邮电高等函授教材

Quick BASIC 语言程序设计

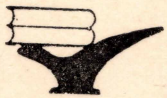
张 富 编著

北京邮电学院出版社

责任编辑：郑　捷
封面设计：宋　靖

ISBN 7-5635-0191-6





ISBN 7-5635-0191-6/TP·17
定　价：11.25 元

Quick BASIC
语言程序设计

张 富 编著

北京邮电学院出版社

图书在版编目(CIP)数据

Quick BASIC 语言程序设计/张富编著.—北京:北京邮电学院出版社,1994

ISBN 7-5635-0191-6

Ⅰ.Q… Ⅱ.张… Ⅲ.BASIC 语言-程序设计 Ⅳ.TP312BA

中国版本图书馆 CIP 数据核字(94)第 09428 号

内 容 简 介

本书详细地介绍了 Quick BASIC 的集成开发环境及其操作方法;全面地讲述了 Quick BASIC 的语言功能和结构化的程序设计方法;通过大量实例说明了 Quick BASIC 在数值计算、数据处理及事务管理等方面的应用。

本书可作为高等学校或培训班的教材,也可作为科技工作者自学参考用书。

前　言

BASIC（Beginner's All-purpose Symbolic Instruction Code）高级程序设计语言，是美国新罕什布尔州 Dartmonth 学院教授 J. G. Kemeny 和 T. E. Kurtz 于 1964 年创立的。此后，BASIC 语言有了许多修改和增强，使它成为世界上最流行的通俗程序设计语言之一。如在微型计算机上广范使用的 Microsoft 公司开发的 BASIC 和 BASICA 系统，它们不仅易学，而且功能也很强，适用于一般的数值计算、数据处理和事务管理。

80 年代，BASIC 语言又有了新的重大发展，出现了如 True BASIC，Quick BASIC，Turbo BASIC 等各种版本的 BASIC 语言。与以前的 BASIC 语言不同，它们都是结构化的程序设计语言，不仅增强了语言功能，还提高了运行速度。本书介绍其中尤为引人重视的、由 Microsoft 公司推出的 Quick BASIC 语言。

Quick BASIC 将编辑、调试、运行和编译融为一体，为程序设计人员提供了一个完整的程序设计开发环境。这个良好的编程环境与丰富的语言功能的结合，可以高速设计出高质量的程序。因为在 Quick BASIC 环境中，所有程序设计工具都已建立起来，并在运行中，能够任意进行编辑、调试与运行或者编译。

归纳起来，Quick BASIC 有如下特点：

• 立即执行程序　Quick BASIC 不是传统的解释程序。它直接把所写的每一新程序行，立即翻译成计算机可执行的指令。无论编程进行到哪里，随时都可以运行程序。

• 编译程序　当对程序的运行结果满意时，为了提高程序的运行速度，可以立即在 Quick BASIC 环境中，把它编译成能在 DOS

状态下独立运行的EXE程序文件。在DOS状态下也能对源程序进行编译并建立EXE文件。

·联机帮助（Help）和语法检查　Quick BASIC的联机帮助能为所有的BASIC语句、函数、菜单命令以及ASCII码提供帮助。Quick BASIC的编辑程序能自动检查键入的语句行的语法，以便及时校正错误。

·立即调试　在Quick BASIC环境中，在编程的同时就可以调试程序。这里有多种调试手段供使用。

·多模块　把大型程序分解为逻辑上相互独立的模块，使程序的编写和调试容易得多，而且使程序结构清楚。Quick BASIC还可以方便地建立多模块程序。它可以在内存中保存所有用户模块并提供一张模块清单。

·多编辑窗口　Quick BASIC的多窗口编辑方式，不仅可同时观察程序的不同部分，而且还为调试程序提供了方便。

除上述程序设计环境特点外，Quick BASIC还增强了语言功能。主要是：

·支持二进制文件　这意味着可以以任意格式建立文件或处理文件。

·使用用户数据类型　用户数据类型的采用，简化了随机文件的输入和输出操作，有利于建立复杂的数据结构。

·子程序和函数可递归调用　子程序和函数的递归调用，能使编写的程序更精炼和更高效。

·完善的流程控制语句　在Quick BASIC中包含有象SELECT CASE和IF---THEN---ELSE--这样一些有利于结构化程序设计的语句块。

·32位（BIT）长整型和8087/80287协处理器　这些功能可帮助我们作快速准确的算术运算，从而提高整个程序的运行速度。

·建立程序库　Quick BASIC可以方便地帮助我们建立和使用用户程序库，从而增强了软件的设计手段。

综上所述，Quick BASIC 是一个具有良好程序设计环境、语言功能强的结构化程序设计语言。它既象以前的 BASIC 那样容易学习和使用，又有 Pascal 语言那样的结构控制能力。

本书以 Quick BASIC 4.0 版本为依据，系统介绍它的程序设计原理和编程技术。

作为程序设计的自学教材，本书力求通俗易懂，深入浅出，由具体问题入手，通过程序实例，讲解语言的功能、应用和程序设计方法。全书内容的安排，遵循由简到繁，循序渐进的原则，因而适合各类程序设计人员自学。

计算机程序设计语言是一种实践性较强的课程。在学习过程中必须重视作练习题和上机操作两个实践环节，这是学好程序设计语言的关键。本书各章后面都有一定数量的习题，供读者练习编程和上机实习之用。

目　　录

前　言

第一章　绪论

第一节　电子计算机系统 …… (1)

第二节　数制与码制 …… (5)

一、数制 …… (5)

二、各种计数制之间的转换 …… (8)

三、数的原码、补码和反码 …… (13)

第三节　电子计算机的应用 …… (15)

一、数值计算 …… (16)

二、数据处理 …… (16)

三、自动控制 …… (16)

四、事务管理 …… (16)

第四节　应用计算机解题的步骤 …… (16)

第五节　程序设计语言 …… (18)

小　结 …… (22)

习　题 …… (23)

第二章　PC DOS 操作系统

第一节　磁盘与磁盘操作系统 …… (24)

一、软盘和软盘驱动器 …… (24)

二、磁盘操作系统 PC DOS …… (26)

三、磁盘文件目录 …… (27)

第二节　DOS 的装入 …… (28)

第三节　PC DOS 常用命令 …… (30)

一、盘片的格式化（初始化） …… (30)
二、磁盘文件目录显示 …… (32)
三、删除文件 …… (33)
四、子目录的建立、进入、退出和删除 …… (33)
五、文件的复制和核对 …… (34)
六、文件的输出 …… (37)
小　结 …… (38)
习　题 …… (38)
第三章　对 Quick BASIC 编程环境的初步认识
第一节　Quick BASIC 系统的安装 …… (39)
一、Quick BASIC 系统的基本软件 …… (39)
二、系统的安装 …… (40)
第二节　Quick BASIC 屏幕 …… (42)
第三节　Quick BASIC 程序的基本操作 …… (44)
一、用户程序的输入 …… (44)
二、程序存入磁盘和从磁盘装入内存 …… (46)
三、BASIC 程序的调试与运行 …… (52)
四、BASIC 程序清单的显示与输出打印 …… (53)
五、退出 Quick BASIC …… (55)
第四节　在磁盘上建立可执行文件 …… (56)
小　结 …… (59)
习　题 …… (60)
第四章　BASIC 语言基础
第一节　语言要素 …… (61)
一、BASIC 字符集 …… (61)
二、BASIC 程序行 …… (63)
第二节　数据类型 …… (65)
一、基本数据类型 …… (65)
二、用户定义的数据类型 …… (67)

第三节　常数与变量 …… (70)
一、常数 …… (71)
二、变量 …… (73)
第四节　BASIC 标准函数 …… (76)
第五节　BASIC 表达式 …… (79)
一、表达式与运算 …… (79)
二、算术运算与算术表达式 …… (79)
三、关系运算与关系表达式 …… (82)
四、逻辑运算与逻辑表达式 …… (83)
第六节　程序模块和过程 …… (86)
一、程序模块 …… (86)
二、过程 …… (86)
三、用户程序库 …… (87)
小　结 …… (88)
习　题 …… (90)
第五章　简单程序设计
第一节　变量的赋值——赋值语句 …… (92)
一、最简单的 BASIC 程序 …… (92)
二、赋值语句 LET …… (93)
三、数据类型转换 …… (95)
第二节　输出语句 …… (96)
一、PRINT 和 LPRINT 语句 …… (97)
二、TAB 函数与 SPC 函数 …… (100)
三、按给定格式输出数据…… (102)
第三节　READ 和 DATA 语句…… (107)
一、DATA 语句…… (107)
二、READ 语句…… (108)
三、恢复数据语句…… (110)
第四节　用键盘输入方式提供数据…… (112)

一、INPUT 语句 …… (112)
二、LINE INPUT 语句 …… (115)
三、键盘输入函数 INKEY $ …… (116)
第五节 简单程序举例…… (117)
小 结…… (122)
习 题…… (124)
第六章 分支程序设计
第一节 流程图…… (127)
第二节 GOTO 语句和 IF 语句 …… (133)
一、无条件转移语句 GOTO …… (133)
二、IF…THEN…ELSE 语句 …… (136)
第三节 SELECT CASE 语句 …… (144)
第四节 RANDOMIZE 语句和 TIMER、RND 函数 …… (148)
小 结…… (153)
习 题…… (153)
第七章 循环程序设计
第一节 循环程序结构…… (157)
第二节 循环语句及其应用…… (163)
一、FOR…NEXT 循环 …… (163)
二、WHILE…WEND 循环 …… (173)
三、DO…LOOP 循环…… (177)
第三节 多重循环程序…… (190)
小 结…… (197)
习 题…… (198)
第八章 下标变量的应用
第一节 数组与下标变量…… (201)
一、数组与下标变量的概念…… (201)
二、数组的说明…… (204)
三、静态数组与动态数组…… (207)

四、LBOUND 和 UBOUND 函数 …… (212)
第二节　下标变量应用举例…… (214)
小　结…… (225)
习　题…… (227)
第九章　过程、模块和库
第一节　过程的定义与调用…… (229)
一、过程的概念…… (229)
二、函数过程的定义和调用…… (232)
三、子程序过程的定义和调用…… (237)
第二节　传送自变量到过程…… (240)
一、传送常数和表达式…… (240)
二、传送变量…… (241)
第三节　变量和常数的作用范围…… (246)
一、局部变量和常数…… (246)
二、全局变量…… (248)
三、用 SHARED 语句定义共享变量 …… (251)
第四节　STATIC 变量和 STATIC 语句 …… (253)
一、STATIC 变量 …… (253)
二、STATIC 语句 …… (255)
第五节　多模块程序结构…… (256)
一、多模块程序…… (256)
二、链接方式的多模块程序…… (268)
第六节　程序库的应用…… (271)
一、库的建立…… (272)
二、使用快速库…… (273)
三、向快速库增添新模块…… (274)
小　结…… (274)
习　题…… (275)
第十章　字符串

第一节　字符串常数与字符串变量…… (276)
一、字符串常数…… (276)
二、字符串变量…… (277)
三、可变长度和固定长度字符串…… (277)
第二节　字符串的合并与比较…… (279)
第三节　字符串函数与语句…… (283)
一、字符串的搜索…… (283)
二、部分字符串的截取…… (287)
三、字符串的修改…… (292)
四、字符串、数与 ASCII 码的转换…… (294)
小　结…… (297)
习　题…… (297)
第十一章　数据文件
第一节　文件概述…… (299)
第二节　顺序文件…… (301)
一、顺序文件的打开与关闭…… (301)
二、顺序文件中的记录格式…… (303)
三、顺序文件的建立和向文件增加数据…… (304)
四、读取顺序文件中的数据…… (306)
五、顺序文件的更新…… (307)
第三节　随机文件…… (310)
一、随机文件的打开与关闭…… (310)
二、随机文件记录的格式…… (310)
三、随机文件的建立…… (312)
四、随机文件的读取…… (313)
第四节　二进制文件…… (316)
一、二进制文件的打开与关闭…… (316)
二、二进制存取…… (317)
三、文件指针…… (320)

第五节　设备文件……（321）
小　结……（323）
习　题……（323）
第十二章　图形
第一节　屏幕显示……（325）
第二节　屏幕显示方式设置……（327）
第三节　基本几何图形的绘制……（329）
一、画点……（329）
二、画直线和矩形框……（332）
三、画圆、椭圆和弧……（337）
第四节　连续画线……（340）
第五节　图形颜色的处理……（347）
一、改变前景和背景颜色……（347）
二、改变调色板属性……（350）
三、填充图形……（354）
第六节　图形窗口……（360）
一、定义图形窗口……（361）
二、建立文本窗口……（362）
三、重定义图形窗口的坐标……（363）
第七节　简单动画技术……（366）
一、字符动画……（366）
二、图形方式动画……（368）
三、用 GET 和 PUT 语句作动画 ……（369）
四、屏幕分页动画……（373）
小　结……（375）
习　题……（375）
第十三章　陷阱技术
第一节　错误陷阱……（376）
一、设置错误陷阱……（376）

二、错误的模拟与识别…… (377)
三、从错误处理程序返回…… (378)
第二节　事件陷阱…… (380)
一、设置事件陷阱…… (380)
二、事件陷阱的打开与关闭…… (381)
三、事件的处理…… (382)
第三节　功能键陷阱和时钟陷阱…… (382)
一、功能键陷阱…… (382)
二、用户定义键陷阱…… (383)
三、时钟陷阱…… (386)
第四节　音乐事件陷阱…… (387)
一、音乐的演奏…… (387)
二、音乐事件陷阱——后台演奏…… (390)
小　结…… (391)
习　题…… (392)
参考文献…… (392)

第一章 绪 论

在开始学习 BASIC 语言程序设计之前，我们先要对电子计算机硬件的基本原理、应用和软件有一个初步的了解，作为学习 BASIC 语言的预备知识。

本章简要介绍电子计算机系统（主要是微型电子计算机系统）的基本构成，数制与码制，计算机的应用，计算机解题的基本过程和计算机程序设计语言等方面的知识。

第一节 电子计算机系统

电子计算机是一种能自动、高速进行运算和数据处理的电子机器。与任何其他计算机械和自动机不同，它能按着人们事先输入给它的一系列指令（即程序），自动地进行工作。这就是说，计算机具有记忆和判断能力。

1946 年，世界上出现第一台电子计算机 “ENIAC”。全机使用了 18000 个电子管，1500 个继电器，耗电 150 千瓦，占地 1800 平方英尺。运算速度为每秒 5000 次。从那时起，至今电子计算机已经历了电子管、晶体管、集成电路、大规模和超大规模集成电路四代的演变。计算机的体积越来越小，功能越来越强，可靠性越来越高，价格越来越低，应用越来越广。下一代电子计算机正在向人工智能方向发展。

电子计算机从原理上可分为电子模拟计算机和电子数字计算机。前者以连续变化的（模拟的）电压表示运算的量，后者以离散的（数字的）量表示运算的量。还有把模拟技术和数字技术结

合起来的混合式电子计算机。

按用途，电子计算机可分为通用电子计算机和专用电子计算机两大类。

按规模（如存储容量的大小，运行速度的高低），计算机又可分为大型、中型、小型和微型电子计算机。我们所涉及的主要是通用的、数字的微型电子计算机。

电子计算机通常由运算器、存储器、控制器、输入设备和输出设备五个基本部分组成，如图 1.1.1 所示。

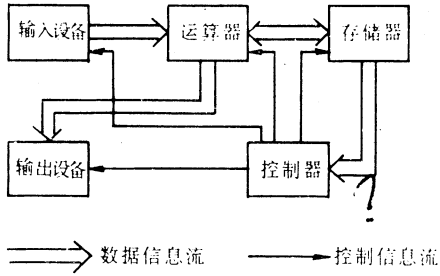


图 1.1.1　电子计算机组成框图

下面通过图 1.1.1 来说明计算机的工作原理。

电子计算机解题与人们用算盘解题的过程非常相似。用算盘解题时，需要算盘作为运算工具；需要有纸和笔，用来记录原始数据、运算的步骤、运算的中间结果和最终结果；还必须有控制整个运算工作的人。

当使用计算机进行运算时，为了把原始数据和计算步骤输入到计算机里，计算机要有输入设备。计算机中能够保存（记忆）原始数据、计算步骤以及计算结果的装置称为存储器。它们相当于上面提到的纸。计算机中担负算盘作用的装置就是运算器，由它来进行各种运算。起着人的大脑指挥作用的装置，在计算机中称为控制器。控制器根据存储器中记忆的计算步骤，指挥计算机的各部分协调的工作。保存在存储器中的计算结果，可由输出设备输出。这样，输入和输出设备相当于人工计算中的笔。

由上述可知，在计算机中有两种信息流动，一种是数据，另一种是控制信号，数据（包括原始数据，计算步骤）是由输入设备通过运算器存入存储器中的。在运算过程中，原始数据从存储器读入运算器进行运算，运算的中间结果和最终结果，再存入存储器中，或由运算器送输出设备输出。这些动作都是数据信息在计算机内部的流动。存储在存储器中的计算步骤叫作程序（Program）。程序也以数据信息的形式进入控制器。控制器根据程序的具体要求，产生一系列控制信号，送往各装置，控制它们完成相应的动作，如进行某种运算，输出数据，数据存入存储器等。在控制器与其他设备之间传送的这些信息，构成了计算机内部的控制信息流。

通常，把运算器和控制器合在一起，统称为中央处理器 CPU（Central Processing Unit）。中央处理器和存储器合在一起，组成计算机的主机。各种输入输出设备统称为计算机的外部设备（Peripheral），简称外设。

常用的输入设备有卡片输入机，键盘等。常用的输出设备有各种打印机，显示器等。

主机中的存储器也称为内存储器，简称内存。除了内存储器外，计算机还常配有外存储器，简称外存。外存的容量要比内存的容量大得多，但存取速度不如内存储器快。外存用来存储暂时不用的大量数据，包括各种软件。常用的外存储器有磁鼓、磁带、磁盘等。

图 1.1.1 中计算机的各个物理部件，构成了计算机的硬件（Hardware）。存储器中存储的程序则称为软件（Software）。

70 年代初，诞生了一代新型计算机——微型计算机（Microcomputer）。它利用大规模集成电路（LSI）技术把中央处理单元（CPU）制造在一个芯片上，称为微处理器（Microprocessor）。利用大规模集成电路技术，还制造出存储器芯片，主机与外设之间的输入输出接口（I/O 接口）芯片等。可见，微型计算机就是由微处

