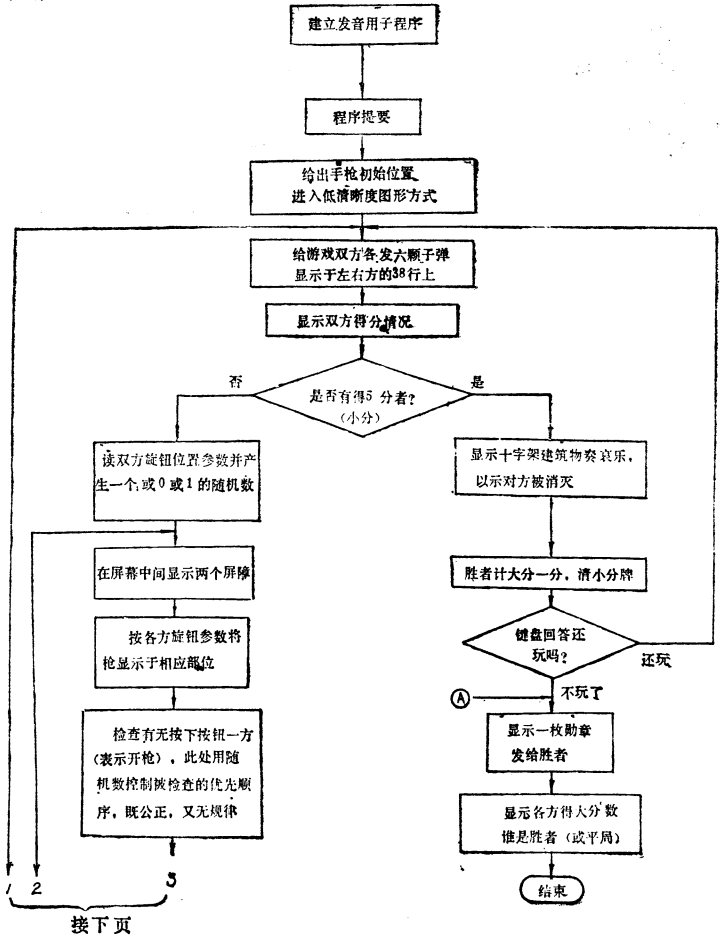
1。选择 E(NTER) 项时，将按你输入的每对歌谱信息发出相应的声音来，并把它们存入A%数组中（这个数组能放501对歌谱信息）。歌谱信息输入完后，可敲入两个 0 作为结束标志。这时机器问你要修改那一对信息，如有需修改的，先给出序号，再输入正确的歌谱信息，这样一直可修改到

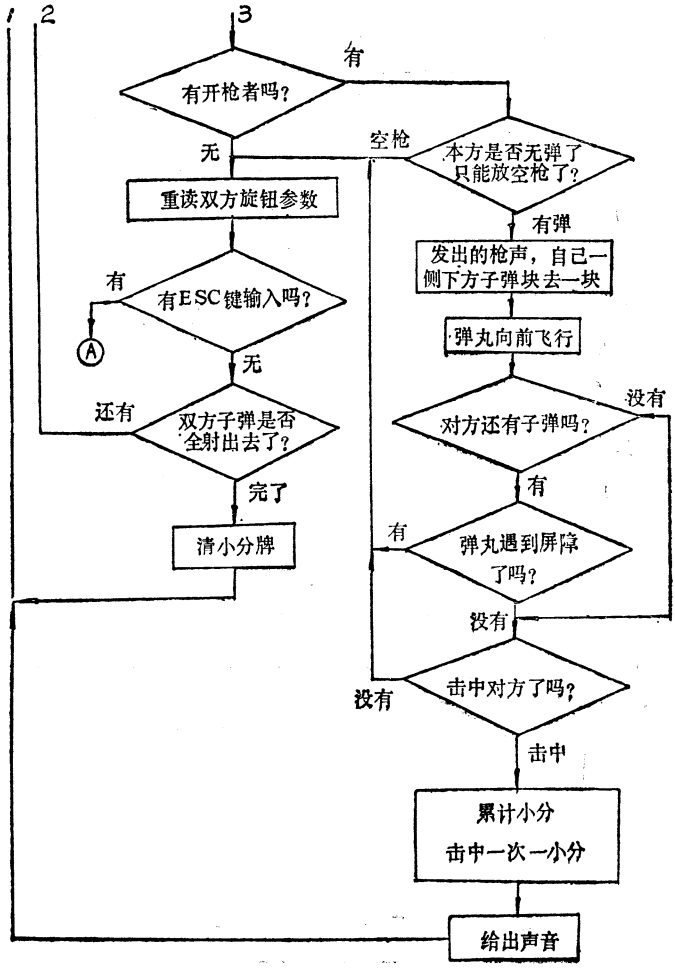
没错。修改时也会按你输入的信息发出声音。到不需要修改时，输入一个0，机器退出这一选择项，你可再选用别的项。