理器、存储器和 I/O 接口电路等大规模集成电路芯片组成的电子计算机。

微型计算机的特点是体积小、重量轻、功耗低、价格低廉，因而最容易普及应用。

微型计算机的一般结构如图 1.1.2 所示。由图 1.1.2 可以看出，在微型计算机中，信息的传送是通过称之为总线的公共信号线进行的。总线通常包括有地址总线（Address Bus）、数据总线（Data Bus）和控制总线（Control Bus）三组，分别用来传送地址、数据和控制信号。

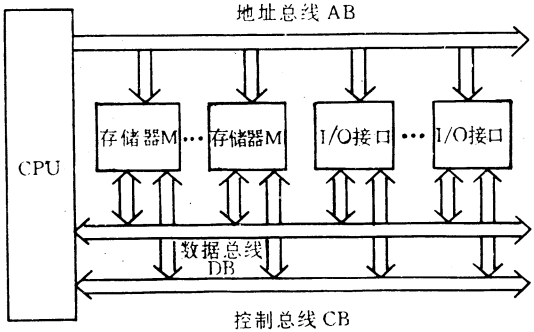


图 1.1.2　微型计算机的结构

我们已经知道，计算机能够自动地进行计算，是因为事先已把程序存在存储器中。程序是用一系列指令表达的计算步骤，或者说是指令的有序集合，每条指令能使计算机完成某一确定的操作。计算机按顺序逐条执行指令，从而完成事先规定的任务。总之，计算机是通过硬件和软件协同工作来完成运算的。

计算机的软件基本上可分为系统软件和应用软件两大类。

系统软件中有用于计算机管理和控制的操作系统，程序设计语言的编译系统以及用于维护测试的服务性软件。

应用软件指的是为解决某种具体问题（如科学计算问题，数据处理问题等）所设计的应用程序。

系统软件和应用软件一般存储在外存储器，如磁盘中，使用时调入内存。

这样，一个完整的电子计算机系统，应包括硬件和软件两大部分。如微型计算机（包括 CPU 和存储器，即主机），加上软件、外设和电源，就构成了微型计算机系统（Microcomputer System）。

图 1.1.3 表示出 IBM PC 微型计算机系统设备的基本配置情况。

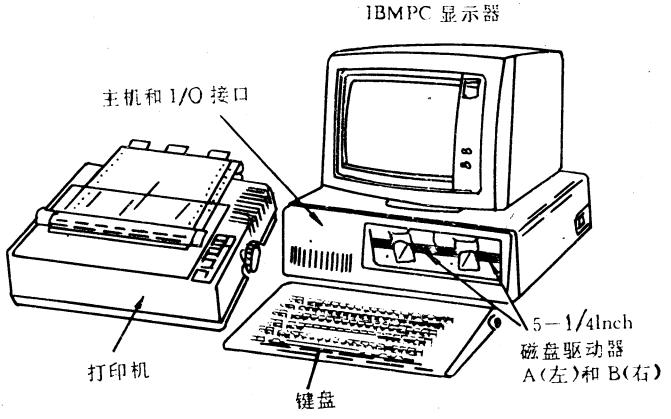


图 1.1.3 微型计算机系统

第二节 数制与码制

在 BASIC 语言中，主要使用十进制数，但是也允许使用八进制数和十六进制数，在逻辑运算中，还要用到二进制数。因此，在这一节，我们将简要地说明各种不同进制数的表示及其互相转换的方法。数在计算机中是以二进制数的某种编码形式存在的。最后，将重点介绍在微型计算机中广泛应用的补码概念。

一、数制

1. 十进制数

我们日常使用的数，基本上都是十进制数。十进制数有以下三个特点：

（1）它有 0，1，2，3，4，5，6，7，8，9 十个数符。任何一个十进制数，都是由其中的若干个数符组合而成的。例如，8.937，89 等就是十进制数。

（2）一个数的大小，是由数符及其在数中的位置决定的。设有一个十进制数

$$d_n \quad d_{n-1} \quad \cdots \quad d_0 \quad d_{-1} \quad d_{-2} \quad \cdots \quad d_{-m}$$

其中每个符号 d_i（$i=n, n-1, \cdots, 0, -1, -2, \cdots \cdots, -m$）都是一个数符。这个数的值可用如下的展开式得到：

$$N=d_n \cdot 10^n+d_{n-1} \cdot 10^{n-1}+\cdots+d_0 \cdot 10^0+d_{-1} \cdot 10^{-1}+d_{-2} \cdot 10^{-2}+\cdots+d_{-m} \cdot 10^{-m}$$

例如，45.23 的展开式可写为

$$4\times10^1+5\times10^0+2\times10^{-1}+3\times10^{-2}$$

式中的“10”是数制的进位基值。十进制数的基值为 10。

（3）十进制数的计数原则是“逢十进一”和“借一当十”。

这三个特点，描述了数制中最本质的内容。从这三个特点出发，不难了解其他数制。

2. 二进制数

在数字电子计算机中，数是以二进制数的形式存储的。二进制数是以“2”为进位基值的数。它有两个数符：0 和 1。它的计数原则是“逢二进一，借一当二”。一个二进制数 1010.101 的展开式为

$$1\times2^3+0\times2^2+1\times2^1+0\times2^0+1\times2^{-1}+0\times2^{-2}+1\times2^{-3}$$

计算上面的展开式得 10.625。这就是 1010.101 二进制数所表示的十进制数。

根据“逢二进一，借一当二”的计数原则，我们很容易进行二进制数的四则运算。

例 1.2.1 计算两个二进制数 10011.01 和 100011.11 之和。

$$\begin{array}{r} 10011.01 \\ +)\ 100011.11 \\ \hline 110111.00 \end{array}$$

即　　10011.01＋100011.11＝110111.00

例 1.2.2　计算两个二进制数 10110.01 和 1100.1 之差。

$$\begin{array}{r} 10110.01 \\ -)\ \ 1100.10 \\ \hline 1001.11 \end{array}$$

3. 八进制数

利用上述数制的三个特点，可以描述计算机技术中广泛使用的八进制数和十六进制数。八进制数和十六进制数是二进制数理想的缩写符号。

八进制数是以“8”为进位基值的数。它有 0，1，2，3，4，5，6，7 八个数符。八进制数的计数原则是“逢八进一，借一当八”。其展开式为

$$N=d_n\cdot 8^n+d_{n-1}\cdot 8^{n-1}+\cdots+d_0\cdot 8^0+d_{-1}\cdot 8^{-i}+d_{-2}\cdot 8^{-2}+\cdots+d_{-m}\cdot 8^{-m}$$

根据“逢八进一，借一当八”的原则，可以进行八进制数的四则运算。

例 1.2.3　计算八进制数 35 和 67 的和。

$$\begin{array}{r} 35 \\ +)\ \ 67 \\ \hline 124 \end{array}$$

即　　35＋67＝124

4. 十六进制数

十六进制数是以“16”为进位基值的数。它有 0，1，2，3，4，5，6，7，8，9，A，B，C，D，E，F 十六个数符。计数原则是“逢十六进一，借一当十六”。例如，十六进制数 35BE 的展开式为

$$3\times 16^3+5\times 16^2+B\times 16^1+E\times 16^0$$

把数符 B 看作十进制数的 11，数符 E 看作是十进制数的 14，上式经十进制计算后，得 13758。这就是十六进制数 35BE 的十进制表示。

根据“逢十六进一，借一当十六”的原则，可以进行十六进制数的四则运算。

由于使用了不同进位制的数，所以在写一个数时，必须说明它的进位计数制，就象上面所写的那样：“二进制数 1101”，“八进制数 35”，“十六进制数 35BE”等。然而，这种写法不太方便。今后，当需要强调数的进位制时，就把数用圆括号括起来，在右括号的下角，用阿拉伯数字表示该数的进位基值。例如，十六进制数 35BE 表示为 $(35BE)_{16}$，在程序中还可写成 35BEH。八进制数 54 表示为 $(54)_8$，二进制数 1010 表示为 $(1010)_2$。

二、各种计数制之间的转换

在使用计算机进行运算时，经常需要进行不同数制的数之间的变换。

二进制数、八进制数和十六进制数转换为十进制数，可直接用数的展开式实现。例如，

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

$$(35)_8=3\times8^1+5\times8^0=(29)_{10}$$

$$(35)_{16}=3\times16^1+5\times16^0=(53)_{10}$$

十进制数转换为二进制数、八进制数和十六进制数，是上述转换的逆变换。因此，转换方法仍是以数的展开式为基础。

十进制正整数 N 的 P 进制数的求法，称为“除 P 取余法”。就是用 P 去除 N，得一整数商和一余数，此余数就是所求 P 进制数的最低位。用 P 再去除所得之商，又得一整数商和一余数。此余数是该 P 进制数的次低位。这样不断地除下去，直到商为 0 止。下面用例子具体说明这个方法。

例 1.2.4 将十进制数 477 转换为八进制数、十六进制数和

二进制数。

用“除八取余法”求十进制数 477 的八进制数：

$$
\begin{aligned}
477 \div 8 &= 59 \quad \text{余} \quad 5 \\
59 \div 8 &= 7 \quad \text{余} \quad 3 \\
7 \div 8 &= 0 \quad \text{余} \quad 7
\end{aligned}
$$



即 $(477)_{10} = (735)_8$

用“除十六取余法”求十进制数 477 的十六进制数：

$$
\begin{aligned}
477 \div 16 &= 29 \quad \text{余} \quad 13\ (\mathrm{D}) \\
29 \div 16 &= 1 \quad \text{余} \quad 13\ (\mathrm{D}) \\
1 \div 16 &= 0 \quad \text{余} \quad 1
\end{aligned}
$$



即 $(477)_{10} = (1\mathrm{DD})_{16}$

用“除 2 取余法”求十进制数 477 的二进制数：

$$
\begin{aligned}
477 \div 2 &= 238 \quad \text{余} \quad 1 \\
238 \div 2 &= 119 \quad \text{余} \quad 0 \\
119 \div 2 &= 59 \quad \text{余} \quad 1 \\
59 \div 2 &= 29 \quad \text{余} \quad 1 \\
29 \div 2 &= 14 \quad \text{余} \quad 1 \\
14 \div 2 &= 7 \quad \text{余} \quad 0 \\
7 \div 2 &= 3 \quad \text{余} \quad 1 \\
3 \div 2 &= 1 \quad \text{余} \quad 1 \\
1 \div 2 &= 0 \quad \text{余} \quad 1
\end{aligned}
$$



即 $(477)_{10} = (111011101)_2$

十进制小数 M（0＜M＜1）的 P 进制数的求法，称为“乘 P 取整法”，就是用 P 乘 M，其乘积的整数部分就是 P 进制小数的最高位。用 P 再去乘上次所得之积的小数部分，又得一乘积，其整数部分是 P 进制数的次高位。这样不断地乘下去，直到乘积的小数

部分等于零时为止。举例说明如下。

例 1.2.5 求十进制数 0.451171875 的八进制数、十六进制数和二进制数。

乘 8 取整：

$$
\begin{aligned}
0.451171875 \times 8 &= 3.609375 \quad \text{取} \quad 3 \\
0.609375 \times 8 &= 4.875 \quad \text{取} \quad 4 \\
0.875 \times 8 &= 7.0 \quad \text{取} \quad 7
\end{aligned}
$$

得 $(0.451171875)_{10} = (0.347)_8$

乘 16 取整：

$$
\begin{aligned}
0.451171875 \times 16 &= 7.21875 \quad \text{取} \quad 7 \\
0.21875 \times 16 &= 3.5 \quad \text{取} \quad 3 \\
0.5 \times 16 &= 8.0 \quad \text{取} \quad 8
\end{aligned}
$$

得 $(0.451171875)_{10} = (0.738)_{16}$

乘 2 取整：

$$
\begin{aligned}
0.451171875 \times 2 &= 0.90234375 \quad \text{取} \quad 0 \\
0.90234375 \times 2 &= 1.8046875 \quad \text{取} \quad 1 \\
0.8046875 \times 2 &= 1.609375 \quad \text{取} \quad 1 \\
0.609375 \times 2 &= 1.21875 \quad \text{取} \quad 1 \\
0.21875 \times 2 &= 0.4375 \quad \text{取} \quad 0 \\
0.4375 \times 2 &= 0.875 \quad \text{取} \quad 0 \\
0.875 \times 2 &= 1.75 \quad \text{取} \quad 1 \\
0.75 \times 2 &= 1.5 \quad \text{取} \quad 1 \\
0.5 \times 2 &= 1.0 \quad \text{取} \quad 1
\end{aligned}
$$

得 $(0.451171875)_{10} = (0.011100111)_2$

必须注意的是，不是任何一个十进制小数都能完全精确地转换成二进制数、八进制数和十六进制数。有时，在“乘 P 取整”过程中，乘积的小数部分永远不会为零。在这种情况下，可根据题

目要求的精度，换算到一定位数就可以结束了。

一个既有整数部分又有小数部分的十进制数，在转换为二进制数、八进制数或十六进制数时，可按上述方法，对整数部分和小数部分分别进行转换。

最后，说明一下二进制数、八进制数和十六进制数之间的互相转换问题。

这三种不同进制的数，进位基值都是 2 的幂，因此，它们之间的转换关系比较简单。归纳起来，就是：三位二进制数对应一位八进制数，四位二进制数对应一位十六进制数。

表 1.2.1 和表 1.2.2 分别列出了二进制数与八进制数和十六进制数之间的对应关系。

按照上述对应关系，不难进行二进制、八进制和十六进制数之间的转换。

二进制数转换为八进制数时，从小数点开始，向左和向右每三位划为一组，写出每一组二进制数所对应的八进制数，便得到所求的八进制数。

表 1.2.1

十进制	二进制	八进制
0	000	0
1	001	1
2	010	2
3	011	3
4	100	4
5	101	5
6	110	6
7	111	7

二进制数转换为十六进制数时，从小数点开始，向左和向右每四位划为一组，写出每一组二进制数所对应的十六进制数，便得到所求的十六进制数。

例 1.2.6 将二进制数 10110100111.10110111 表示成八进制数和十六进制数。

表 1.2.2

十进制	二进制	十六进制
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

转换成八进制数：

10	110	100	111	.	101	101	110
↓	↓	↓	↓		↓	↓	↓
2	6	4	7	.	5	5	6

即 $(10110100111.10110111)_2 = (2647.556)_8$

转换成十六进制数：

101	1010	0111	.	1011	0111
↓	↓	↓		↓	↓
5	A	7	.	B	7

即 $(10110100111.10110111)_2=(5A7.B7)_{16}$

同理，可以非常容易地把八进制数和十六进制数转换为二进制数。

例 1.2.7 将八进制数 735461.14 和十六进制数 32DF 转换为二进制数。

八进制数 735461.14 换算为二进制数：

7	3	5	4	6	1	.	1	4
↓	↓	↓	↓	↓	↓		↓	↓
111	011	101	100	110	001	.	001	100

即 $(735461.14)_8=(111011101100110001.001100)_2$

十六进制数 32DF 换算为二进制数：

3	2	D	F
↓	↓	↓	↓
0011	0010	1101	1111

即 $(32DF)_{16}=(0011001011011111)_2$

三、数的原码、补码和反码

数的码制是用来解决二进制数在计算机中的表示形式。数在计算机中的表示形式称为机器数。

机器数首先要解决的是数的正负号在计算机中的表示。在计算机中，正号通常用“0”表示，负号用“1”表示，并且一般是放在数的最高位之前。例如，十进制数+5 和−5 在机器中可分别表示为（设机器字长为十六位）：

+5	0000000000000101
−5	1000000000000101

数的这种表示方法称为原码，记为 $[5]_{原}$ 和 $[-5]_{原}$。

两个原码数相加时，首先要判断两数的符号是否相同。如果符号相同，就进行加法运算，其结果数的符号就是参加运算数的符号。如果两数的符号不同，就要进一步比较两数绝对值的大小，绝对值大的数作被减数，另一个作减数，进行减法运算。被减数的符号即为结果数的符号。

从上述加法运算的过程可以看出，既使是一个很简单的加法运算，其过程也是相当复杂的。这必然导致运算器的复杂化和计算时间的增长。

有没有办法，不管数是正还是负，将符号和数作为一个整体，参加一次加法运算，就能得到正确的结果呢？这种办法是有的。这就是下面要着重介绍的补码问题。

目前绝大多数计算机，数都用补码表示。为了说明补码，先讲一下反码。

一个正数的反码，等于该数的原码。一个负数的反码，等于该数原码的各二进制位变反，即 0 变 1，1 变 0，但符号位不变。

例如，+5 和 −5 的反码为（设字长 16 位）：

$[+5]_{反}$ = 0000 0000 0000 0101

$[-5]_{反}$ = 1111 1111 1111 1010

由反码很容易得出补码。

一个正数的补码，等于该数的原码。一个负数的补码，等于该数的反码在末位加 1。

例如，+5 和 −5 的补码为

$[+5]_{补}$ = 0000 0000 0000 0101

$[-5]_{补}$ = 1111 1111 1111 1011

数采用补码表示后，加法就变得非常简单。

进行两个补码数相加时，无论它们的正负号如何，只要作一次加法运算，就能得到正确的结果。其和仍是用补码表示的。这个运算可表示为：

$$[X-Y]_{补}=[X]_{补}+[-Y]_{补}$$

$$[X]_{补}+[Y]_{补}=[X+Y]_{补}$$

例 1.2.8 设机器字长为 8 位。已知 X=9，Y=−4，用补码计算 X+Y。

$$[X]_{补}=[+9]_{补}=0000\quad 1001$$
$$+)\ [Y]_{补}=[-4]_{补}=1111\quad 1100$$
$$[X+Y]_{补}=\quad 10000\quad 0101$$