2．选择L(ISTEN)项时，可将你刚刚编写的曲子完整地唱一遍，然后返回让你重新选择别的项。没有选用过E，这一项实现不了。

3．选择P(RINT)项时，可将你的歌谱信息全部依序打印出来，然后返回让你再选别项。没有选过E，这一项也实现不了。

（三）





第 十 章

(一) BASIC中的SAVE和LOAD命令只能和磁带机发生关系，它们都不带程序名。DOS中的SAVE和 LOAD 命令是和磁盘打交道的，这两个命令都得给出程序名。BASIC中RUN命令，只能运行内存中的当前程序。保

留字RUN之后可以带一个行号，指明从给定的行号开始运行那个程序。但RUN之后不能带程序名。而DOS中的RUN命令必须带一个程序名，指明运行磁盘上的哪个程序。

（二）顺序文本文件是把程序中送来的字符按其ASCII码一个挨一个紧密排下去的，文件从头到尾没有空白字节，因此占用磁盘空间较小，这是它的优点。但要想修改或使用其中某一字段或某一记录的内容时，较麻烦些，这是其缺点。随机文本文件的记录长度是固定的，因此修改或使用其中某一记录非常方便。但随机文件中一个记录和下一个记录之间可能会留下一些空白，因此它占用较大的磁盘空间。

根据两种文件存放格式上的特点，一个文件中的信息需随机地修改或使用其中某一指定记录时，最好使用随机文件，以便较容易地将读写指针指向它；否则尽量使用顺序文件，以免浪费磁盘空间。

（三）

该程序运行，屏上先后出现：

```
YOU GET TO TYPE ONE STRING AT A TIME
A STRING MAY HAVE UP TO 239 CHARACTERS
THIS PROGRAM LETS YOU WRITE TEXT FILES

(PRESS THE RETURN KEY TO QUIT.)
TYPE STRING #1: HERE'S STRING 1
(PRESS THE RETURN KEY TO QUIT.)
TYPE STRING #2: AND MY SECOND STRING
(PRESS THE RETURN KEY TO QUIT.)
TYPE STRING #3: ON WE GO
(PRESS THE RETURN KEY TO QUIT.)
TYPE STRING #4: ENOUGH ALREADY!
(PRESS THE RETURN KEY TO QUIT.)
TYPE STRING #5:

WHAT FILE NAME? TEST
```

```
OPEN TEST
DELETE TEST
OPEN TEST
WRITE TEST
4
HERE'S STRING 1
AND MY SECOND STRING
ON WE GO
ENOUGH ALREADY!
CLOSE TEST
```

这个程序运行后，将在当前盘上形成一个名为TEST的顺序文本文件。

这个文件的ASCII码对应字符序列是：

4↙HERE'S STRING 1↙ AND MY SECOND STRING ↙ON WE GO↙ ENOUGH ALREADY! ↙

(四)

该程序运行时，可在屏幕上看到：

```
THIS PROGRAM RETRIEVES TEXT FILES
CREATED BY THE 'MAKE TEXT' PROGRAM
MON I IS IN EFFEAT

NAME OF TEXT FILE? TEST

? 4
? HERE'S STRING 1
? AND MY SECOND STRING
? ON WE GO
? ENOUGH ALREADY!
```

(五)

这个程序运行时，可以在当前磁盘上建立一个磁盘顺序文本文件，名为CODE-POKES

程序中的变量COUNTER，在循环体内可以用来统计循环次数，而它的有效值为1到9。当它为1时，首先遇到一个不带输出项的PRINT语句(110行)，产生回车换行动作。由于10、40、50行的作用，它实际向CODE－POKES文件送去一个回车符。再由120行向文件送出 LINENUMBER的值(初值为10，经130行后可变为20)。循环到第10次，COUNTER的值再次变为1，再向文件送去一个回车符和一个LINENUMBER的新值。其间经140行向文件送去九组数据。因PLACE是循环变量，它的取值依次为770，771，…，790，而770到790号单元中是发音用的机器语言子程序，因而在循环体内，PEEK(PLACE)的取值将是十进制数173，48，192，…，就是那个子程序序列的十进制表示。而140行输出的POKE，逗号，冒号，都是字符串常量。因此送入文件的将是：

↙

```
10  POKE 770, 173 : POKE 771, 48 : POKE 772, 192 :
…… : POKE 778, 3↙
20  POKE 779, 240 : …
……
```

由于是顺序文件，它们一个挨着一个紧密地排列起来。它们是一些由POKE语句构成的一个个程序行。

执行EXEC CODE－POKES后，这些程序像从键盘上输入一样，成了若干行由POKE语句组成的BASIC程序。

这时使用RUN命令，唱歌用子程序又可建立起来。

这里又看到了命令EXEC命令的功能：可以把机器语言程序（其实不只是程序）变成了用POKE语句组成的BASIC程序。

（六）

```
10  D$ = CHR$ (4)
20  PRINT D$; "OPEN RA－FILE, L30"
30  FOR J = 12 TO 15
40  PRINT D$; "READ RA－FILE, R"; J
50  INPUT A$
60  IF LEFT$ (A$, 3) = "DOS" THEN PRINT "RECO
```

```
      RD";   J; "WAS  CHANGED。"
  70  NEXT  J
  80  PRINT  D$; "CLOSE  RA-FILE"
```

(七)

你可以在他原来的程序头部和尾部加上如下的程序行：

```
  10  DIM  A(49)
  20  D$ = CHR$( 4 )
  30  PRINT  D$; "OPEN  DATAFILE"
  40  PRINT  D$; "READ  DATAFILE"
  50  FOR  I=0  TO  49
  60  PRINT  A(I)
  70  NEXT  I
  80  PRINT  D$; "CLOSE  DATAFILE"
 10010  PRINT  D$; "OPEN  DATAFILE"
 10020  PRINT  D$; "DELETE  DATAFILE"
 10030  PRINT  D$; "OPEN  DATAFILE"
 10040  PRINT  D$; "WRITE  DATAFILE"
 10050  FOR  I=0  TO  49
 10060  PRINT  A(I)
 10070  NEXT  I
 10080  PRINT  D$; "CLOSE  DATAFILE"
```

在处理首批数据时，可先执行立即执行命令：

```
DIM  A(49)
RUN  100
```

处理以后的数据时，直接用RUN命令即可。

另外在100到 10000 行的程序中不得再有对A数组的说明语句，也不得再有使A数组初始化的语句。

第十一章

(一)

```
300-A2  00
```

```
302-A9  00
304-9D  00  20
307-E8
308-D0  FA
30A-EE  06  03
30D-AD  06  03
310  C9  40
312  D0  EE
314  00
```

(二)

```
3000-38
3001-48
3002-E9  01
3004-D0  FC
3006-68
3007-E9  01
3009-D0  F6
300B-60
```

这个程序是个延时子程序。它可依转入此子程序时累加器 A 中数值大小，使延迟较长或较短的时间。

(三) 首地址为$F273

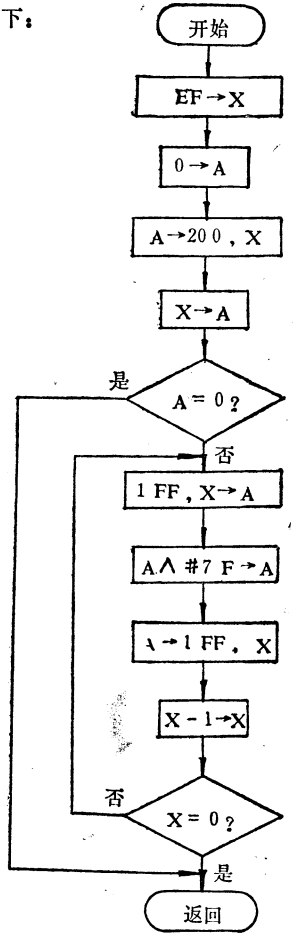
```
LDA  #$FF
BNE  $F279
LDA  #$3F
LDX  #$00
STA  $32
STX  $F3
RTS
```

当在监控中用命令F273G时，将把$FF(即十进制255) 送入$32 (即十进制50) 号单元。相当于执行BASIC命令NORMAL，使以后的屏幕为正常

方式(黑底白字)。在监控中用F277G命令时,则相当于使用BASIC命令 INVERSE, 使屏幕为反方式(白底黑字)。

(四)

这段程序框图如下:



这段子程序的功能是, 根据$EF号单元中数确定一个长度(字节数), 将$200号单元开始的那么多个连续单元中的数, 最高位清0, 后六位不变。当$EF中的数为0时, 只往$200中送0。

（五）

```
306-18              CLC
307-AD  00  03      LDA  $0300
30A-6D  02  03      ADC  $0302
30D-8D  04  03      STA  $0304
310-AD  01  03      LDA  $0301
313-6D  03  03      ADC  $0303
316-8D  05  03      STA  $0305
319-00
```

（六）两种方法都是错误的。无论那种方法，都是先从内存单元号码小的地址移起。当把$2000～$20FF移入$2100～$21FF之后，$2100～$21FF中原来的内容被冲掉了。所以这样再移动的结果，是用$2000～$20FF中的内容重复地被移入以后的内存区了。

对第1种方法，可作如下改动：

```
306-A9  2F      LDA  *$2F
30A-A9  30      LDA  *$30
315-C6  29      DEC  $29
317-C6  2B      DEC  $2B
31B-C9  1F      CMP  *$1F
```

实际上只需修改$307、$30B、$315、$317和$31C五个单元的内容，其余均可不变。

对第二种方法，可改用下列命令：

```
]CALL-151↙
*3100<2000.2FFFM↙
*2100<3100.40FFM↙
*3D0G↙
]
```

（七）

A为0：　$28中为$0
　　$29中为$4

A为1：　$28中为$80

　　$29中为$4

A为2：　$28中为$0

　　$29中为$5

A为3：　$28中为$80

　　$29中为$5

A为4：　$28中为$0

　　$29中为$6

第十二章

(一)

程序区内容：

800- 00 0C 08 0A 00 81 49 D0

808- 31 C1 33 00 17 08 14 00

810- 41 28 49 29 D0 49 00 1D

818- 08 1E 00 82 00 23 08 28

820- 00 80 00 00 00

各指针单元的内容

67- 01

68- 08 26 08 26 08 26 08

AF- 26

B0- 08

(二)