最前面的一位 1 自动丢失，所以得

$$[9]_{补}+[-4]_{补}=[9-5]_{补}=0000\quad 0101$$

由于结果是正数的补码，可知计算结果为

$$(101)_2=(5)_{10}$$

例 1.2.9 设机器字长为 8 位，已知 X=−9，Y=4，用补码计算 X+Y。

$$[X]_{补}=[-9]_{补}=1111\quad 0111$$
$$[Y]_{补}=[+4]_{补}=0000\quad 0100$$
$$[X+Y]_{补}=[-9]_{补}+[4]_{补}=1111\quad 1011$$

结果是一个负数的补码。

将负数补码的各位（符号位除外）变反，即 0 变 1，1 变 0，末位上再加 1，就得到该数的原码。对于本例题有

$$[X+Y]_{原}=1000\quad 0101$$

最终得

$$X+Y=(-101)_2=(-5)_{10}$$

第三节　电子计算机的应用

计算机的应用十分广泛。从尖端科学技术、工农业生产到日常生活，到处都有计算机的应用。例如，宇宙飞船的发射，机床加工零件，都可由计算机自动、准确地完成。计算机的应用，归纳起来，大致有以下几个方面：

一、数值计算

在现代科学技术中，有大量复杂的科学计算问题，需要借助于计算机才能解决。应用计算机进行数值计算，可以节省人力，赢得时间（在有些问题中，时间是极为重要的）以及得到更精确的结果。

二、数据处理

用计算机对数据及时的记录、整理与计算，并加工成所要求的信息，称为数据处理。如企业管理，图书资料管理，实验数据的处理，仓库管理，银行算帐等都有大量数据需要记录、检索和加工。数据处理的特点是：数据量大，计算比较简单，时间性强等。用计算机进行数据处理，可以使工作效率提高到一个更高的水平。

三、自动控制

计算机可以用于各种自动控制系统，如生产过程的自动化，交通自动控制，仪表自动化等。在邮电领域中，计算机广范应用于交换系统、邮政自动化以及自动测试等方面。

四、事务管理

在机关企业的办公自动化和事务管理方面，计算机的应用越来越普及，发挥的作用也越来越大。例如人员设备文书档案的管理，工资财务的管理，信息数据的查询等。

第四节　应用计算机解题的步骤

使用计算机解题，大体上有以下几个步骤：

1. 提出和分析问题

首先针对所研究的物理过程，分析和研究已知数据、所要求解的未知量以及求解的精度等。

2. 构造数学模型

通过上述的分析和研究，把实际的物理过程用数学或逻辑关系表达出来，这就是构造数学模型的过程。例如，有许多问题可归结为计算公式

$$A = \sum_{k=0}^{n} a_k \times x^{n-k}$$

$$S = 2\int_0^x \sin x \mathrm{d}x$$

等，它们就是相应问题的数学模型。

3. 选择计算方法

对所建立的数学模型，选择适当的计算方法。有时建立的数学模型不适合于计算机计算。例如，计算机若直接按上面的第一个算式计算 A 值，当 n 很大时，计算工作量就相当大，因而需要较长的计算时间。如果把它变换成下式，则不仅可以减少计算量，还可以简化程序设计：

$$A=(\cdots)((((a_0x+a_1)x+a_2)x+a_3)x+\cdots$$
$$+a_{n-2})x+a_{n-1})x+a_n$$

有些数学模型不能直接用计算机进行计算。例如，上面的第二个算式计算的是 sin 的积分，它就不能直接算出，这时需要采用某种近似计算方法。研究适合计算机计算的算法的学科称为“数值计算方法”。

4. 程序设计

用程序设计语言，按照选定的算法，把计算过程描述出来，就是程序设计。简单的问题，可以根据算式或算法，直接把程序写出来。对于逻辑关系比较复杂的问题，在编写程序之前，还需要根据算法，把计算步骤逐条地列出来，或用流程图直观地表达出来，然后，按照所列的计算步骤或流程图，用程序设计语言编写

程序。这样做可在程序设计过程中少犯错误。

5. 调试程序与运行

上机计算，要经过一个对程序进行反复调试和修改的过程。因为程序中难免有些错误。这个过程要一直进行到程序运行正常并获得结果为止。

6. 分析计算结果

计算机的输出数据是否正确，需要分析。从实际物理过程到计算机输出结果，经过了许多过程，数学模型不合适、计算方法不正确、计算精度不够、程序设计有错都会导致不能令人满意的计算结果。这时应当重复上述某些步骤，直到获得正确的结果。

本课程重点解决第四步骤，即程序设计的问题。但在一定程度上，也涉及到计算方法问题，即第三步骤的问题。而第五步骤上机调试和运行计算，则是本课程的实践环节，读者应给以足够的重视。

第五节　程序设计语言

我们已经知道，要使计算机进行某项工作，需要把完成该项工作的步骤编制成程序。这里就存在一个问题，即用什么样的语言工具来描述程序。我们把用于编写程序的语言叫作程序设计语言。

每种型号的CPU都有自己的一套指令(instruction)，称作该机的指令系统。不同类型的CPU构成的计算机，其指令系统也不相同。指令系统中的每条指令，可使计算机完成一个确定的基本操作，这种指令系统是计算机唯一能够认识并执行的语言，称为机器语言(Machine Language)。

机器语言在计算机中用二进制代码表示。在电子计算机问世的初期，人们就是直接用二进制代码表示的机器语言指令编写程序的，完全是一种手工操作，因此也称为手编程序。用机器语言

编写的程序，输入计算机后，便能立即执行。

由于机器语言指令是用数码表示的，很难记忆，又由于一条机器指令只能完成一个简单的基本操作，一个很简单的算式，也要用好多条指令才能表达出来，因此，用机器语言编写程序，要花费较大的人力和时间，编写出来的程序，不直观，且容易出错，修改错误也极不方便。由于各种计算机的指令系统不同，使得所编制的程序，不能在不同的计算机上运行，也就是说，用机器语言编写的程序，是面向机器的，没有通用性，因而不便于交流。

上述情况，不利于计算机应用的普及。为了改变设计程序的这种落后状态，在发展计算机硬件的同时，加强了对计算机程序设计语言的研究。

人们首先想到，用一些简单、直观，接近于人类自然语言(主要是英文)的文字符号，代表机器指令的操作内容和数据地址，这些代表机器指令的文字符号，称为记忆符，整个指令系统的记忆符号就构成了所谓的符号语言。在符号语言中，每条机器指令有一条与之对应的符号指令。用符号语言编写的程序，称为符号语言程序。

计算机并不懂得符号语言，必须将符号语言程序翻译成机器语言程序，计算机才能运行。将记忆符号换成二进制的机器指令代码的工作，称为“代真”。代真工作一般不需要人工去完成，而是由称为汇编器或汇编程序(Assembler)的软件自动完成的。汇编程序起着“翻译员”的作用。

符号语言也称为汇编语言，用汇编语言编写的程序，称为汇编语言程序。这种用非机器语言编写的程序，也叫源程序(Source Program)，而由汇编程序生成的机器语言程序，叫作目的程序或目标程序(Object Program)。将汇编语言源程序翻译成机器语言的目标程序的过程如图 1.5.1 所示。

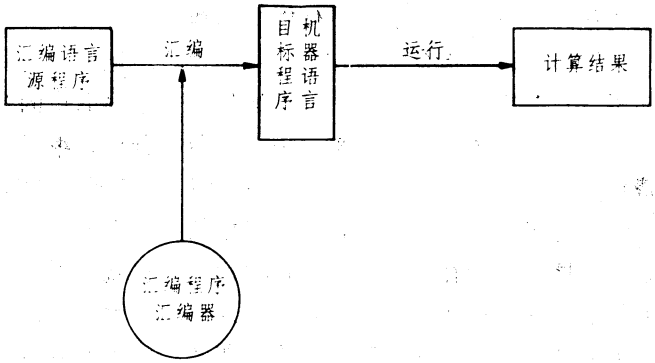


图 1.5.1 汇编过程

显然，汇编程序要比机器语言程序直观，指令容易记忆，程序也容易编写和修改。

虽然汇编语言较机器语言前进了一步，但它仍然与自然语言和数学表达方法有较大的差别。与机器语言一样，汇编语言仍是为特定的计算机服务的，不同类型的计算机，其汇编语言是不一样的，它仍然是一种面向机器的语言。用面向机器的语言编制程序，仍有较大的难度，并且没有通用性，因而不易在非计算机专业人员中推广计算机的应用。

为了普及计算机的应用，高效率地编制复杂的程序，出现了所谓的高级程序设计语言，简称高级语言(Hight Level Lang-uage)。这类语言接近自然语言和数学公式的表达方法，而且不依赖于计算机的具体型号，使用这种语言编写程序，不需了解计算机的结构和指令系统，因而可以把精力全部放在程序的逻辑上。

用高级语言编写的源程序，也需要通过翻译程序翻译成目标程序，才能在计算机上运行。

高级程序设计语言简单易学，易于推广。用高级语言编写程序，难度小、效率高、简明易读、容易修改，是编程的有力工具。

目前应用较广的高级程序设计语言有：FORTRAN，ALGOL，COBOL，PASCAL，BASIC，C 语言，dBASE 等。

把源程序翻译成目标程序有两种方法：一种是采用编译器（编译程序 Compiler）的编译方法；另一种是采用解释器（解释程序 Iterpreter）的解释方法。

编译方法是利用编译程序，把源程序作为一个整体，完整地把它翻译成目标程序，然后，这个目标程序便可以在操作系统的支持下独立地运行。编译过程如图 1.5.2 所示。

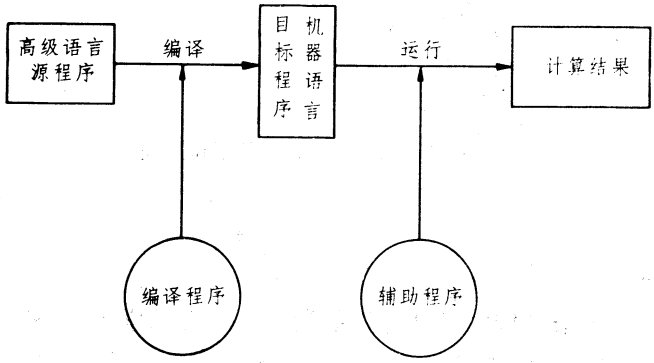


图 1.5.2　编译过程

解释方法是利用解释程序，以源程序的语句为单位，逐条地把源程序翻译成目标程序。这个目标程序只能在解释环境下运行，而不能在操作系统下独立运行。解释系统的工作过程如图 1.5.3 所示。

一般说来，解释方法运行程序，要比运行编译出的目标程序多花费机器时间。

BASIC 系统既提供解释程序，也提供编译程序。因此，用户可以选用任意一种方法运行程序。一种可能的开发程序的方法，是利用解释程序调试程序方便的特点，调试和试运行程序。程序经调试运行正确后，再用编译程序把它编译成可独立运行的目标程序，然后在操作系统下运行这个程序。

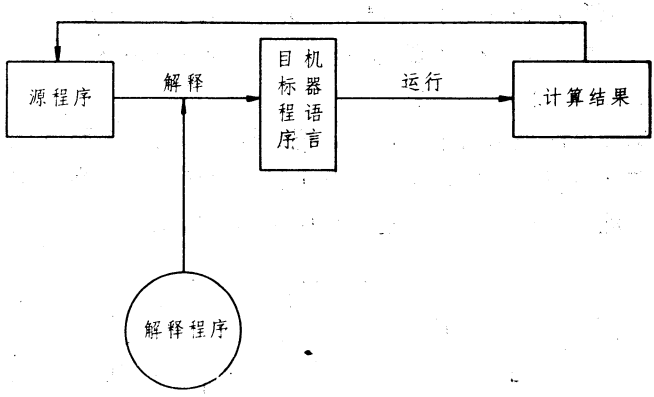


图 1.5.3　解释系统的工作过程

小　　结

对于未曾接触过计算机的读者，在开始学习 BASIC 语言之前，先要对计算机有一个初步地了解，这是掌握好 BASIC 语言所必要的。为此，本章作为预备知识，介绍了与学习计算机语言密切相关的一些问题。

本章主要讲述了以下四个方面的问题：

1. 电子计算机的组成和基本工作原理以及用电子计算机解题的基本步骤。

2. 简要地叙述了计算机技术中使用的几种数制，它们之间的转换和码制，主要是补码的应用。

3. 电子计算机在数值计算、数据处理、自动控制和事务管理四个主要领域的应用。

4. 简单扼要地介绍了计算机机器语言、汇编语言和高级语言的基本知识。

习　　题

1-1　电子计算机是由哪几个基本部分组成的？试说明其工作原理。

1-2　试说明微型计算机的结构。

1-3　什么是计算机的硬件和软件？

1-4　什么是微处理器？

1-5　试说明电子计算机在各主要领域中的应用情况。

1-6　利用电子计算机解题的步骤有哪些？

1-7　什么是机器语言？什么是汇编语言？什么是高级语言？

1-8　说明高级语言的编译与解释两种翻译系统的主要区别。

1-9　将下列十进制数转换为八进制数、十六进制数和二进制数：

(a) 75　　(b) 926

(c) 13.48828125　　(d) 0.15625

1-10　将下列二进制数转换成八进制数、十进制数和十六进制数：

(a)　1101011　　(b)　1010111

(c)　0.11011　　(d)　1011.1011

1-11　将下列数转换为十进制数：

(a)　$(321)_8$　　(b)　$(3AF2)_{16}$

1-12　将下列八进制数转换为二进制数和十六进制数：

(a)　3745　　(b)　52.45

1-13　设计算机字长为 8 位，用补码计算下列二进制数的算式：

(a)　1110＋1010　　(b)　1010＋0110

(c)　1110－1011　　(d)　1011－1110

(e)　1101＋1101　　(f)　1010－0111

第二章　PC DOS 操作系统

BASIC 系统是在操作系统的支持下运行的，因此，在学习 BASIC 语言之前，首先了解一些操作系统的基本知识和常用的一些操作系统命令。这些命令，在程序开发过程中，是很有用的。

第一节　磁盘与磁盘操作系统

一、软盘和软盘驱动器

在使用 Quick BASIC 的时候，必须先学会使用磁盘操作系统。磁盘操作系统是计算机系统中最重要的一个系统软件，这一节将结合操作 Quick BASIC 的需要，简要地介绍一些有关使用磁盘操作系统，简称 DOS（Disk Operating System）的知识。

在微型计算机系统中，广泛使用的外存储器是磁盘，磁盘又分为软磁盘和硬磁盘两种。计算机的系统软件，如操作系统，各种语言系统，用户编写的各种应用程序，都存储在磁盘上。当需要使用磁盘上的某个程序时，则从磁盘调进内存。

软磁盘是表面涂有铁磁物质并能转动的圆形塑料薄片，装在一个封闭的纸保护套里，如图 2.1.1 所示。

沿着中心圆记录信息，这些同心圆称为磁道（Track）。磁道被划分为若干个部分，称为扇区。软盘上存储信息的空间称为磁盘容量，用字节（Byte）数来表示。一千字节简记为 KB，一兆字节简记为 MB。软盘的容量、磁道和扇区数目，取决于软盘的类型。

在微型计算机上使用的软磁盘一般有表 2.1.1 所示的几种。

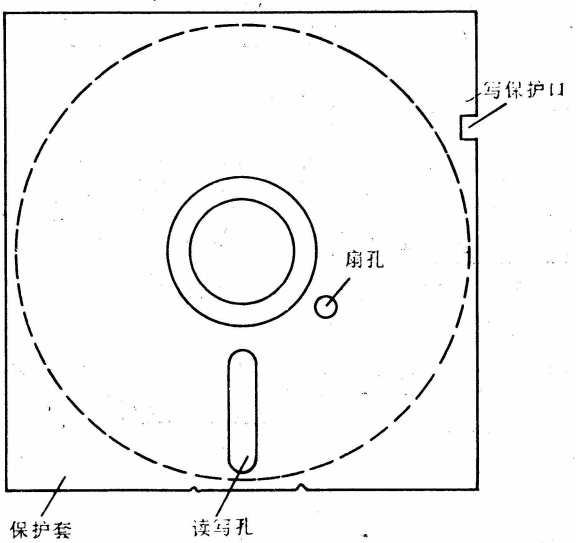


图 2.1.1 软磁盘

表 2.1.1

磁盘尺寸	说明	容量	磁道数	扇区数
5.25 英寸	单面双密度	180KB	40	8，9
5.25 英寸	双面双密度	360KB	40	8，9
5.25 英寸	双面高密度	1.2MB	80	15
3.5 英寸	双面高密度	1.44MB	80	80
3.5 英寸	双面	720KB	80	9

磁盘插入计算机系统的磁盘驱动器内，在磁盘操作系统的控制下，通过软盘旋转和磁盘驱动器磁头在磁道间的来回移动，实现往磁盘里写数据或从磁盘上读出数据。程序或数据从磁盘上取进内存称作读操作，简称读磁盘。把内存中的程序或数据存入磁盘称作写操作，简称写磁盘。计算机系统通常配置一到两个软磁

盘驱动器，分别称为A盘驱动器和B盘驱动器。硬磁盘称为C盘驱动器。符号A，B和C还称为盘符。

二、磁盘操作系统PC DOS

PC DOS（Personal Computer Disk Operation System）是美国Microsoft公司为IBM PC系列微型计算机开发的磁盘操作系统。它是用户与计算机间的接口，用户通过操作系统来使用计算机。操作系统管理包括磁盘在内的各种外围设备及内存储器，帮助用户更有效和更方便地使用计算机。

所有记录在磁盘上的数据，无论它们是系统软件还是用户程序，都称作文件。每个文件都有自己的文件名。操作系统根据文件名来对指定的文件进行读、写等各种操作。PC DOS规定的文件名的格式如下：

文件名.扩展名

其中文件名以英文字母开头的1到8个字符组成，这些字符可以是：

英文字母a～z（不区分大写与小写）；

数字0～9；

某些特殊字符，如，#，@，!，%等。

文件扩展名是跟在文件名后面，由一个句点开始的短名。扩展名可以由一个、两个或三个字符组成，常用来表示文件的类型，它不是必不可少的。例如，NEWFILE.BAS和ABCDE都可以作为文件名，前者有扩展名，后者没有扩展名。

在文件名格式的规定中，使用*和?两个专用字符，称为通配符。字符*用于代表一个任意字符串，例如，TURBO.*表示那些任意扩展名的所有turbo文件。字符?在一个文件名或扩展名中，代表一个任意字符，例如，磁盘中含有ABCDE.XYZ，ABIDE.XXX，ABODE.XZZ三个文件，文件名