将$801号单元置为$0A，将$802号单元置为$08，这个程序即可用LIST命令显示出来。但要运行或修改(包括增、删)，需将$AF和$69号单元改为$24，将$B0和$6A号单元改为$08。

直接读此程序区，可知此程序为：

```
10  A=10
20  B=A*A
30  PRINT  B
```

```
40  END
```

（三）这个机器语言子程序投入运行时，可以把程序区的首链头接好，并能找到程序区的尾，同时将程序尾、变量首、数组首、尾指针也恢复好。因此，它是一个修复被NEW破坏了的BASIC程序的程序。

若把这个子程序以RESCUE为名存入磁盘，可直接使用BRUN RESCUE命令完成修复工作。

（四）

改写后的程序如下：

```
300- A2 05        LDX  #$05
302- BD 00 08     LDA  $0800, X
305- F0 03        BEQ  $030A
307— E8           INX
308- D0 F8        BNE  $0302
30A- 18           CLC
30B- 8A           TXA
30C- 69 01        ADC  #$01
30E- 8D 01 08     STA  $0801
311- 8D 1F 03     STA  $031F
314- A9 08        LDA  #$08
316- 8D 02 08     STA  $0802
319- 8D 20 03     STA  $0320
31C- A2 00        LDX  #$00
31E- BD 00 00     LDA  $0000, X
321- 95 2A        STA  $2A, X
323- D0 07        BNE  $032C
325- E0 01        CPX  #$01
327- F0 14        BEQ  $033D
329- E8           INX
32A- D0 F2        BNE  $031E
32C- E0 01        CPX  #$01
32E- D0 F9        BNE  $0329
```

```
330- A5 2A        LDA $2A
332- 8D 1F 03     STA $031F
335- A5 2B        LDA $2B
337- 8D 20 03     STA $0320
33A- 4C 1C 03     JMP $031C
33D- 18           CLC
33E- AD 1F 03     LDA $031F
341- 69 02        ADC #$02
343- 85 69        STA $69
345- 85 6B        STA $6B
347- 85 6D        STA $6D
349- 85 AF        STA $AF
34B- AD 20 03     LDA $0320
34E- 69 00        ADC #$00
350- 85 6A        STA $6A
352- 85 6C        STA $6C
354- 85 6E        STA $6E
356- 85 B0        STA $B0
358- 60           RTS
```

（五）在输入程序之前使用HIMEM：命令或LOMEM：命令，同在程序一开始使用HIMEM：命令或LOMEM命令，效果完全一样。

在程序中部使用HIMEM：命令，会使此前的各字符串变量的值失效（除程序中具有字符串变量名="…"形式的字符串变量外）。程序中部使用LO MEM：命令，会使此前的变量或数组失效。

在程序尾部使用HIMEM：或LOMEM：命令，没有任何效果。只能对下次运行（或下一个程序运行）产生作用。

第十三章

（一）在整数BASIC中，没有自定义函数功能，也没有READ/DATA语句，所以程序的第10、20、40行都无法执行。

程序可改写如下：

```
10  INPUT  A
20  IFA=0  THEN  50
30  PRINT  A↑2+1
40  GOTO  10
50  END
```

或改写成：

```
10  A=1 : B=2 : C=3 : D=4 : E=5
20  IF  A=0  THEN  50
30  PRINT  A↑2+1
40  A=B : B=C : C=D : D=E  : 1=0
50  GOTO  20
60  END
```

（二）程序可用来作加减乘除练习，两个操数都在100以内。作减法时保证被减数大于减数(因而不会产生负的结果)。

把150行和160改为另附的150行那样，是不行的。因为无论A与B为何种关系，A=B：B=T两个语句总要被执行，这在A>B的情况下，被减数将不是这次随机产生的，弄不好会使A-B产生负结果。

（三）

```
10   N=100
20   M=200
30   FOR  I=N  TO  M  STEP  2
40   U=1
50   U=U+1
60   W=U
70   GOSUB  200
80   IF  P=0  THEN  50
90   V=I-U
100  W=V
110  GOSUB  200
120  IF  P=0  THEN  50
130  PRINT  I; "="; U; "+"; V
```

```
140 NEXT I
150 END
200 P=0
210 FOR J=2 TO W-1
220 IF (W MOD J)=0 THEN 250
230 NEXT J
240 P=1
250 RETURN
```

(四)

```
10  A0=-32767 : A1=0
20  B0=0 : B1=32767
30  INPUT A
40  IF A=-32767 THEN 400
50  GOSUB (SGN (A)+2)*100
60  GOTO 30
100 IF A<A1 THEN A1=A
110 IF A>A0 THEN A0=A
120 AA=AA+A
130 RETURN
200 N=N+1
210 RETURN
300 IF A<B1 THEN B1=A
310 IF A>B0 THEN B0=A
320 BB=BB+A
330 RETURN
400 PRINT N, AA, BB
410 PRINT A0, A1, B0, B1
420 IF -AA<=BB THEN 600
430 PRINT : PRINT "TOTAL<0"
600 END
```