AB? DE.X? Z

代表了 ABCDE. XYZ 和 ABODE. XZZ 两个文件。

三、磁盘文件目录

在硬盘和软盘上可以存储大量的文件。通过相应的 DOS 命令能够察看磁盘上文件目录。在 PC DOS 中采用树形结构目录，树形结构目录由最高级根目录和若干级的子目录组成，如图 2.1.2 所示。

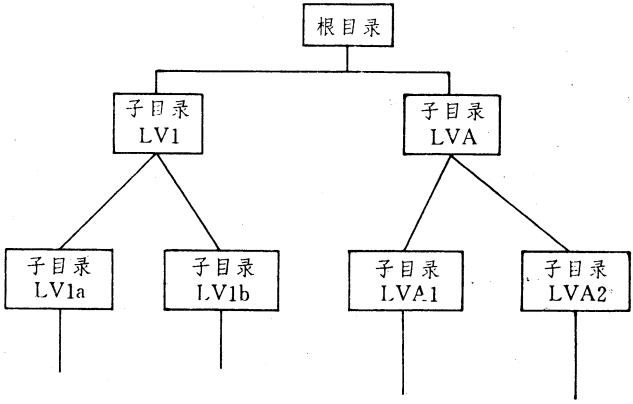


图 2.1.2　磁盘目录的树形结构

一个磁盘文件，既可以存储在根目录中，也可以存储在某一级子目录中。为了定位一个文件（如读、写或显示某一指定文件），除了文件名和文件扩展名外，DOS 还必须知道存储该文件的磁盘在哪个磁盘驱动器里，在根目录中还是某一级子目录中。因此，在文件名前需要写上这些有关信息。这样的信息叫作路径。如果文件存储在根目录中，路径就是一个盘符。盘符的写法是，盘符字母后跟一个冒号。下面是一个完整的文件名的例子：

B：myfile. bas

在磁盘设置有子目录的情况下，一些文件可以存在子目录里。子目录名的名称与文件名的格式一样。多级子目录的表示方法是

\子目录名 1\子目录名 2\・・・

.如果文件 wang. bas 存储在子目录 Lv1 中的子目录 Lv2 中，该盘插在 B 驱动器中，则包括路径在内的完整的文件名应写为

B：\Lv1\Lv2\wang. bas

第二节　DOS 的装入

磁盘操作系统本身也是存储在软磁盘或硬磁盘上的，它的最主要的文件是 COMMAND. COM，IBMBIO. COM 和 IBMDOS. COM（后两个文件在磁盘目录上看不见），此外，它还包括许多命令文件，如 DISKCOPY. COM 和实用程序如 LINK. EXE 等。

使用 DOS 之前，应先把 DOS 装入内存。

对于使用硬盘的微型计算机，操作系统应事先装入硬盘中，只要打开电源，操作系统便会自动装入内存，这个过程称为启动系统。用接通电源的方法启动操作系统的运行，称为冷启动。如果计算机没有硬盘，或虽有硬盘，但想用软盘启动（即装入 DOS），则先要把 DOS 盘片插入 A 磁盘驱动器，然后接通电源，DOS 便可进入内存。

系统启动后，显示器屏幕上出现如下信息：

Current date is Tue 1-01-1980

Enter nuw date：

这时系统等待你打入当前日期，其格式为月-日-年。如果不想输入任何日期，只要按回车键即可，接着又显示出要求输入当前时间的信息：

Current time is：　0：00：32：.57

Enter new time：

这时可以输入当前时间。打入时间的格式为时：分：秒。如果不想输入任何新的时间，则只需按回车键。如果系统是通过 A 盘启动的，这时屏幕显示出

A>

这个符号表示 DOS 已经装入，系统启动成功，当前盘是 A，可以接受命令。所谓当前盘，是指对磁盘进行各种操作时，如果文件名的路径中不包含盘符，则 DOS 认为所省略的盘符为当前盘，换句话说，当前盘的盘符可以省略。

如果系统是通过硬盘启动的，则屏幕显示出

C>

当前盘为 C 盘。

如果机器已经接通电源，重新装入 DOS 的方法是：将〈Alt〉键和〈Ctrl〉键按住，同时再按下〈Del〉键，于是 DOS 调入内存。这个操作过程叫作系统的热启动。

如果当前盘为 A，则文件的存取等项操作都自动的在 A 盘上进行，而在文件名前无需打入盘符 A。但是，如果被操作的文件在当前盘的子目录中，这时文件名前虽然可以不写盘符，但必须有子目录的名字。例如，\Lv1\wang. bas。

如果文件不在当前盘上，则必须在被操作的文件名前写上完整的路径名。例如，当前盘为 C，存有一个名为 zhang. bas 的文件的软盘现在 B 驱动器中，为了读或写这个文件，应该这样写文件名：

B：zhang. bas

其中 B：是磁盘驱动器标识符。

当前盘是可以转换的。若想把当前磁盘驱动器 A 改为 C，操作方法是在提示符 A>的后面打入 B:，接着按回车键，即

A>B:

这时屏幕显示出当前盘符：

B>

不仅可以改变当前盘，还可以由当前盘的根目录进入某个子目录。这时文件名前不但可以省略当前盘符，还可以省略当前子目录名，由于文件名上省略了路径名，从而简化了操作，这方面的内容将在下一节介绍。

操作系统 DOS 装入内存后，便能调入和使用 Quick BASIC 系统了。

第三节　PC DOS 常用命令

DOS 中有很多命令供用户使用，在这一节我们介绍其中一些最常用的。

操作系统的命令可分为内部命令和外部命令两类。外部命令驻留在磁盘上，需要时随时调入。内部命令则随着操作系统一同进入内存并驻留在内存中，使用这些命令时，不再需要从磁盘调入。

一、盘片的格式化（初始化）

新的盘片在使用之前，必须首先对它进行格式化。格式化就是用软件方法，按一定的格式，把盘片划分为磁道和扇区，记录一些初始数据，检查出盘片上损坏不能使用的部分，使之成为 DOS 可存取的记录格式。

格式化用过的旧盘片时，它上面的文件将全部丢失。

格式化命令是磁盘操作系统的一个外部命令，因此，格式化时，在硬盘或插在软盘驱动器中的软盘上必须有文件 FORMAT. COM。

磁盘格式化命令的格式是

[d:] [path] FORMAT d: [/s] [/1] [/8] [/v] [/4]

命令中用方括号括起来的参数（开关）项是任选的，命令中可以不用。选用参数时，不需要打入方括号。这些参数的意义如下：

[d:] [path]　用于指定 FORMAT. COM 文件的路径；

d:　用于指定所要格式化磁盘的驱动器。如果只有一个磁盘驱动器，DOS 将这个驱动器既看成是驱动器 A，又看成是驱动器 B，这时，被格式化盘的盘符在磁盘格式化命令中可以省略。

s/　用于把操作系统文件 IBMBIO. COM，IBMDOS. COM 和 COMMAND. COM 写入被格式化的盘；

/1　当对磁盘进行单面格式化时选用此项开关；

/8　此开关的选用将使每个磁道按 8 个扇区格式化。命令中不带此开关时，每个磁道划分为 9 个扇区；

/v　选用此开关表示要在格式化的盘上写上用户将要输入的卷名。卷名是赋给磁盘的一个标志或名字。

/4　此参数用于在高容量磁盘驱动器上格式化单面或双面 180KB 和 360KB 的软盘。

执行 FORMAT 命令时，显示器上显示出提醒把待格式化的盘片插入相应驱动器的信息。例如，键入命令

FORMAT　B：/s

并按 ENTER（回车键），便是要求用 B 驱动器进行格式化，同时将系统文件拷贝到被格式化的盘上。FORMAT 命令调入内存并运行后，显示如下信息：

Insert new diskette for drve B：

and strike any key when ready

这时，将待格式化的盘插入 B 驱动器，然后按任意一个键，格式化便开始，此时将看到信息

Formating...

过一会，格式化结束，显示变为

Formating... Format complete

System transferred

同时给出有关磁盘空间可使用情况的信息，例如：

362496 bytes total disk space

40960 bytes used by system

321536 bytes available on disk

而且还问是否继续进行格式化：

Format another (y/n)?

根据需要，可以回答 y（是）或者 n（否）。

二、磁盘文件目录显示

我们经常需要查看磁盘文件目录，寻找需要的文件。使用磁盘文件目录显示命令 DIR，可以在显示器的屏幕上显示出磁盘上的全部文件或指定的部分文件，这是一个 DOS 内部命令（调用此命令时，DOS 盘可不在驱动器中），其格式为

DIR [d:] [path] [文件名 [扩展名]] [/p] [/w]

其中 d 为盘符，它的缺省值（盘符省略）为当前盘。[path] 为被显示文件的路径，命令中不含文件名时，显示给定路径的全部文件。

有三种目录显示格式。当命令中不含开关/p 和/w 时，每行只显示一个文件，每行显示的内容依次为文件名，扩展名，文件的长度（字节数），建立文件的月、日、年和时间。在目录的最后一行，显示文件数和剩下的空闲可用的磁盘容量（字节数）。

例如，要显示 B 磁盘驱动器中盘上的根目录，可用命令

DIR b:

下面是一个显示根目录的例子：

```
Volume in drve B has no label
Directory of B: \

COMMAND     COM  23791    12-30-85   12: 00p
TURBO       EXE  56321     5-02-89    5: 50a
TURBO       TPL  44352     5-02-89    5: 50a

    3 File (s)          91136 bytes free
```

显示的第一行说明该盘在格式化时没有赋给卷名。

使用/w 开关时，显示第二种目录格式。这种显示格式是只显示每个文件的文件名和扩展名，并且每行显示五个文件。例如，我们用第二种显示格式查看上述 B 盘根目录的文件名，可用下面的命令：

DIR b：/w

其显示内容如下：

Volume in drive B has no label

Directory of B：\

COMMAND COM TURBO EXE TURBO TPL

3 file (s) 91136 bytes free

命令 DIR B：TURBO.*/W 可给出如下的显示：

Volume in drive B has no label

Directory of B：\

TURBO EXE TURBO TPL

2 File (s) 91136 bytes free

当盘内的文件很多，屏幕容纳不下时，可使用/p 开关。这时文件目录将逐屏显示，一般情况下，每屏显示 24 行，按任意一个键，就继续显示下一屏幕。

三、删除文件

用户需要从磁盘上删除一个或若干个文件时，可用 DOS 内部命令：

ERASE [d:] 文件名 [. 扩展名]

在文件名或扩展名中用 * 号或？号，可删除一些文件。也可以使用下面的命令，其功能是一样的：

DEL [d:] 文件名 [. 扩展名]

例如删除 B 盘（设盘缺省值为 A）上的文件 wag. bas，其命令为

DEL b：wag. bas

四、子目录的建立、进入、退出和删除

1. 建立子目录

如果想把文件存入或拷贝（复制）到一个子目录中，而现在

盘上并没有所需要的子目录，则要先建立子目录，然后才能把文件写入该子目录中。

建立子目录的命令是

MD [d:] \子目录名

而显示子目录的命令是

DIR [d:] \子目录名

根据目录的树形结构，可以在子目录中建立子目录。建立子目录的命令如下：

MD [d:] \子目录名 1 \子目录名 2

2. 进入和退出子目录

用户启动 DOS 时，是处在根目录，即使建立了一个子目录，它也不会自动地变为当前的工作目录，因此，在寻找那个子目录中的文件时，文件名前必须带有路径。

我们也可以把子目录名设置为当前工作目录（路径缺省值），使得文件操作时把子目录名能够略去。这个命令的格式是

CD [d:] \子目录名 1 \子目录名 2...

要返回根目录，可用命令

CD\

3. 删除子目录

在清除子目录之前，首先要删除该子目录中的全部文件，然后退出该子目录，再用 RD 命令进行删除。命令格式如下：

RD [d:] \子目录名

在以下的命令格式中，必要时，文件名都包含路径名。

五、文件的复制和核对

文件的复制分为个别文件的复制和整个磁盘文件的复制两种情况。如果想要了解复制的文件或整个盘是否与原来的文件或盘完全一致，还可以用相应的命令进行检查核对。

1. 软磁盘的复制与比较

(1) 复制软磁盘

盘复制（拷贝）命令是 DOS 的一个外部命令，其格式为

[d:] [path] DISKCOPY [d:] [d:] [/1]

命令中的 [d:] [path] 指定 DISKCOPY 命令文件的磁盘驱动器和路径。第一个盘符 [d:] 指定源盘驱动器，第二个盘符 [d:] 指定目的盘驱动器。命令的功能是将源盘的文件全部复制到目的盘。当使用开关/1 时，表示只复制盘的第一面，否则复制两面。

如果目的盘事先没有格式化，则这个命令将先进行格式化，然后进行复制。在命令执行过程中，有信息告诉源盘应插入哪个驱动器，目的盘应插入哪个驱动器。复制完成后，显示信息

Copy complete

并且询问，是否继续使用这个命令进行复制：

Copy another (y/n)?。

回答 N 或 Y。

下面是一个只使用 A 盘驱动器进行盘复制的例子。首先打入命令：

A〉diskcopy

然后按回车键，于是显示出如下信息：

Insert source diskette in drive A:

Strike any key when ready

将源盘插入 A 驱动器后，按任意键，继续工作。这时显示如下信息：

Copying 8 sectors per track, 2 side (s) 过一会，出现下面的信息：

Insert traget diskette in drive A:

Strike any key when ready

这时，将源盘从驱动器 A 取出，将目的盘放入 A 驱动器，按回车键后开始拷贝。当拷贝结束时，显示信息

Copy complete

Copy another (y/n)?

如果不继续复制，键入 n 并按回车键，便返回 DOS 命令状态。

(2) 盘比较

用磁盘比较命令 DISKCOMP 可以核对所复制的目的盘是否与源盘一致，这也是 DOS 的一个外部命令，其格式是

[d:] [path] DISKCOMP [d: [d:]] [/1]

命令中的开关 [/1] 表示仅比较软盘的第一面。

例如，打入命令

A>diskcomp a: b: /1

于是显示信息

Insert FIRST diskette in drive A:

Insrt SECOND diskette in drive B:

Strike any key when ready

按照上述提示，将源盘插入 A 驱动器，将目的盘插入 B 盘驱动器。最后按一任意键，比较便开始进行，同时显示信息

Comparing 8 sectors per track, 1 side (s) 如果比较结果没有问题，则给出信息

Diskette compare ok

最后，询问是否还要进行比较，这同上一个命令的情形一样。

2. 个别文件的复制与比较

(1) 某个或某些文件的复制

当需要复制一个或几个文件到指定的磁盘时，可使用命令 COPY，它是一个 DOS 内部命令，其格式为

COPY [d:] [path] 文件名 [. 扩展名] [d:] [path] [文件名 . [扩展名]]

COPY 命令后面依次为被拷贝的源文件盘符、路径和文件名，目标文件盘符、路径和文件名。

例如，将当前盘 A 上的 wag. bas 文件以相同的文件名和扩展名复制到 B 盘，其命令为

A＞COPY A：wag. bas B：wag. bas

文件被复制后，显示信息

1 File (s) copied

在上面的命令中，当前盘的盘符A：可以省略，又由于两个文件名相同，复制的目标文件名也可以省略，于是上面的命令简化为下面的形式：

COPY wag. bas B：

如果在同一个盘上复制文件，目标文件名必须不同于源文件名。但是，若不是在当前盘上复制，盘符不能省略。

我们知道，文件名和扩展名中的＊号能代表任意的字符串，因此，COPY 命令中使用这个符号可以同时复制一些文件。例如命令

A＞COPY ＊. ＊B：

可将当前A盘上的所有文件，以相同的文件名和扩展名，复制到B盘上。

(2) 文件的比较

文件被复制后，它们的一致性可用命令COMP来检查。COMP命令是DOS外部命令，命令的格式为

COMP [d：] [path] [文件名] [. 扩展名] [d：] [path] [文件名] [扩展名]

例如，比较A，B两个盘上的同名文件wag. bas，其命令为

A＞COMP wag. bas b：wag. bas

六、文件的输出

当需要查看文件内容时，可用TYPE命令，将文件输出到打印机打印出来，或者输出到显示器显示出来。只有用ASCII码存储的文件，这个命令才能以可读的形式输出。TYPE是DOS的内部命令。

在显示器屏幕上显示文件内容时的命令格式为

TYPE [d：] [path] 文件名 [. 扩展名]

将文件输出到打印机打印时的命令格式为

TYPE [d:] [path] 文件名 [. 扩展名] >PRN

其中 PRN 代表打印机

小　　结

本章简要地介绍了关于磁盘操作系统(DOS)的基本知识和在开发 BASIC 程序过程中常用的一些 DOS 命令。本章中提到的内容是最基本的，是学习本课程的起码要求。在许多情况下，需要掌握更多的操作系统方面的知识，因此，必要时请参看有关的参考书。本章讲到的一些 DOS 命令，要求能掌握它们的功能并能正确的使用。

习　　题

2-1　什么是磁盘操作系统？什么是冷启动和热启动？

2-2　为什么要对磁盘格式化？怎样进行磁盘格式化？

2-3　上机练习 DOS 常用命令的使用方法。

第三章　对 Quick BASIC 编程环境的初步认识

本章首先介绍如何安装 Quick BASIC。然后，通过对 Quick BASIC 的简单使用，初步了解 Quick BASIC 的程序设计环境。

学完本章之后，将知道怎样在硬盘上或软盘上安装 Quick BASIC 软件；怎样启动 Quick BASIC 和如何初步利用 Quick BASIC 环境，进行简单的程序操作，如程序的编辑、装入、运行、存储以及将源程序编译成机器语言的“可执行文件”（扩展名为 EXE 的文件）等。

第一节　Quick BASIC 系统的安装

一、Quick BASIC 系统的基本软件

Quick BASIC 系统主要包括下列文件：

QB. EXE　Quick BASIC 的程序开发环境。用于在存储器中建立、调试和运行 BASIC 程序，以及建立程序库。还用于建立可在 DOS（操作系统）状态下独立运行的可执行文件（. EXE 文件）

BC. EXE　编译程序。用于在 Quick BASIC 环境或 DOS 状态下建立可执行文件或用户程序库。

BRUN40. EXE　运行模块。它的用途是运行用 BRUN40. LIB 建立的可执行文件。

BRUN40. LIB　运行模块库。用于建立可执行文件。

LIB. EXE　独立库管理程序。用于建立与可执行文件连接的用户独立库。

LINK. EXE　连接程序。用于在 Quick BASIC 环境和 DOS 下建立可执行文件、用户 Quick 库和独立库。

BCOM40. LIB　库文件。用该库建立的可执行文件，在运行时，无需 BRUN40. EXE 文件的支持。

BQLB40. LIB　库文件。用于建立用户 Quick 库的支援例程。

QB. HLP　联机帮助文件

二、系统的安装

以上系统软件分别存在三张软磁盘上。使用时，应把它们拷贝到自己软盘上或硬盘上，配上操作系统，使之作好系统运行前的准备工作。这就是系统的安装。Quick BASIC 可以在 PC DOS 2. 0 或更高版本的操作系统支持下工作。

1. 在硬盘上安装 Quick BASIC

如果使用的微型计算机配有硬盘，可以把上述全部软件拷贝到硬磁盘（盘符为 C）上。既可以把全部文件拷贝到一个目录或子目录里，也可以把不同类型的文件分别拷贝到不同的子目录里。

可以用第一张盘上的 SETUP. BAT 把 Quick BASIC 拷入硬盘。不用 SETUP. BAT，使用上一节介绍的 COPY 命令也能安装 Quick BASIC。

例如，在使用子目录的情况下，将 EXE 文件拷贝到 bin 子目录里，LIB 文件拷入 lib 子目录里。为此，首先要用 MD 命令建立相应的子目录，然后用 COPY 命令把所需要的文件拷入。在使用系统之前，最好先设置环境变量，以便告诉系统到何处去找所需要的文件。

设置环境变量的方法是，用 PATH 命令告诉系统寻找 EXE 可执行文件的路径，如

PATH C：\bin

用 SET 命令可以告诉系统在何处可找到 LIB 库文件，如

SET LIB＝C：\lib

如果在使用系统之前，不设置环境变量，系统可能找不到所需要的文件，这时，会给出询问有关文件路径的信息。

例如，当系统需要 BC. EXE 文件而找不到时，会显示出如下的信息，并要求输入 BC 文件的路径：

Cannot find file（BC）. Input path：

这时只要在冒号后面键入相应文件的路径，如

C：\bin

程序便可继续往下进行。

2. 在软盘上安装 Quick BASIC

如果使用的微机有两个软盘驱动器，没有或不用硬盘，那么，要用 COPY 命令把原系统盘上的 Quick BASIC 系统软件拷贝到自己的软盘上。建议将 Quick BASIC 分别复制到如下的三张软盘上：

第一号盘　COMMAND. COM
　　QB. EXE
　　QB. HLP

第二号盘　BRUN40. EXE
　　BRUN40. LIB
　　BQLB40. LIB
　　LINK. EXE
　　LIB. EXE
　　BC. EXE

第三号盘　BCOM40. LIB
　　LINK. EXE

另用一张（第四号盘）软盘存储用户程序和用户库。使用上述盘配置时，系统盘放在 A 驱动器，用户盘放在 B 驱动器。可以从 B 盘启动（调用）Quick BASIC，也可以从 A 盘启动。

为从 B 盘调用 Quick BASIC，把一号盘放入 A 驱动器，四号盘放入 B 驱动器，然后进行下列操作：

（1）键入盘符 b：并按 ENTER（回车）键，使 B 盘为当前盘

(盘符缺省值)。

(2) 键入 a：qb，以调用 Quick BASIC 中的文件 QB. EXE (对于黑白监示器最好用 a：qb/b 调用)。

第二节　Quick BASIC 屏幕

用上节最后的操作调出 Quick BASIC 后，屏幕显示如图 3. 2. 1 所示，也称为编辑屏幕。这是我们的主要工作环境，因此，首先要熟悉它。参照图 3. 2. 1，将屏幕的主要部分解释如下：

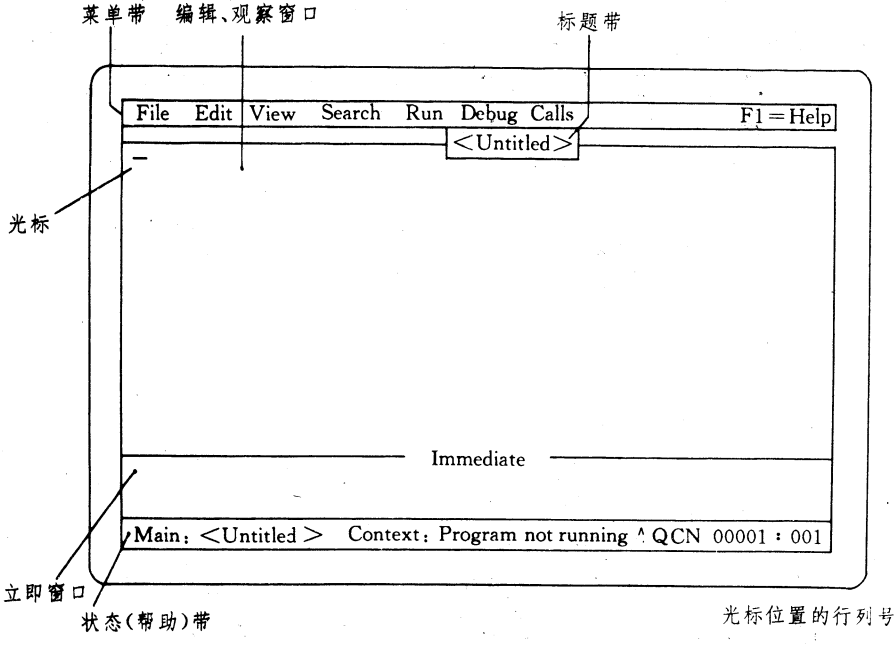


图 3. 2. 1　Quick BASIC 屏幕

1. 菜单带 (Menubar)

在这个菜单中，为用户提供了八种功能选择，供操作程序用。这八种功能是：

File　用于调用与文件操作有关命令的子菜单。

Edit　用于调用与编辑文本有关命令的子菜单。

View　用于调用与显示文本有关命令的子菜单。

Search　用于调用与查询文本中字符串有关命令的子菜单。

Run　用于调用与运行程序有关命令的子菜单。

Debug　用于调用与调试程序有关命令的子菜单。

Calls　用于显示主模块名和所调用的过程名。

Help　用于提供联机帮助。

每个菜单名字中高亮字符表示选择相应菜单时需要按的键。选某个菜单命令的操作步骤是：

（1）按＜Alt＞键；

（2）按菜单带中相应名字的高亮字符键，于是屏幕上显示出子菜单。

另一种操作方法是：

（1）按＜Alt＞键；

（2）按方向键（→或←键），以选择所需要的菜单；

（3）选到后按回车键，显示相应的子菜单。

2. 观察窗口（View window）

这部分屏幕用于输入和显示存储器中的 BASIC 源程序。

3. 立即窗口（Immediate window）

在此窗口用户能立即执行一些 BASIC 语句，这对检查和调试程序是很有用的。按 F6 键可使光标在观察窗口和立即窗口之间转移，从而实现在相应窗口中的操作。光标所在的窗口叫作活动窗口。

4. 光标（Cursor）

它指示当前输入文本字符的位置。输入一个字符后，光标自动向右移动。

5. 标题带（Title bar）

这里显示当前在观察窗口显示的文件名，如果没有命名，则

为〈untitled〉。

6. 状态带（Status bar）

在状态带中显示程序的主模块名，光标在观察窗口或立即窗口位置的行列号，以及其他状态信息。

第三节　Quick BASIC 程序的基本操作

一、用户程序的输入

在初步了解了 Quick BASIC 编程环境后，就可以开始在这个环境中开发程序了。

首先我们看如何把已经编好的程序输入给计算机，假设有下面的程序例题：

例 3.3.1　已知圆半径为 2.5 米，试编写计算此圆的周长和面积的程序。

解此问题的 BASIC 程序如下：

```
Pi=3.14159                                    ' Pi=π
Radius=2.5                                    ' Radius=半径
Area=Pi * Radius * Radius                     ' Area=圆面积
Circumference=2 * Pi * Radius                 ' Circumference=
                                              ' 圆周长
PRINT"Area=";Area                             ' 输出计算结果
PRINT"Circumference=";Circumference
END                                           ' 程序结束
```

在程序的每一行后面（单引号之后），都给出了对该程序行语句意义的简单说明。关于程序编写的方法，以后还要详细介绍，但是，只要读者知道计算圆周长和圆面积的公式，上面的程序是不难看懂的。

现在，我们要在 Quick BASIC 编程环境下，把这个程序输入到

计算机的内存。为此，首先启动 Quick BASIC。Quick BASIC 被启动后，显示器便显示出前面介绍的屏幕样式，这时系统实际上已经处于编辑状态，即可以进行程序的输入与修改等编辑工作。

输入程序的方法很简单，因为系统已经处在编辑状态下，所以，在键盘上打入的任何字符，Quick BASIC 都认为写入的是 BASIC 语言的语句，因此，只要按照例 3.3.1 中的程序清单，将字符逐个打入就行了。

输入程序时，用在行尾按回车键（ENTER）或者用方向键（箭头键）把光标移出当前行的方法完成一行的输入。

在通过键盘输入源程序的过程中，难免按错键。为了修改错误，最常遇到的问题是需要插入字符、删除字符，插入行和删除行等，这类工作称为编辑。

插入和删除字符的方法很简单。当需要删除某个字符时，用方向键把光标移到待删除字符的位置，然后按＜Del＞键，该字符就被删掉了。

如果要在某处插入字符，同样用方向键移动光标，把光标移到插入位置的右侧字符处，打入所要插入的字符即可。

如果想要删去正在输入的字符（光标尚未离开当前行），可利用退格键。每按一次退格键，便删除光标前的一个字符，同时光标向左移一个格。连续按退格键，可连续向左删除字符。

用方向键把光标移至要删除行上，按＜Ctrl＞和 Y 键（先按住 Ctrl 不放，接着按 Y），该行即被删除。

用方向键将光标移至某行的行首，按＜Ctrl＞和 N 键，便在该行的前面出一空行，于是在这行可写入新的程序行，也就实现了插入行。

为了方便快速地修改错误，Quick BASIC 系统提供的编辑功能是相当强的。我们已经知道，Quick BASIC 提供给用户的是一个集成环境，它把文本编辑器、编译程序和调试程序一起放在一个灵巧的编辑器（Smart editor）中。要想了解它许多完善的文本编辑功

能，请参阅有关手册，这里就不介绍了。

需要指出的是，用户每输入完一行，Smart 编辑器都完成下列工作：

（1）检查行中有无语法错误，如有，将指出错误的性质。例如，用户错误地输入了语句

value＝@35

系统经语法检查后，将指出@35 处有错误，其性质为

Expected：expression

即@35 处应该是表达式。这时，用户可通过按＜Esc＞键，返回编辑屏幕，进行相应的修改。

（2）经语法检查后，如果输入行没有错误，便对该行格式化。格式化后，所有的 BASIC 的关键字（即表示语句操作命令的英文字符）都以大写字母的形式出现，操作符的前后出现空格；变量名和过程名中的大小写字母保持恒定，如果用户在某处改变了一个变量的大小写法，则该变量的各行的写法同时改变；还在适当的地方加标点（如果用户忽略了的话）。

例如，用户输入一条显示 value1 和 value2 两个变量值的语句

Print value1 value2

在按回车键后，关键字 Print 格式化为 PRINT，两个操作符之间加入分号，最后格式化的语句形式为

PRINT value1；value2

又如，输入语句：let Value1＝35，则格式化后的形式是

LET Value1　＝　35

即关键字 let 改为大写，在等号的前后都加入了空格。

（3）如果该行语法正确的话，就把这行翻译成为可执行的形式，这就是说，键入的每一行都是准备好了可以立即执行的。

二、程序存入磁盘和从磁盘装入内存

现在程序已经通过键盘输入内存。程序通常是以文件的形式

保存在磁盘中，当需要时则把它从磁盘取出装入内存，以便运行（执行）这个程序。

在 Quick BASIC 环境下，读写磁盘文件（程序）是通过 File 的子菜单命令实现的。调用 File 子菜单的操作步骤如下：

（1）首先按＜Alt＞键，进入菜单带，这时" File" 变成高亮度显示，表示它被选中。

（2）按 F 键或＜回车＞键以显示 File 菜单的所有命令（子菜单）。在这个子菜单中，有下列一些命令：

New Program

Open Program • • •

Merge • • •

Save

Save As • • •

Save All • • •

Create File • • •

Load File • • •

Unload File • • •

Print • • •

Dos Shell

Exit

在每个命令中都有一个高亮字符，按相应的高亮字符键，便可以选相应的命令。也可以通过方向键↑和↓，将光带移到相应命令上，然后按回车键，选择相应命令。

1. 文件写入磁盘

在上述菜单中，用于将程序写入磁盘的命令有 Save，Save As 和 Save All，当前可以使用前两个命令中的任何一个。如果程序已有文件名（该程序是从磁盘读入内存的），现在仍用这个文件名存盘，这时用 Save 命令就可以了。如果打算更换文件名，就必须用 Save As 命令。Save As 命令要求用户输入文件名。

现在假定程序刚刚输入，还没有文件名，应该使用 Save As 命令，但是用 Save 命令也可以，因为程序尚无文件名，系统会提醒输入文件名。Save 或 Save As 命令被选中后，屏幕显示如图 3.3.1 所示的对话框。

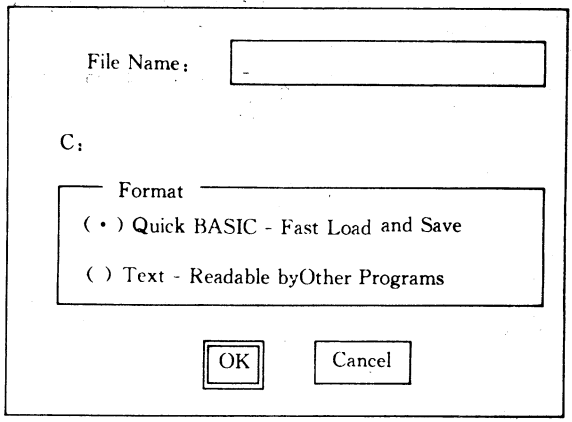


图 3.3.1　程序写盘对话框

图中 File Name 框里填写文件名，然后按<Tab>键，将光标移至 Format 框，进行文件存储格式的选择。

有两种文件存储格式：第一种格式是用系统的内码存储文件，在这种情况下，读写磁盘较快；第二种格式是用 ASCII 码存储文件，这种格式也称为文本格式。用文本格式存储的文件叫作文本文件，这种文件对于其他程序来说，是可读的。例如，用操作系统的 TYPE 命令就能够以可读的形式将文件显示或打印出来。但是，用这种格式读写磁盘较慢，并且占用的存储空间较大。

存储格式的选择步骤是：

(1) 用方向键（↑和↓）将光标移至 Format 框中的相应圆括号内；

(2) 按回车键，程序写入磁盘。

由于对话框会自动选择第一种存储格式，因此，用内码存储

文件时（BASIC 源程序一般用这种格式），就不必进行上述存储格式选择的操作。填写文件名后按回车键，程序便开始写入磁盘，在写入期间，状态带显示“Saving”字样。

如果在按回车键之前，想要终止当前的程序存盘操作，可通过按<Tab>键，将光标移至 Cancel 框（这时 Cancel 框将变为双线框），然后按回车键，于是返回编辑屏幕。按<Esc>键，同样可返回编辑屏幕。

2. 磁盘文件装入内存

在 File 子菜单中，Open Program 和 Load File 两个命令可用于读磁盘文件。

Open Program 用于往内存装入一个新程序，如果这时内存中有其他程序（该文件自装入后没有被编辑修改过），则内存中原来的程序将被新文件代替。

Open Program 被调用后，显示如图 3.3.2 所示的对话框。对话

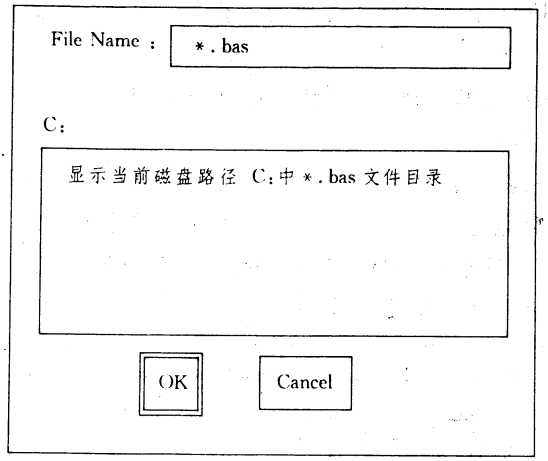


图 3.3.2　Open Program 对话框

框中的第一栏 File Name 是填写需要装入文件名的地方。键入文件名后按回车键，磁盘上相应文件便开始读进内存，同时，在屏幕

的状态带显示“Loading and parsing”。程序成功的装入内存后，屏幕的标题带显示出读入的文件名，这时在屏幕上的观察窗口可以看到文件的内容。

对话框的第二栏内，显示着当前磁盘路径下的文件目录。当前磁盘路径显示在本栏的上面，如果在这个目录中有你所要的文件，你可以不在 File Name 栏打入文件名而进行如下的操作：

（1）按＜Tab＞键，将光标移至目录栏；

（2）用方向键（↑ ↓ →←四个键）将光标移到所要装入的文件名上；

（3）按回车键，完成读磁盘操作。

如果你的文件不在当前的目录中，可以通过在 File Name 栏打入正确的盘符或路径名来改变当前磁盘路径，从而改变目录栏内的显示内容。或者在 File Name 栏内，直接打入带有路径的文件名，文件也能读入内存。

按＜Esc＞键可中断正在进行的操作。同样的效果还可以用下面的操作达到：

（1）按＜Tab＞键，使光标移至 Cancel 框；

（2）按回车键。

当读入新磁盘文件时，内存中已有程序存在，并且对该程序已进行编辑但还没有存入磁盘，则系统通过显示如图 3.3.3 所示的对话框，提醒是否要将内存中的程序存入磁盘，只有得到回答后，才能装入现在所要的程序。