（五）现就把一个浮点BASIC程序转化成一个整数BASIC程序（相反的转化方法是类似的）说明如下：

首先将浮点BASIC程序中不符合整数 BASIC 程序规则的那些语句加以适当修改或暂行删去。然后可用下列方法之一：

1．把浮点BASIC程序清单打印出来，用INT命令进入整数BASIC状态，再照清单逐行输入。

2．将浮点BASIC程序存入磁盘。用 LIST 命令将开始的若干程序行列示于屏幕上，再用INT 命令进入整数 BASIC状态，用编辑字符将屏上的一个个程序编辑成整数BASIC程序行，也存入磁盘。再将浮点 BASIC程序调入内存，用LIST命令列示出程序后续程序行，紧接着调入已编辑存盘的整数BASIC程序段，并再次用编辑字符将屏上的程序行编辑成整数BASIC程序行，存入磁盘上。这样反复到编辑完全部程序。

这一方法的要点是：在浮点BASIC状态下，可以把浮点 BASIC的程序段列示于屏幕上。使用 INT或FP 命令除了会使屏上增加一行命令并使屏上的内容上推外，并不改变屏上的内容。这样，在浮点 BASIC 状态下列示于屏幕上的程序行，可用编辑字符编辑成整数 BASIC程序行。对已编辑好的整数BASIC 程序的反复存入磁盘和从磁盘调入内存，是为了将后来编辑的程序行和已编辑好的程序行，编辑在同一个整数BASIC程序中。不要忘记：任何一次使用FP或INT命令，或一次LOAD命令，都会使内存中原来的BASIC程序清除掉。

3．写一个建立文本文件的程序，将浮点BASIC程序的各程序行输出到文件中去。再在整数 BASIC 系统下写一检索文件的程序，运行它可将文件中的程序行装入内存。这个程序便转化成整数BASIC程序了。

4．用机器语言写一个程序，也可以完成这种转换。但程序稍复杂。

（六）

```
10  A = 10
20  B = A + B
30  PRINT  B
40  END
```

(七)

```
95D2 - 0B  0A  00  4F  C1  34
95D8 - B3  03  00  72  01  0E  14  00
95E0 - 55  C9  56  B0  00  00  57  B3
95E8 - 03  00  01  0A  1E  00  C1  2D
95F0 - C9  72  71  C9  01  06  28  00
95F8 - 59  C9  01  05  32  00  51  01
4A - 00  08  00  96
CA - D2  95  00  08
```

(八)

```
800 - C1  00  0C  08  00  00  01  00
808 - 02  00  03  00  C9  00  12  08
810 - 04  00
4A - 00  08  00  96
CA - D2  95  12  08
```

附　录

附录1：ASCII码表

1.机内ASCII码表：

ASCII码 十进制	ASCII码 十六进制	ASCII 字符	键	ASCII码 十进制	ASCII码 十六进制	ASCII 字符	键
0	0	NULL	ctrl@	24	18	CAN	ctrlX
1	1	SOH	ctrlA	25	19	EM	ctrlY
2	2	STX	ctrlB	26	1A	SUB	ctrlZ
3	3	ETX	ctrlC	27	1B	ESCAPE	ESC
4	4	ET	ctrlD	28	1C	FS	n/a
5	5	ENQ	ctrlE	29	1D	GS	ctrl shift-M
6	6	ACK	ctrlF	30	1E	RS	ctrl∧
7	7	BEL	ctrlG	31	1F	US	n/a
8	8	BS	ctrlH或←	32	20	SPACE	Space
9	9	HT	ctrlI	33	21	!	!
10	A	LF	ctrlJ	34	22	"	"
11	B	VT	ctrlK	35	23	#	#
12	C	FF	ctrlL	36	24	$	$
13	D	CR	ctrlM或RETURN	37	25	%	%
14	E	SO	ctrlN	38	26	&	&
15	F	SI	ctrlO	39	27	'	'
16	10	DLE	ctrlP	40	28	(	(
17	11	DC_1	ctrlQ	41	29	)	)
18	12	DC_2	ctrlR	42	2A	*	*
19	13	DC_3	ctrlS	43	2B	+	+
20	14	DC_4	ctrlT	44	2C	,	,
21	15	NAK	ctrlU或→	45	2D	-	-
22	16	SYN	ctrlV	46	2E	.	.
23	17	ETB	ctrlW	47	2F	/	/

续表1

ASCII码 十进制	ASCII码 十六进制	ASCII字符	键	ASCII码 十进制	ASCII码 十六进制	ASCII字符	键
48	30	0	0	72	48	H	H
49	31	1	1	73	49	I	I
50	32	2	2	74	4A	J	J
51	33	3	3	75	4B	K	K
52	34	4	4	76	4C	L	L
53	35	5	5	77	4D	M	M
54	36	6	6	78	4E	N	N
55	37	7	7	79	4F	O	O
56	38	8	8	80	50	P	P
57	39	9	9	81	51	Q	Q
58	3A	:	:	82	52	R	R
59	3B	;	;	83	53	S	S
60	3C	<	<	84	54	T	T
61	3D	=	=	85	55	U	U
62	3E	>	>	86	56	V	V
63	3F	?	?	87	57	W	W
64	40	@	@	88	58	X	X
65	41	A	A	89	59	Y	Y
66	42	B	B	90	5A	Z	Z
67	43	C	C	91	5B	[	n/a
68	44	D	D	92	5C	/	n/a
69	45	E	E	93	5D	]	](shift-M)
70	46	F	F	94	5E	∧	∧
71	47	G	G	95	5F	-	n/a