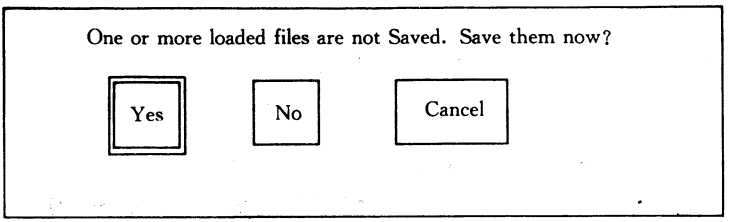


图 3.3.3　读取文件的对话框

对话框提供三种选择：

（1）当前 Yes 框被选（因它是双线框），按回车键后，内存中的程序写入磁盘，接着新文件进入内存；

（2）按＜Tab＞键，使光标移至 No 框，按回车键后，原内存中程序被清除，新文件进入内存；

（3）按＜Esc＞键，以中断读盘命令，或者按＜Tab＞键，使光标移至 Cancel 框，接着按回车键，以中断当前命令，然后返回编辑屏幕。

有时在读入新文件之前，调用 New Program 命令，来清除内存中原有的程序，这时，如果内存中的程序经过修改尚未存盘，也会出现图 3.3.3 所示的对话框。

Load File 命令也能把磁盘文件装入内存，但它是把文件作为一个程序模块（Module），包含文件（Include，关于包含文件将在以后介绍）或文本文件（Document）装入内存，而已在内存中的程序，将继续留在内存中。这里，文本文件指的是用 ASCII 码存储的任何文件，包括非程序文件。现在我们是要装入 BASIC 程序，应该把文件作为一个模块来装入。

Load File 命令被调用后，屏幕显示如图 3.3.4 所示的对话框。

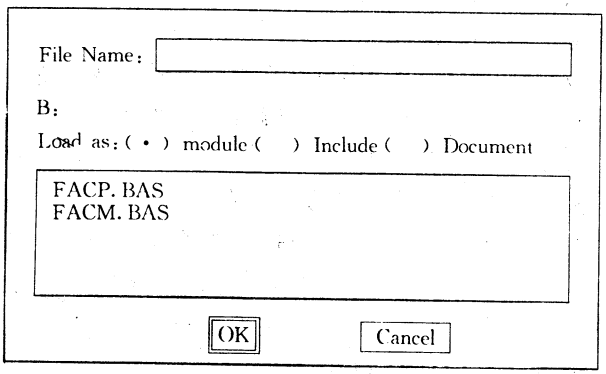


图 3.3.4　Load File 对话框

在 File Name 框内打入所要读取的文件名。Load as 行的（·）符号表示当前要读入文件的类型。文件类型可以按下述操作步骤进行选择：

（1）按＜Tab＞键，将光标移至 Load as 行；

（2）用方向键将光标移到所要求的文件类型（Module）；

（3）按回车键，所要求的程序便可读入内存。

文件目录框内显示着当前盘路径中的文件名，它们可以象前面介绍的那样（见 Open Program 命令的对话框），用来选择所要读取的文件。

三、BASIC 程序的调式与运行

程序经过设计、输入和存盘等一系列操作后，就为程序的运行（执行）作好了准备。虽然，在程序的输入过程中，系统对输入的语句进行了语法检查，并且根据所指出的错误性质，修改了相应的语句，但在程序运行中，还会发现一些错误，通过试运行来发现程序中的错误并进行改正，称为程序的动态调试，简称调试。

为了调试和运行一个程序，而该程序当前又不在内存中，则首先应把该程序从磁盘调入内存。

1. BASIC 程序的运行

现在假设例 3.3.1 的程序已在内存，这时在观察窗口可以看到该源程序清单。为运行程序需要调用菜单带中 Run 命令的子菜单命令，为此，要进行下列操作：

（1）按＜Alt＞键，使光标移至菜单带；

（2）按方向键或字母键 R，以选择菜单带中 Run 的子菜单；

（3）显示出子菜单后，按 S 键以选用运行程序的 Start 命令，或者用方向键移动子菜单上的光带来选择 Start 命令；

（4）按回车键，程序开始运行。

程序的运行结果，显示在输出屏幕上，例如，例 3.3.1 的运

行结果，在输出屏幕显示

Area＝19.6394

Circumference＝15.7095

按任意一个键，便可返回 Quick BASIC 编辑屏幕，也可以用＜F4＞键在编辑屏幕和输出屏幕之间进行切换。

2. 程序的调试

Quick BASIC 系统提供使用方便、功能很强的程序调试手段。通过调用菜单带的 Debug，可以得到它的子菜单。使用子菜单中的命令，可以很容易地设置程序执行中的断点，进行动态跟踪，单步执行，观察程序运行中变量或表达式的值等。

调试工具的使用，需要掌握更多的程序设计知识和较高的操作技能，因此，关于程序调试工具的使用，不在这里介绍。

四、BASIC 程序清单的显示与输出打印

1. 程序清单的显示

在 Quick BASIC 编程环境下，编辑屏幕的观察窗口总是显示着当前输入至内存的程序。但是，一般情况下（不改变编辑窗口的尺寸时），观察窗口只能显示 18 行程序，未显示在当前窗口的那部分程序，可以通过下述的操作，将其移入观察窗口。

利用方向键（↑和↓）操作。

当光标在窗口的第一行时，每按一次↑键，屏幕显示便下滚一行，从而整个屏幕往前显示一行。在光标处于最下面一行的情况下，每按一次↓键，屏幕显示往上滚一行，于是整个屏幕往后显示一行。通过上述操作，可以从有限的观察窗口看到全部程序。

使用＜PgUp＞键和＜PgDn＞键操作。

按＜PgUp＞键，可以把当前整个观察窗口显示的内容，更换为它前面的一页。按＜PgDn＞键，则把当前整个观察窗口显示的内容，更换为它后面的一页。可见，无论程序有多么长，通过操作＜PgUp＞键和＜PgDn＞键，同样可以把他们依次地显示在屏幕

上的观察窗口。

2. 打印程序清单

在打印机上输出程序清单，需要调用 File 子菜单中的 Print 命令。其操作过程如下：

(1) 按<Alt>键或方向键，以选择菜单带中的 File；

(2) 按 F 键或回车键，以显示 File 的子菜单；

(3) 按 P 键，以选择 Print 命令，或通过按方向键，将菜单上的光带移动到 Print 行并按回车键；

这时屏幕显示出如图 3.3.5 所示的对话框。由这个对话框可以看出，系统为用户提供了下列四种打印选择：

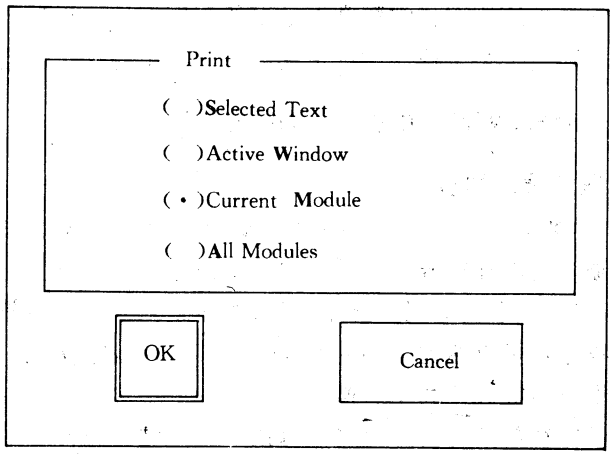


图 3.3.5 Print 对话框

(1) Selected Text　　打印所选择的文本；

(2) Active Window　　打印工作窗口的文本；

(3) Current Module　　打印当前程序模块；

(4) All Modules　　打印输入内存的所有程序模块。

这里只介绍其中常用的第一和第三种。

由图 3.3.5 可知，Print 命令被调用后，自动选择第三种打印情况，即打印当前程序模块（该模块显示在观察窗口）。例如，现

在例 3.3.1 的程序已装入内存，那么，只要按回车键，该程序清单便被打印出来。

如果只想打印当前程序模块的某一部分，则必须在调用 Print 命令之前，把要打印的部分选择出来，这叫选择文本。选择好文本后才能调用 Print 命令。

选择文本是在观察窗口上进行的。首先将光标移到所要选择的行，然后通过按<Shift>键并同时按方向键，选择具体的字符，按 Shift+Right 键，选择光标右边的字符，按 Shift+Lift 键，选择光标左边的字符。被选文本呈高亮度显示。按任一方向键，可把文本恢复为未选状态。按 Shift+Up 键，可选光标前一行，按 Shift+Dn 键，可选光标所在行。

所要打印的文本部分选择好后，按前述操作过程调用 Print 命令，用方向键或 S 键在 Print 对话框选择打印 Selected Text 的方式，最后，按回车键，打印便开始。

五、退出 Quick BASIC

当操作完 BASIC 程序，需要返回 DOS 时，有两种退出 Quick BASIC 环境的方法，这就是 File 子菜单中的 Dos Shell 和 Exit 两个命令。

Dos Shell 命令是临时返回 DOS 命令层，Quick BASIC 仍保留在内存中。返回 DOS 后，可以执行各种 DOS 命令。键入 EXIT 并按回车键，便又从 DOS 命令层返回 Quick BASIC，屏幕显示恢复原样。

执行 File 子菜单中的 Exit 命令时，内存中的 Quick BASIC 将被清除并立即返回 DOS。但是，在执行 Exit 命令时，如果内存中还有新的程序或修改过的程序尚未存盘，Quick BASIC 将显示一个如图 3.3.3 所示的对话框。询问是否要将文件存盘，可有以下三种选择：

(1) 按回车键，以其当前状态将该文件存入磁盘，然后从内

存清除 Quick BASIC 并返回 DOS；

（2）按 N 键或用＜Tab＞键将光标移至 No 框，然后按回车键，这时不保存文件，而是直接清除 Quick BASIC 后返回 DOS；

（3）按＜Esc＞键或用＜Tab＞键，将光标移至 Cancel 框。按回车键，于是返回 Quick BASIC 环境，即不退出。

第四节　在磁盘上建立可执行文件

一旦程序能在 Quick BASIC 环境下正确运行，我们便可构造一个不需 Quick BASIC 支持，可在 DOS 下直接运行的程序副本。这个程序是用机器语言表示的，称为可执行文件，其文件扩展名为 .EXE。在 DOS 提示符下，键入文件名（无需扩展名），即可运行它。

可在 DOS 状态下直接运行的 EXE 独立文件，有更高的运行速度。

在建立一个可执行文件时，首先需要用编译程序（BC.EXE）将 BASIC 源程序编译成一个以 .OBJ 为扩展名的目标文件，它是一个机器语言程序，但尚不能执行。目标文件建立之后，使用连接程序（LINK.EXE）将目标文件模块和正确运行程序所需要的一些库文件（BRUN40.LIB，BCOM40.LIB 等）连接起来，以形成一个可执行文件。所建立起来的目标文件和可执行的独立文件，都直接存储在磁盘上。

因为在 DOS 状态下运行可执行文件，要比在 Quick BASIC 环境下运行内存中解释编译的程序快得多，所以，对于一些大型的、需要较长时间运行的程序，建立并运行可执行文件，是很有必要的。

可执行文件可以在 Quick BASIC 环境下建立，也可以在 DOS 状态下建立。在这一节，只介绍在 Quick BASIC 环境下建立可执行文件的操作方法。

Quick BASIC 系统可以建立两种可执行文件。一种是能在 DOS 状态下完全独立运行的可执行文件，另一种是在 DOS 状态下需要一个称为运行时模块（BRUN40. EXE）支撑才能运行的可执行文件。前者，在编译过程中需要 BCOM40. LIB 库文件的支持，形成的可执行文件较长；后者需要 BRUN40. LIB 库文件的支持，可执行文件较短。

在建立可执行文件的过程中，当需要用到上面提到的某个文件时，系统要确认一下，所需文件是否在当前目录中，或者通过事先设置的 Path 路径及 LIB 环境变量找到。当找不到所要求的文件时，将显示出信息，要求输入读取相应文件的路径。例如，在进行编译时找不到 BC. EXE 文件，则有如下的显示：

Cannot find file (BC) . Input Path：这时把含有 BC. EXE 文件的磁盘插入某驱动器（如果该文件不在任何驱动器的话），然后键入盘符和路径，编译即可进行下去。例如，键入 a：或 c：\bin：

Cannot find file （BC） . Input Path：a：

Cannot find find （BC） . Input Path：c：\bin

在 Quick BASIC 环境下构造可执行文件的步骤如下：

（1） 按＜Alt＞键和 R 键，以显示 Run 的菜单；

（2）按 X 键，以选择用于建立可执行文件的 Make EXE File 命令。如果 BASIC 源程序尚未存盘，会出现如图 3. 3. 3 所示的对话框，提醒把源程序存盘，这时按 Y 键，便可以把源程序存入磁盘。如果源程序已经存盘，则将出现如图 3. 4. 1 所示的建立可执行文件的对话框。

从这个对话框可以看到，在 EXE File Name：的后面自动地给出一个与源程序文件名相同，但扩展名为 . EXE 的可执行文件名，如果用户同意将这个文件名作为自己的可执行文件名，就不必再输入可执行文件名了，这时，只要按回车键，编译便开始进行。否则，用户可以输入自己的文件名。

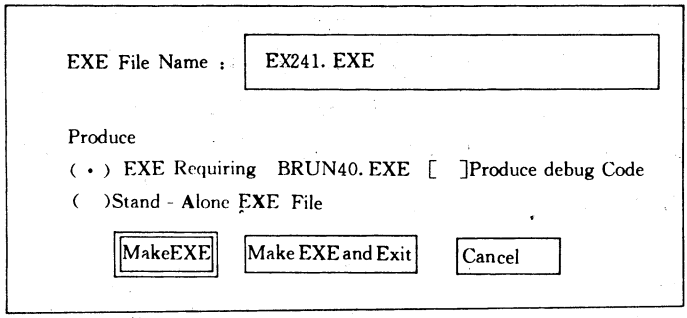


图 3.4.1 Mak EXE 文件对话框

在对话框 Produce：中有三项功能待用户选定：

①EXE Requring BRUN40. EXE 建立使用运行模块 BRUN40 . EXE 的可执行文件；

②Stand-Alone EXE File 建立自己能独立运行的可执行文件；

③Produce Debug Code 程序运行时有错误检查功能。主要检查算术溢出，数组下标越界等。在每行程序执行完毕，还能检查是否有 Ctl＋Break 键输入。Ctl＋Break 组合键能使程序停止运行。

其中第一项是自动选中的（双线框）。如果用户想建立的就是使用运行模块的可执行文件，就不必改变它了。如果不需要提供运行时的错误信息，就不需为此进行任何操作。

用户需要在 Produce 的三项功能中作某种选择时，要继续进行下述操作：

（3）按＜Tab＞键，将光标移至 Produce 栏内；

（4）按 X 键或 A 键（也可以使用方向键），在两种 EXE 文件中间进行选择；

（5）按 D 键，确定是否要 Debug Code。

对话框的底部还有三项功能供用户选择使用：

①Make EXE 生成可执行文件，然后返回 Quick BASIC 环境；

②Make EXE and Exit 生成可执行文件，然后返回 DOS 状态；

③Cancel　取消当前命令，返回编辑屏幕。

上述三种功能框中，对话框自动选择 Make EXE（用双线框表示）功能，如果这正是所需要的，只要按回车键，编译便开始。如果想进行新的选择，可继续进行下述操作：

（6）按＜Tab＞键，在三种功能之间进行选择，被选中的以双线框表示。也可以用＜Tab＞键将光标移入上述三个框中的任意一个后，用 M 键选择 Make EXE 功能，用 X 键选择 Make EXE and Exit 功能，而 Cancel 框仍要用＜Tab＞来选择。

（7）按回车键，实现最后的选择：或者进行编译，产生可执行文件，或者取消编译命令，返回编辑屏幕。

EXE 可执行文件是在 DOS 状态下运行的，因此，在运行可执行文件之前，必须用 File 菜单中的 Exit 或 DOS Shell 命令退出 Quick BASIC，返回到 DOS 后，方能运行 EXE 文件。运行 EXE 文件时，只要在 DOS 提示符后打入 EXE 文件名即可（扩展名 EXE 可以省略）。例如，存有可执行文件 EX241. EXE 的磁盘，插入 B 盘驱动器，当前盘为 A 盘，运行此文件时，命令形式为

A＞B：EX241

小　　结

本章在介绍了 Quick BASIC 系统在硬盘和软盘上的安装后，重点讲解了 BASIC 程序的基本操作方法，其中包括输入和编辑源程序，调试和运行程序，将源程序存入磁盘以及将源程序从磁盘读入内存等。这些基本操作，通过上机实习，要逐步达到熟练的程度。

对于一些大型的、复杂的、运行时间较长的程序，为了提高运行速度，可以把它们编译成可在 DOS 状态下独立运行的 EXE 可执行文件。将来我们还会看到，在其他一些场合，EXE 文件也是十分有益的。因此，我们应当学会将一个在 Quick BASIC 环境下

调试好的源程序，编译成可独立执行的 EXE 文件的方法。

习　　题

3-1　在 Quick BASIC 环境下，从键盘输入例 3.3.1 的程序文本，练习程序的输入、编辑、存盘和读盘（从磁盘装入程序）。

3-2　调试和运行例 3.3.1 的程序，直到能较熟练的操作并得到正确的结果，最后打印出程序清单。

3-3　在 Quick BASIC 环境下，对例 3.3.1 的程序进行编译。要求建立两种 EXE 可执行文件，在 DOS 状态下运行所得到的两个可执行文件并得到正确的结果。

第四章　BASIC 语言基础

本章给出构成 BASIC 语言的基础。

通过这一章的学习，我们将知道 BASIC 语言是由哪些字符组成的，它有多少种数据类型，多少种运算，怎样由这些字符组成常量、变量和表达式。最后，对 BASIC 语言的最大单元——模块和过程，给以简单说明。