2. 显示ASCII码

反方式						闪烁方式				控制字符		正常方式				小写	
十进→表示	十六进表示→	0	16	32	48	64	80	96	112	128	144	160	176	192	208	224	240
↓	↓	$00	$10	$20	$30	$40	$50	$60	$70	$80	$90	$A0	$B0	$C0	$D0	$E0	$F0
0	$0	@	P		0	@	P		0	@	P		0	@	P		0
1	$1	A	Q	!	1	A	Q	!	1	A	Q	!	1	A	Q	!	1
2	$2	B	R	"	2	B	R	"	2	B	R	"	2	B	R	"	2
3	$3	C	S	#	3	C	S	#	3	C	S	#	3	C	S	#	3
4	$4	D	T	$	4	D	T	$	4	D	T	$	4	D	T	$	4
5	$5	E	U	%	5	E	U	%	5	E	U	%	5	E	U	%	5
6	$6	F	V	&	6	F	V	&	6	F	V	&	6	F	V	&	6
7	$7	G	W	'	7	G	W	'	7	G	W	'	7	G	W	'	7
8	$8	H	X	(	8	H	X	(	8	H	X	(	8	H	X	(	8
9	$9	I	Y	)	9	I	Y	)	9	I	Y	)	9	I	Y	)	9
10	$A	J	Z	*	:	J	Z	*	:	J	Z	*	:	J	Z	*	:
11	$B	K	[	+	;	K	[	+	;	K	[	+	;	K	[	+	;
12	$C	L	/	,	<	L	/	,	<	L	/	,	<	L	/	,	<
13	$D	M	]	-	=	M	]	-	=	M	]	-	=	M	]	-	=
14	$E	N	∧	.	>	N	∧	.	>	N	∧	.	>	N	∧	.	>
15	$F	O	←	/	?	O	←	/	?	O	←	/	?	O	←	/	?

附录2. 保留字表

1.浮点BASIC保留字

本书中说明它们的章节如ABS，在第二章§4中描述过，标为二、4。

ABS	二、4	AND	四、4	ASC	七、5	AT	八、2	ATN	二、4
CALL	九、1	CHR$	七、5	CLEAR	十二、6	COLOR=	八、2	CONT	四、6
COS	二、4	DATA	三、5	DEF	六、3	DEL	二、6	DIM	七、2
DRAW	八、4	END	二、2	EXP	二、4	FLASH	八、1	FN	六、3
FOR	五、1	FRE	十二、6	GET	九、2	GOSUB	六、1	GOTO	四、2
GR	八、2	HCOLOR=	八、3	HGR	八、3	HGR2	八、3	HIMEM:	十二、6
HLIN	八、2	HOME	二、6	HPLOT	八、3	HTAB	八、1	IF	四、3
IN#	三、8	INPUT	三、4	INT	二、4	INVERSE	八、1	LEFT$	七、5
LEN	七、5	LET	三、3	LIST	二、6	LOAD	二、6	LOG	二、4
LOMEM:	十二、6	MID$	七、5	NEW	二、6	NEXT	五、1	NORMAL	八、1
NOT	四、4	NOTRACE	五、4	ON	六、2	ONEER	六、2	OR	四、4
PDL	九、2	PEEK	八、1	PLOT	八、2	POKE	八、1	POP	六、1
POS	八、1	PRINT	三、2	PR#	三、8	READ	三、5	RECALL	七、3
REM	四、7	RESTORE	三、6	RESUME	六、2	RETURN	六、1	RIGHT$	七、5
RND	二、4	ROT=	八、4	RUN	二、6	SAVE	二、6	SCALE=	八、4
SCRN(	八、2	SGN	二、4	SHLOAD	十二、6	SIN	二、4	SPC(	八、1
SPEED=	九、2	SQR	二、4	STEP	五、1	STOP	四、6	STORE	七、3
STR$	七、5	TAB(	八、1	TAN	二、4	TEXT	八、1	THEN	四、3
TO	五、1	TRACE	五、4	USR	十二、6	VAL	七、5	VLIN	八、2
VTAB	八、1	WAIT	十二、6	XDRAW	八、4	&	十二、6		

2.浮点BASIC保留字内部代码表：

保留字	编码	保留字	编码	保留字	编码	保留字	编码	保留字	编码
END	80	HTAB	96	RUN	AC	SPC(	C3	POS	D9
FOR	81	HOME	97	IF	AD	THEN	C4	SQR	DA
NEXT	82	ROT=	98	RESTORE	AE	AT	C5	RND	DB
DATA	83	SCALE=	99	&	AF	NOT	C6	LOG	DC
INPUT	84	SHLOAD	9A	GOSUB	B0	STEP	C7	EXP	DD
DEL	85	TRACE	9B	RETURN	B1	+	C8	COS	DE
DIM	86	NOTRACE	9C	REM	B2	-	C9	SIN	DF
READ	87	NORMAL	9D	STOP	B3	*	CA	TAN	E0
GR	88	INVERSE	9E	ON WAIT	B4 B5	/	CB	ATN	E1
TEXT	89	FLASH	9F	LOAD	B6	↑	CC	PEEK	E2
PR*	8A	COLOR=	A0	SAVE	B7	AND	CD	LEN	E3
IN*	8B	POP	A1	DEF	B8	OR	CE	STR$	E4
CALL	8C	VTAB	A2	POKE	B9	>	CF	VAL	E5
PLOT	8D	HIMEM:	A3	PRINT	BA	=	D0	ASC	E6
HLIN	8E	LOMEM:	A4	CONT	BB	<	D1	CHR$	E7
VLIN	8F	ONERR	A5	LIST	BC	SGN	D2	LEFT$	E8
HGR2	90	RESUME	A6	CLEAR	BD	INT	D3	RIGHT$	E9
HGR	91	RECALL	A7	GET	BE	ABS	D4	MID$	EA
HCOLOR=	92	STORE	A8	NEW	BF	USR	D5		
HPLOT	93	SPEED=	A9	TAB(	C0	FRE	D6		
DRAW	94	LET	AA	TO	C1	SCRN(	D7		
XDRAW	95	GOTO	AB	FN	C2	PDL	D8		

3.浮点BASIC各保留字解释执行入口地址(十六进制表示)

保留字	入口	保留字	入口	保留字	入口	保留字	入口
END	D870	HGR2	F3D8	COLOR=	F24F	GOSUB	D921
FOR	L766	HGR	F3E2	POP	D96B	RETURN	D96B
NEXT	DCF3	HCOLOR=	F6E9	UTAB	F256	REM	D9DC
DATA	D995	HPOLT	F6FE	HIMEM:	F286	STOP	D86E
INPUT	DBB2	DRAW	F769	LOMEM:	F2A6	ON	D9EC
DEL	F331	XDRAW	F76F	ONERR	F2CB	WAIT	E784
DIM	DFD9	VTAB	F7E7	RESUME	F318	LOAD	D8C9
READ	DBE2	HOME	FC58	RECALL	F3BC	SAVE	D8B0
GR	F390	ROT=	F721	STORE	F39F	DEF	E313
TEXT	F399	SCALE=	F727	SPEED=	F262	POKE	E77B
PR#	F1E5	SHLOAD	F775	LET	DA46	PRINT	DAD5
IN#	F1DE	TRACE	F26D	GOTO	D93E	CONT	D896
CALL	F1D5	NOTRACE	F26F	RUN	D912	LIST	D6A5
PLOT	F225	NORMAL	F273	IF	D9C9	CLEAR	D66A
HLIN	F232	INVERSE	F277	RESTORE	D849	GET	DBA0
VLIN	F241	FLASH	F280	&	03F5	NEW	D649

4.整数BASIC保留字

ABS	AND	ASC(	AT	AUTO
CALL	CLR	COLOR=	CON	DEL
DIM	DSP	END	FOR	GOSUB
GOTO	GR	HIMEM:	HLIN	IF
IN#	INPUT	LEN(	LET	LIST
LOAD	LOMEM:	MAN	MID	MOD
NEW	NEXT	NODSP	NOT	NOTRACE
OR	PDL	PEEK	PLOT	POKE
POP	PRINT	PR#	REM	RETURN
RND	RUN	SAVE	SCRN(	SGN
STEP	TAB	TEXT	THEN	TO
TRACE	VLIN	VTAB		

5.整数BASIC各保留字及符号的内部代码：

	0	1	2	3	4	5	6	7
0	HIMEM:	HIMEM:	^	SGN	$ 字符串变量名尾符	TAB	IF	= 串型变量赋值号
1	程序行结束符	LOMEM:	+	ABS	$	END	PRINT 后跟串型变量	= 数型变量赋值号
2	–	+ 加	(用在对字符定义长度的DIM中	PDL	(用于字符串某一字符位置取得字符	INPUT后跟串变量名	PRINT 后跟数型量	) 合法的右括号
3	: 程序行中两个语句分隔符	– 减	, DIM中两个最大下标值中间	RNDX	, 用在DIM中，其后说明的是串变量	INPUT其后带提示字符串	PRINT 其后无输出项	)
4	LOAD	* 乘	THEN 后跟行号	(DIM说明的第一个名字为数组	, 用DIM对多个名字说明时，其后为数组	INPUT后跟数型变量或数组元素	POKE	LIST 后跟行号
5	SAVE	/ 除	THEN 后跟语句	+ 正	; 用在PRINT中，其后为串型量	FOR	, 跟在POKE之后	, 跟LIST之后
6	CON	= 相等对数型量	, 用于INPUT中，其后为字符串变量名	– 负	; 用在PRINT中，其后为数型量	= 循环变量赋初值	COLOR=	LIST 列示全部程序
7	RUN带行号	* 不等对数型量	, 用于INPUT中，其后为数型量	NOT	; 用在PRINT语句尾	TO	PLOT	POP