第一节　语言要素

一、BASIC 字符集

BASIC 字符是 BASIC 语言的最小单元，由这些字符组成标号、关键字、变量和运算符，它们又构成了程序的语句。因此，学习 BASIC 语言，首先要知道 BASIC 语言使用哪些字符。

BASIC 字符集由英文大写字母字符 A—Z 与小写字母字符 a—z，数字字符 0—9 和一些专用特殊字符组成。

特殊字符在 BASIC 语句和表达式中，有特定的意义。这些特殊字符列在表 4.1.1 中，其中，回车符有时还用 ↵ 表示，空格符用 ␣ 表示。

还有许多字符，虽然它们在 BASIC 语言中没有特定的意义，但可以在屏幕上显示或打印出来。

所有上面提到的这些字符，在计算机中都采用 ASCII (American Standard Code for Information Interchange 美国信息交换标准代码) 编码。表 4.1.2 给出部分常用字符的 ASCII 编码，表中 Dec 表示十进制码，Hex 表示十六进制码。

表 4.1.1

字 符	名 称	字 符	名 称
Enter	回车	:	冒号
	空白（空格）	;	分号
!	感叹号	<	小于号
$	美元号	=	等号
%	百分号	>	大于号
&	和号	?	问号
'	单引号（撇号）	@	单价号
(和)	左、右圆括号	[和]	左、右方括号
*	星号	\	后斜线
+	加号	^	上箭头
—	减号	_	下划线
.	句号	#	数目号
/	斜号	,	逗号

表 4.1.2

字符	ASCII		字符	ASCII		字符	ASCII	
	Dec	Hex		Dec	Hex		Dec	Hex
	32	20	@	64	40	`	96	60
!	33	21	A	65	41	a	97	61
"	34	22	B	66	42	b	98	62
#	35	23	C	67	43	c	99	63
$	36	24	D	68	44	d	100	64
%	37	25	E	69	45	e	101	65
&	38	26	F	70	46	f	102	66
'	39	27	G	71	47	g	103	67
(	40	28	H	72	48	h	104	68
)	41	29	I	73	49	i	105	69
*	42	2A	J	74	4A	j	106	6A
+	43	2B	K	75	4B	k	107	6B

字符	ASCII		字符	ASCII		字符	ASCII	
	Dec	Hex		Dec	Hex		Dec	Hex
,	44	2C	L	76	4C	l	108	6C
—	45	2D	M	77	4D	m	109	6D
.	46	2E	N	78	4E	n	110	6E
/	47	2F	O	79	4F	o	111	6F
0	48	30	P	80	50	p	112	70
1	49	31	Q	81	51	g	113	71
2	50	32	R	82	52	r	114	72
3	51	33	S	83	53	s	115	73
4	52	34	T	84	54	t	116	74
5	53	35	U	85	55	u	117	75
6	54	36	V	86	56	v	118	76
7	55	37	W	87	57	w	119	77
8	56	38	X	88	58	x	120	78
9	57	39	Y	89	59	y	121	79
:	58	3A	Z	90	5A	z	122	7A
;	59	3B	[	91	5B	{	123	7B
<	60	3C	\	92	5C	|	124	7C
=	61	3D	]	93	5D	}	125	7D
>	62	3E	^	94	5E	~	126	7E
?	63	3F	—	95	5F			

二、BASIC 程序行

由例 3.3.1 的程序，我们可以看到，BASIC 程序是由一些程序行组成的。BASIC 程序行可以包含一个或若干个语句，也可以没有语句。BASIC 程序行具有以下格式：

〔行标识符〕[语句][：语句]…[' 注释]

由上面的行格式可知，一个程序行的开头部分是行标识符。行标识符相当于给该行起的名字，用于区别于其他程序行。行标识符后面是一系列语句。语句之间用冒号分割开，单引号（撇号）后面的字符集合，被看成是说明部分，对程序的运行没有影响。上述格式中的方括号，说明该项内容是程序行的任选项，是可有也

可以没有的程序行内容。

下面举几个程序行的例子。

例 4.1.1 程序行格式举例。

```
this-line: PRINT "hello"
A$ = "hello": PRINT "hello"
A$ = "hello"    ' this is a BASIC statement
PRINT "hello"
```

以上 BASIC 程序行，具有如下的语法形式：

```
行标识符  语句
语句：语句：语句
语句  '注释
语句
```

在 BASIC 语言程序中，可以使用两种类型的行标识符——行号和标号。

行号可以是 1 到 65529 范围内的任意数。0 行号有特殊的用途，以后专门介绍。行号的大小没有意义，这与传统的 BASIC 语言不同，在传统的 BASIC 语言中，程序是按行号由小到大顺序执行的。

标号是由 1 到 40 个字母数字组成，并且以字母开头，以冒号结尾，但不允许使用 BASIC 关键字。所谓 BASIC 关键字，就是 BASIC 语句格式中含有特定意义的字符组合。例如，在例 3.3.1 程序中，PRINT 便是一个 BASIC 关键字。

在行标号中，不区分字母的大、小写。例如，标号 alpha: 和 ALpha:，认为是同一个标号。

一个程序行，最多只能有一个行标识符。

BASIC 语句分为可执行语句和非执行语句两大类。可执行语句用来说明程序要求执行的具体操作内容，如输入、赋值、输出打印等。非执行语句起说明、定义等作用。

在 Quick BASIC 环境下编辑或输入程序时，每个程序行不能

超过 256 个字符。

以上我们简要地介绍了 BASIC 程序行的格式，不难看出，同一个程序，可以有多种书写形式，即源程序的形式是多种多样的。

例 4.1.1 重写例 3.3.1 的程序。

在下面的程序例子中，我们将使用行号和标号两种行标识符，一个程序行内有一个或两个可执行 BASIC 语句以及非执行 BASIC 语句（注释语句）等。源程序如下：

```
                ' Example 4.1.1
                PI=3.14159: Radius=2.5
computation:    L=2*PI*Radius: S=PI*Radius*Radius
                PRINT "Area=";S        'output
                PRINT "circumference=";L
                END
```

第二节 数据类型

一、基本数据类型

BASIC 程序中的每一个数据，必属于某种规定的数据类型。不同类型的数据，在计算机中的存储方式和所表示的数据范围是不同的，BASIC 语言提供的数据类型，可分为两大类：基本数据类型和用户定义数据类型。

基本数据类型又可分为数字类型和字符串类型。

数字类型包括整型数、长整型数、单精度浮点数和双精度浮点数四种类型。字符串类型包括定长度字符串和变长度字符串两种类型。

下面分别介绍上述各种基本数据类型。

1. 整型数

整型数占内存的两个字节，用二进制补码表示。整型数可用

来表示从－32768 到＋32767 范围内的整数。

2. 长整型数

长整型数占内存的四个字节，用二进制补码表示。长整型数可用来表示从－2147483648 到＋2147483647 范围内的整数。

3. 单精度浮点数（单精度实数）

Quick BASIC 使用 IEEE（Institute of Electrical and Electronics Engineers）格式的浮点数。

数的浮点表示法，类似于通常的科学记数法，或称指数记数法。在科学记数法中，数 0.025 写成 $2.5 \cdot 10^{-2}$的形式，其中关键的两个数字是指数－2 和尾数 2.5。浮点数就是只保存指数、尾数和符号的一种数。

在单精度浮点数中，符号占 1 位（bit），指数占 8 位，尾数占 23 位。所以，一个单精度浮点数占用内存的四个字节。

单精度浮点数准确到 7 位十进制数。数的表示范围是从 $-3.402823 \cdot 10^{38}$到$+3.402823 \cdot 10^{38}$。

4. 双精度浮点数（双精度实数）

双精度浮点数，简称双精度数，是对单精度数的扩充。它占内存的 8 个字节，其中符号占 1 位，指数占 11 位，尾数占 52 位。

双精度数准确到 15 位或 16 位十进制数。它所表示的数的范围是从 $-1.79763134862316 \cdot 10^{308}$到$+1.79763134862316 \cdot 10^{308}$。

图 4.2.1 表示了上述四种不同数据类型的存储格式。

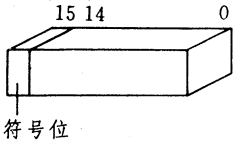
5. 变长度字符串

变长度字符串是一个最多可有 32767 个字符的序列，这些字符的代码范围为 ASCII 码的 0—255。这种类型数据的长度等于它存储的字符数，并且其长度随着存储的字符数的变化而变化。

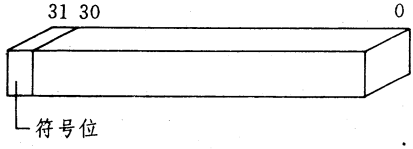
6. 定长度字符串

定长度字符串也是一个最多有 32767 个字符的序列，但这种数据类型的长度被一说明所固定。

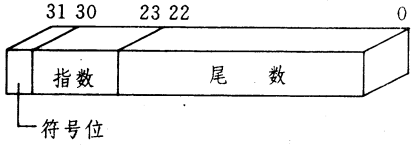
整型数 Integer



长整型数 Long integer



单精度实数 Single - presision real



双精度实数 Double - presision real

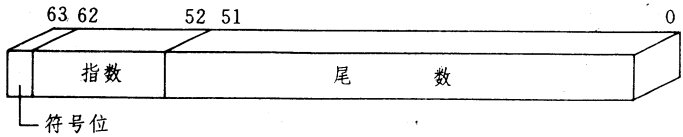


图 4.2.1 BASIC 数的存储格式

说明字符串类型数据的方法将在下一节介绍。

二、用户定义的数据类型

BASIC 允许用户使用 TYPE 语句定义新的数据类型。用户定义的数据类型是由 BASIC 的基本数据类型（变长度字符串类型除外）组合而成的。

TYPE 语句的格式如下：

```
TYPE usertype
    elementname AS typename
    elementname AS typename
    ⋮
END TYPE
```

在上述语句格式中，大写字母部分为 TYPE 语句的关键字。其余部分是由用户确定的标识符。

usertype 是用户给所定义的数据类型起的名字。这个名字最多可以包含 40 个字符，其第一个字符必须是字母，但不能以 FN 开始。BASIC 语言自己保留了一些字，不能给用户使用，因为那些字有一定的含意和用途，这些字称为 BASIC 保留字。表 4.2.1 给出了 BASIC 的所有保留字。

表 4.2.1

ABS	ACCES	ALLAS	AND
ANY	APPEND	AS	ASC
ATN	BASE	BEEP	BINARY
BLOAD	BSAVE	BYVAL	CALL
CALLS	CASE	CDBL	CDECL
CHALN	CHDIR	CHR $	CINT
CIRCLE	CLEAR	CLNG	CLOSE
CLS	COLOR	COM	COMMAND $
COMMON	CONST	COS	CSNG
CSRLIN	CVD	CVDMBF	CVI
CVS	CVSMBF	CVL	DATA
DATE $	DECLARE	DEF	DEFDBL
DEFINT	DEFLNG	DEFSNG	DEFSTR
DIM	DO	DOUBLE	DRAW
ELSE	ELSEIF	END	ENDIF
ENVIRON	ENVIRON $	EOF	EQV
ERASE	ERDEV	ERDEV $	ERL
ERR	ERROR	EXIT	EXP

（续表）

FIELD	FILEATTR	FILES	FIX
FOR	FRE	FREEFILE	FUNCTION
GET	GOSUB	GOTO	HEX $
IF	IMP	INKEY $	INP
INPUT	INPUT $	INSTR	INT
INTEGER	IOCTL	IOCTL $	IS
KEY	KILL	LBOUND	LCASE $
LEFT $	LEN	LET	LINE
LIST	LOC	LOCAL	LOCATE
LOCK	LOF	LOG	LONG
LOOP	LPOS	LPRINT	LSET
LTRIM $	MID $	MKD $	MKDIR
MKDMBF $	MKI $	MKL $	MKS $
MKSMBF $	MOD	NAME	NEXT
NOT	OCT $	OFF	ON
OPEN	OPTION	OR	OUT
OUTPUT	PAINT	PALETTE	PCOPY
PEEK	PEN	PLAY	PMAP
POINT	POKE	POS	PRESET
PRINT	PSET	PUT	RANDOM
RANDOMIZE	READ	REDIM	REM
RESET	RESTORE	RESUME	RETURN
RIGHT $	RMDIR	RND	RSET
RTRIM $	RUN	SADD	SCREEN
SEEK	SEG	SELECT	SETMEM
SGN	SHARED	SHELL	SIGNAL
SIN	SINGLE	SLEEP	SOUND
SPACE $	SPC	SQR	STATIC
STEP	STICK	STOP	STR $
STRIG	STRING	STRING $	SUB
SWAP	SYSTEM	TAB	TAN
THEN	TIME $	TIMER	TO
TROFF	TRON	TYPE	UBOUND
UCASE $	UNLOCK	UNTIL	USING
VAL	VARPTR	VARPTR $	VARSEG VIEW
WAIT	WEND	WHILE	WIDTH
WINDOW	WRITE	XOR	

elementname 是用户定义的数据类型元素的名字。用户可以在 TYPE 语句中定义若干个元素，每个元素最多可包含 40 个字符，它不是保留字，不能以 FN 开始，第一个字符必须是字母。

typename 可以是下述 BASIC 数据类型：

整数	INTEGER
长整数	LONG
单精度实数	SINGLE
双精度实数	DOUBLE
定长度字符串	STRING * n

其中 n 是字符串数据的长度。

例如下面的 TYPE 语句，定义了一个叫作 SymTabEntry 的数据类型。这个新的数据类型，包含了一个定长度字符串（长度为 40 个字符）Identifier，一个长整型数 LineNumber 和一个整型数 Value，共三个元素：

```
TYPE SymTabEntry
    Identifier    AS STRING * 40
    LineNumber    AS LONG
    Value         AS INTEGER
END TYPE
```

用户完成上述定义后，SymTabEntry 就成为一个合法的新数据类型，与基本数据类型起着同等的作用。

这种用户定义的数据类型，也叫作记录，其中的元素叫作记录的域。

关于用户定义的数据类型的应用，将在以后有关章节介绍。

第三节　常数与变量

由上一节我们已经知道，BASIC 语言提供了多种数据类型。无论是何种类型的数据，在程序的应用中，不是表现为常数，就是

表现为变量。那么，在 BASIC 语言中，各种数据类型的常数和变量是怎样表示的呢？这就是本节要讨论的问题。

一、常数

常数是预先定义的，在程序运行期间不变的量。在 BASIC 语言中，有两种形式的常数：数值常数和字符串常数。

表示数值大小的常数，叫作数值常数。例如 1.23 就是一个数值常数。字符串常数表示的则是一个字符序列，例如 abc 可看作是字符串常数。

1. 字符串常数

字符串常数是包含在双引号中的，最多可有 32767 个字符的序列（双引号和回车、换行符除外）。下面是合法的字符串常数：

“HELLO”

“$25，000.000”

“Number of Employers”

2. 数值常数

数值常数包括正负数，共有四种类型：整型、长整型、单精度实数和双精度实数。整型数和长整型数，除了可以用十进制数表示外，还可以用十六进制数和八进制数表示。下面介绍各种类型的数值常数的表示方法。

整型数的十进制表示与通常的书写方法没什么不同。它所表示的数的范围是从－32768 到＋32767，数的后缀％是整型数的记号。如整数 100 可写成 100％，但在这种情况下，后缀％是完全不必要的。

整型数的十六进制表示方法，是在数字前加前缀 &H 或 &h。例如，十六进制数 32F 应写为 &H32F 或 &h32F。十六进制整型数的范围是从 &H0 到 &HFFFF。

整型数的八进制表示方法，是在数字前加前缀 & 或 &O。例如，八进制数 1234 应写为 &1234 或 &O1234。八进制整型数的范

围是从 &O 到 &177777。

长整型数的十进制表示，与通常的书写方法，没有什么不同。它所表示的数的范围是从－2 147 483 648 到＋2 147 483 647，后缀 & 是长整型数的标记，但一般情况下可不用。

长整型的十六进制表示方法，是在数字前加前缀 &H 或 &h，在数字后加后缀 &，例如 &HO&。十六进制长整型数的范围是从 &h0& 到 &hFFFFFFFF&。

单精度实数保留 7 位数字的精度。以下三种书写形式都表示单精度数：

(1) 以字母 E 表示的指数形式的数。例如，单精度数 $-1.09 \cdot 10^{-6}$ 可表示为－1.09E－6。

(2) 以感叹号！为后缀的数。例如，单精度数 25（并非整数），可写为 25!。

(3) 数字不超过 7 位，且带有小数点的定点形式的数。例如，数 3.14159 便自动认为是单精度数，既使没有后缀!，也不会发生问题。

单精度数所表示的范围是从－3.402823E＋38 到＋3.402823E＋38。

双精度实数保留 16 位的精度。双精度实数也有三种表示方法：

(1) 以字母 D 表示的指数形式的数。例如，双精度数 $4.35 \cdot 10^{-10}$ 可写为 4.35D－10。

(2) 以数目号＃为后缀的数。例如 1234.5＃表示的是双精度数。

(3) 数字超过 7 位，没有表示指数的 E 和后缀！的带有小数点的定点数。例如，数 7654321.35 是双精度数，并且可以没有后缀＃。

超出长整型表示范围的整数形式的数为双精度数，并且系统会自动为其填上后缀＃。

BASIC 语言允许用户用符号代表数值常数，称为符号常数。我们可以用符号 Num 代表 1.23，用符号 Ch 代表字符串常数 abc。符号常数，如 Num，Ch，在使用前先要用 CONST 语句定义。

CONST 语句的格式如下：

CONST constantname＝expression [，constantname＝expression] …

其中 constantname 是由用户给定的符号常数名。它最多可包含 40 个字符，第一个字符必须是字母，但不要以 FN 开头，并且不能是 BASIC 保留字。

可以在符号常数名的最后加一类型符，以说明其类型。如前所述，%为整型说明符，& 为长整型说明符，! 为单精度说明符，# 为双精度说明符，$ 为字符串说明符。

expression 是赋给符号常数名的值，在最简单的情况下，它是一个常数值。

下面是一个定义符号常数例子：

CONST MaxChars%＝254，MaxBuf%＝MaxChars%＋1

在这个例子中，我们定义了两个符号常数，它们都是整型常数。经过这样定义后，MaxChars 和 MaxBuf 便可以在程序中作为整型常数使用，并且类型符可以省略。

二、变量

变量就是用符号（也称为标识符）来表示数据，并且这个符号的取值在程序中是可变的，这个符号叫作变量名。

BASIC 的变量名，最多可包含 40 个字符。变量名的第一个字符必须是字母，但不能以 FN 开始，变量名不能是 BASIC 保留字，但允许在变量名中包含保留字。例如，变量名 LOG 是非法的，因为 LOG 是 BASIC 保留字。但是变量名 TimeLog 是合法的。