续表

	0	1	2	3	4	5	6	7
8	RUN 不带行号	>=	" 字符串常量左引号	(表达式左括号	, 用在PRINT中，其后为串型量	STEP	, 用在PLOT中	NODSP 对串型变量
9	DEL	>	" 字符串常量右引号	= 相等对串型量	, 用在PRINT中，其后为数型量	NEXT	HLIN	NODSP 对数型量
A	, 用在DEL命令中	<=	(用于使用串型量中分量	# 不等对串型量	, 用于PRINT语句尾	, 用于对多个循环变量NEXT时的分隔	, 用在HLIN中	NOTRACE
B	NEW	<>	!	LEN(	TEXT	RETURN	AT	DSP 后跟串变量
C	CLR	<	!	ASC(	GR	GOSUB	VLIN	DSP 后跟数型量
D	AUTO	AND	(对数组元素使用或跟在TAB之后	SCRN(	CALL	REM	, 用在VLIN中	TRACE
E	, 用在AUTO命令中	OR	PEEK	, 用于SCRN函数中	DIM 对字符说明	LET	AT	PR#
F	MAN	MOD	RND	(函数左括号	DIM 对数组说明	GOTO	VTAB	IN#

附录3：出错信息表

1.浮点BASIC出错信息

浮点 BASIC 出错信息的编号，寄存于222号单元中。运行中的错误同时会显示出来，用户可视情况作必要处理。

现将各错误信息编号、显示内容及其意义依次说明如下：

① 0号：NEXT WITHOUT FOR ERROR

NEXT没有对应的FOR

②16号：SYNTAX ERROR

语法错误

③22号：RETURN WITHOUT GOSUB ERROR

RETURN没有对应的GOSUB

④42号：OUT OF DATA ERROR

DATA语句提供的数据不够

⑤53号：ILLEGAL QUANTITY ERROR

数值不合法。

⑥69号：OVERFLOW ERROR

数值太大或太小。

⑦77号：OUT OF MEMORY ERROR

内存空间不够。

⑧90号：UNDEF'D STATEMENT ERROR

行号没有定义。

⑨107号：BAD SUBSCRIPT ERROR

下标太大。

⑩120号：REDIM'D ARRAY ERROR

数组被重复说明

⑪133号：DIVISION BY ZERO ERROR

作除法运算除数为 0。

⑫149号：ILLEGAL DIRECT ERROR

不能用于立即执行方式。

⑬163号：TYPE MISMATCH ERROR

数据类型不相容。

⑭176号：STRING TOO LONG ERROR

字符串太长。

⑮191号：FORMULA TOO COMPLEX ERROR

有两个以上的IF…THEN在同一程序行中，视为错误句法。

⑯210号：CAN'T CONTINUE ERROR

不能继续运行。

⑰224号：UNDEF'D FUNCTION

函数没有定义

2.整数BASIC出错信息

*** SYNTAX ERR	语法或打印错误。
*** >32767 ERR	输入或计算的数值小于－32767或大于32767。
*** >255 ERR	一个应限制在0到255范围内的数超出了这个范围。
*** BAD BRANCH ERR	企图跳转至程序中不存在的行号。
*** BAD RETURN ERR	企图执行多于已执行的GOSUB语句的 RETURN语句。
*** BAD NEXT ERR	企图执行没有对应的FOR 语句的NEXT语句。
*** 16 GOSUBS ERR	GOSUB语嵌套深度大

	于16。
* * * 16 FOR ERR	循环嵌套超过16层。
* * * NO END ERR	最后执行的语句不是END。
* * * MEM FULL ERR	程序需要的存贮量超过存贮器调配的容量。
* * * TOO LONG ERR	在输入行中超过12层括号或多于128个字符。
* * * DIM ERR	企图定义一个已定义过的字符串或数组。
* * * RANGE ERR	一个数组大于定义的维数或小于1，或者HLIN、VLIN、PLOT、TAB、VTAB的变量超出范围。
* * * STR OVFL ERR	赋于一个字符串的字符个数超过定义时给定的长度。
* * * STRING ERR	企图执行一个不合法的串操作。
ERTYPE LINE	对于INPUT语句，输入不合法的数据。这一信息要求重新输入不合法的项。

附录4：专用单元表

十六进制	十进制		用途
	正地址	负地址	
20	32	-65504	文本窗口 左边界
21	33	-65503	文本窗口 宽
22	34	-65502	文本窗口 顶
23	35	-65501	文本窗口 底
32	50	-65486	文本方式下字符显示方式
C050	49232	-16304	显示图形方式
C051	49233	-16303	显示文本方式
C052	49234	-16302	整幅屏幕
C053	49235	-16301	混合方式
C054	49236	-16300	显示第1页面
C055	49237	-16299	显示第2页面
C056	49238	-16298	低清晰度图形
C057	49239	-16297	高清晰度图形
C000	49152	-16384	键盘接收单元
C010	49168	-16368	清键单元
C030	49200	-16336	扬声器
C061	49249	-16287	0号按钮输入
C062	49250	-16286	1号按钮输入
C063	49251	-16285	2号按钮输入
C064	49252	-16284	0号旋钮输入
C065	49253	-16283	1号旋钮输入
C066	49254	-16282	2号旋钮输入
C067	49255	-16281	3号旋钮输入