变量分为简单变量和下标变量两类。下标变量将在“下标变量的应用”一章介绍。本节只讨论简单变量，简单变量有时简称

为变量。

简单变量可以是各种基本数据类型，也可以是用户定义类型。因此，有整型变量、长整型变量、单精度变量、双精度变量、字符串变量和用户定义类型变量，用户定义类型变量也叫作记录变量。

可以用下述三种方式之一来说明简单变量的类型。

(1) 在变量名后面加一个说明变量类型的后缀。

后缀 $ 是用于字符串变量的类型说明符。例如，A$ 是一个字符串变量，它的值是一个字符串。

整型变量的类型说明符后缀是%，长整型变量的类型说明符后缀是&，单精度变量的类型说明符后缀是!，双精度变量的类型说明符后缀是#。例如：

A% 是整型变量；
A& 是长整型变量；
A! 是单精度变量；
A# 是双精度变量。

这些变量名都是A，但是他们代表的不是同一个变量。这相当于把类型说明符后缀看成是变量名的一部分。

记录变量没有类型说明符，因此，它不能用这种方式说明。

(2) 用带有 AS Type 的 BASIC 语句来说明变量类型。

有多个 BASIC 语句可以用来说明变量类型。这里只介绍用 DIM 语句说明变量类型的方法。

DIM 语句的格式如下：

DIM Variable AS Type

这里 Variable 代表的是变量名，Type 是变量类型，它可以是：

INTEGR	整型变量；
LONG	长整型变量；
SINGLE	单精度变量；

DOUBLE	双精度变量；
STRING	变长度字符串变量；
STRING * n	长度为 n 的定长字符串变量；
用户定义的类型名	用户定义类型变量（记录变量）。

设变量 A 是整型变量，B 是单精度变量，我们可以这样说明它们：

```
DIM  A  AS  INTEGER
DIM  B  AS  SINGLE
```

如果令 String1 是变长字符串变量，String2 是长度为 7 个字符的定长度字符串变量，则要这样说明它们：

```
DIM  String1  AS  STRING
DIM  String2  AS  STRING * 7
```

如果有如下的用户定义数据类型 InventoryItem：

```
TYPE  InventoryItem
      Description      AS    STRING * 25
      Number           AS    STRING * 10
      Quantity         AS    LONG
      OderPoint        AS    LONG
END TYPE
```

则要定义 CurrenItem 和 PreviousItem 为 InventoryItem 类型变量时，DIM 语句形式为：

```
DIM CurrentItem AS InventoryItem, PreviousItem AS InventorItem
```

(3) 用 DEF Type 语句说明变量类型

DEF Type 语句的格式是：

```
DEFINT    字母范围 [, 字母范围] …
DEESNG    字母范围 [, 字母范围] …
DEFDBL    字母范围 [, 字母范围] …
```

DEFLNG　字母范围［，字母范围］…

DEFSTR　字母范围［，字母范围］…

其字母范围是所定义的变量的头一个字母在英文字母表中的范围。它的书写格式是：

字母 1［—字母 2］

DEFINT 说明的是整型变量，DEFSNG 定义的是单精度变量，DEFDBL 定义的是双精度变量，DEFLNG 定义的是长整型变量，DEFSTR 定义的是字符串型变量。

下面是应用 DEF Type 语句定义变量类型的几个例句：

DEFINT　A—C

DEFSNG　D

DEFDBL　E—G，X—Z

第一个例句说明，以字母 A，B 或 C 开头的变量，是整型变量。例如，BCY 是整型变量。第二个例句说明，以字母 D 开头的变量是单精度变量。例如，DAB，DCY 都是单精度变量。第三个例句说明，以字母 E 到 G 和以 X 到 Z 开头的变量，都是双精度变量。例如，EXAM，YOK 都是双精度变量。

但要注意，类型说明符（后缀％，&，！，#，$）总是比 DEF Type 优先起作用。例如，在程序中虽已使用了上述三个变量类型定义语句，但变量 DMNK $ 是字符串型变量，而不是单精度变量。

再有，DEF Type 语句不影响记录元素。

符号常数也不受 DEF Type 语句的影响。

当一个变量没有用任何一种方式说明其类型时，则把它作为单精度变量处理。

当数值超出相应类型数据所能表示的范围的最大绝对值时，称为溢出（Overflow），计算机给出相应的错误信息；小于所能表示的最小绝对值时，则按 0 处理。

第四节 BASIC 标准函数

为了方便用户，BASIC 语言提供了大量的函数，称为标准函数。这些标准函数已被定义，其名称作为 BASIC 语言的标准标识符，用户可以直接使用。

BASIC 语言定义的标准函数很多，本书不能全部介绍。本节主要介绍算术函数。有些函数将在有关章节再作介绍。

标准函数由函数名和自变量两部分组成。函数名由三个或三个以上字母组成，有些函数名以 $ 结尾，自变量跟在函数名后面，并括在圆括号内。例如，开平方函数为 SQR（X）。它是计算 X 的平方根。

在程序中写出函数名并给定自变量 X 的值，BASIC 就会计算出相应的函数值。例如，下面的 BASIC 语句会在屏幕上显示出 $\sqrt{4}$ 和 $\sqrt{x+y*z}$ 的值：

```
A=SQR (x+y*z)
PRINT SQR (4), A
```

这里 SQR 便是求平方根函数。

自变量 X，除了可以是数值常数（包括符号常数）也可以是数值型变量，甚至是一个比较复杂的且按 BASIC 格式表示的算术式。我们可以把自变量 X 的所有上述可能情况，概括地称为数值表达式。关于 BASIC 表达式的更多的细节问题，将在下一节介绍。

下面介绍几个常用的算术运算方面的 BASIC 标准函数。

1. SIN（X）

这个函数计算以弧度为单位的角 X 的正弦值，即 sinx。X 为双精度时，按双精度计算函数值，否则，按单精度计算。

2. COS（x）

这个函数计算以弧度为单位的角 X 的余弦值，即 cosx。X 为双精度时，按双精度计算函数值，否则，按单精度计算。

3. TAN (X)

这个函数计算以弧度为单位的角 X 的正切值，即 tgx。X 为双精度时，按双精度计算正切值，否则，按单精度计算。

4. ATN (X)

这个函数计算 X 的反正切值，即 actgx。结果是以弧度为单位给出的，其范围在 $-\pi/2$ 到 $\pi/2$ 之间。X 是双精度数，ATN 按双精度计算反正切值，否则，给出单精度计算结果。

5. EXP (X)

这个函数计算指数函数 e^x 的值。指数 X 的大小必须使函数 e^x 不产生溢出错误。X 为双精度数时，EXP 按双精度计算，否则，按单精度计算。

6. LOG (X)

这个函数计算自然对数 lnx 的值。X 的值必须大于零，否则，会产生非法函数调用（Illegal function call）的错误。

7. ABS (X)

这个函数计算 X 的绝对值，即 $|x|$。

8. SQR (X)

这个函数计算 X 的平方根。其中 X 必须大于等于零。

9. SGN (X)

这个函数能够求出 X 的符号的信息。

X＞0 时，SGN 给出 1；

X＝0 时，SGN 给出 0；

X＜0 时，SGN 给出－1。

10. FIX (X)

函数 FIX (X) 的功能是取 X 的整数部分。

11. INT (X)

函数 INT (X) 的功能是取一个不大于 X 的最大整数。

12. CINT (X)

函数 CINT (X) 的功能是，将 X 的小数部分四舍五入后，取

X 的整数部分。X 的值必须在－32768 到 32767 范围内，否则，会产生溢出错误。

从表 4.4.1 的数值例子，可以看出 FIX，INT 和 CINT 三个函数在取整数部分时，在功能上的重要差别。

表 4.4.1

X	INT(X)	CINT(X)	FIX(X)
99.3	99	99	99
99.5	99	100	99
99.7	99	100	99
－99.3	－100	－99	－99
－99.5	－100	－100	－99
－99.7	－100	－100	－99

第五节　BASIC 表达式

一、表达式与运算

任何数学计算问题，都是用数学式子表达，按数学式子进行运算的。使用 BASIC 语言进行运算也是如此。用 BASIC 语言表达的运算式子，称为 BASIC 表达式。具体地说，BASIC 表达式是用 BASIC 运算符将 BASIC 语言的常数、变量以及标准函数连接起来的计算式。最简单的 BASIC 表达式，可能只含有一个常数，或一个变量，或一个标准函数。

BASIC 表达式可分对字符串量进行运算的字符串表达式和对数值量进行运算的数值表达式。字符串表达式及运算将在“字符串处理”一章讲述，本节只介绍 BASIC 数值表达式。

数值表达式中可包含三类不同性质的运算。按其运算的优先次序，从高到低，这三种运算依次为：算术运算，关系运算，逻辑运算。

下面分别介绍这三种运算。

二、算术运算与算术表达式

算术运算是由 BASIC 算术运算符对常数、变量和标准函数进行的操作。算术运算的计算式叫作算术表达式。在算术表达式中可以用圆括号来改变表达式中的运算次序，这和通常的算术运算规则是一样的，不过在 BASIC 表达式中只使用一种圆括号，而算术表达式中可以在圆括号外再套圆括号，形成多层圆括号的表达式。

BASIC 语言的算术运算符，按其运算优先次序，从高到低，共有 7 种，如表 4.5.1 所示。

表 4.5.1

运算符	操　　作	举　　例
^	乘幂	X^Y
*，/	乘和除。同优先级，由左向右顺序运算	X * Y/Z
\	整除。参加运算的数进位成整数后再除，其商截尾为整型或长整型数	A\B
MOD	模运算。结果为整除的余数	A MOD B
+，−	加和减。同优先级，由左向右依次运算	A−B+C

下面举例说明整除（\）和模运算（MOD）。

整除运算

在进行整除时，首先经四舍五入后，除数与被除数被化成整型或长整型数，然后以整数商作为计算结果。

例如：10\4 的结果为 2；

−32 768.499\10 的结果为−3 276；

25.68\6.99 的结果为 3。

模运算

A MOD B 的运算，实际上是求 A\B 的余数。

例如：7 MOD 4 的结果为 3；

25.68 MOD 6.99 的结果为 5。

我们说过，BASIC 表达式中，可以有标准函数。它是作为一个复杂运算的结果出现的，因此，它有比上述七种运算都高的优先级，仅次于括号。

算术运算的优先顺序，可用图 4.5.1 表示。

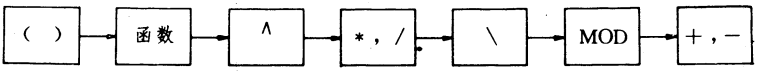


图 4.5.1　算术运算的优先级

例 4.5.1　写出数学算式

$$a+b\times c\times d/2+e^2$$

的 BASIC 表达式并说明其运算顺序。已知 a=2，b=3，c=4，d=5 和 e=6，试求表达式的值。

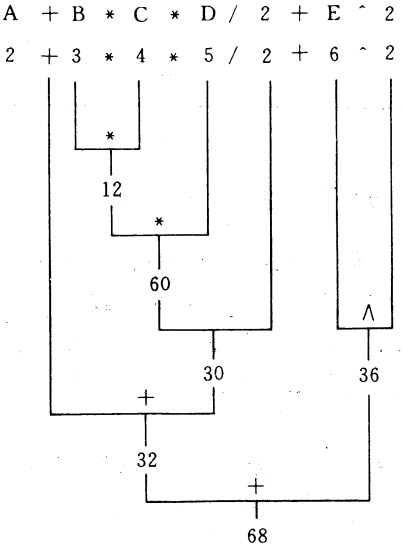


图 4.5.2　例 4.5.1 的解

其 BASIC 表达式为

$$A+B\times C\times D/2+E\wedge 2$$

运算顺序及结果示于图 4.5.2。

例 4.5.2 数学算式与 BASIC 表达式的转换及常见错误，举例示于表 4.5.2。

表 4.5.2

数学算式	正确的 BASIC 表达式	错误的 BASIC 表达式
$\frac{1}{2}$	1/2	$\frac{1}{2}$
$(-5)^{\frac{2}{3}}$	(-5)∧(2/3)	(-5)∧2/3
(a+b)(a−b)	(A+B)*(A−B)	(A+B)(A−B)
$\frac{5}{3x}$	5/(3*x)或 5/3/x	5/3*x
$\frac{a\cdot b}{c\cdot d}$	a*b/(c*d)或 a*b/c/d	a*b/c*d
$\frac{A+B}{C+D}$	(A+B)/(C+D)	A+B/C+D
$\frac{2\cos\pi x}{\ln x}$	2*cos(3.14159*x)/LOG(x)	2cos(π*x)/lnx

三、关系运算与关系表达式

关系运算用于比较两个量。比较的结果只有两个可能的值：或者是“真”(True)，其值为−1，或者是“假”(False)，其值为 0。这两个数称为逻辑值。比较运算的结果，主要用于判断（真还是假），以决定程序的流程。BASIC 在判断一个量时，把非零的值都认为是“真”。

关系运算中共有六个运算符。这些运算符号，它们的功能以及表达式举例，列于表 4.5.3 中。

表 4.5.3

运 算 符	检 查 的 关 系	表 达 式
=	相等	X=Y
<>	不等	X<>Y
<	小于	X<Y
>	大于	X>Y
<=	小于等于	X<=Y
>=	大于等于	X>=Y

所有关系运算符的优先级都是相同的。当表达式中既有算术运算又有关系运算时，先作算术运算，后作比较。所以关系表达式有如下的一般形式：

<算术表达式 1><关系算符><算术表达式 2>

例如，关系表达式

X+Y< (T−1) /Z

例 4.5.3 计算关系表达式 X+Y< (T−1) /Z，已知 X=5，Y=7，T=10，Z=3。

将已知数据代入关系表达式，得

5+7< (10−1) /3

12<3

关系式 12<3 不成立，所以比较结果为“假”(0)。

四、逻辑运算与逻辑表达式

BASIC 语言中有 6 种逻辑运算。这 6 种逻辑运算的符号和名称，按其优先级由高到低的次序，列入表 4.5.4 中。

表 4.5.4

运 算 符	名 称
NOT	逻辑非(logical complement)
AND	与(conjunction)
OR	或(disjunction)
XOR	异或(exclusive or)
EQV	相等(equivalence)
IMP	蕴涵(implication)

用 T 表示真值，F 表示假值，这 6 种逻辑运算的定义，如表 4.5.5 所示。

表 4.5.5

操作数	逻 辑 运 算 的 结 果					
X Y	NOT X	X AND Y	X OR Y	X XOR Y	X EQV Y	X IMP Y
T T	F	T	T	F	T	T
T F	F	F	T	T	F	F
F T	T	F	T	T	F	T
F F	T	F	F	F	T	T

在表达式中，逻辑运算在算术运算和关系运算之后进行。

如果参与逻辑运算的操作数的值只有 0 和－1，逻辑运算的结果也只有 0 和－1。例如，有 X＝－1，Y＝0，则 X AND Y 运算的结果为 0。

一般情况下，两个数值进行逻辑运算时，如果它们不是整型数，则自动经四舍五入后转换为整型数或长整型数。并且必须不超出长整型数所能表示的范围，然后进行逻辑运算。逻辑运算是按位（bit，二进制数的位）进行的。其中每位的“1”看成是 T（真），“0”看成是 F（假）。

下面举例说明上述各种逻辑运算。

例 4.5.4 进行下列逻辑运算

(1) 63 AND 16

(2) 4 OR 2

(3) NOT 2

63 AND 16 的结果是 16，因为

$$\begin{aligned} 63 &= 00000000\ 00111111 \\ \text{AND } 16 &= 00000000\ 00010000 \\ \hline 16 &= 00000000\ 00010000 \end{aligned}$$

用同样方法，按 OR 运算的定义，4 OR 2 的结果是 6。

根据 NOT 运算的定义，NOT 2 的二进制表示为

$$\begin{aligned} 2 &= 00000000\ 00000010 \\ \text{NOT } 2 &= 11111111\ 11111101 \end{aligned}$$

根据补码的定义，它表示的是－3。所以，NOT 2 的结果是－3。

例 4.5.5 说明下列逻辑表达式的运算过程和结果：

A AND B OR C AND NOT D

已知 A＝5，B＝7，C＝12，D＝3。

用图 4.5.3 说明这个表达式运算过程和结果。

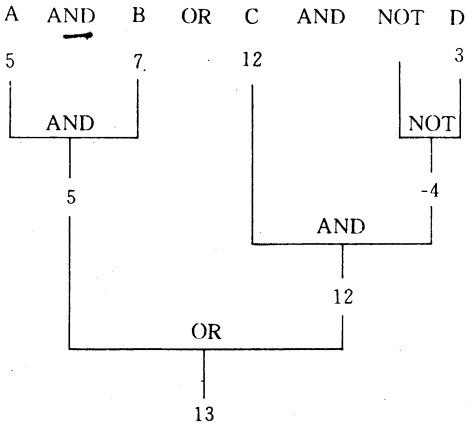


图 4.5.3　例 4.5.5 的解

第六节　程序模块和过程

由本章第一节知道，BASIC 程序是由一些程序行组成的。每个程序行的格式如前所述，它可能包含标号、语句和注释等。

一个程序要完成一定功能、任务。这些功能和任务，可能很简单，也可能很复杂。因此，如何把程序设计得结构清楚、合理和高效率，是一个很重要的问题。特别是对于复杂的、大型的程序，这个问题就更为突出。

本节简要的介绍现代程序设计技术中广泛采用的模块结构的概念。

一、程序模块

将一个复杂的程序分解为若干个较小的，简单的任务的序列是简化程序设计，缩短程序开发周期和方便调试的有效方法。实现这种方法的技术，就是程序的模块化设计。BASIC 语言为程序的模块化，提供了一定的手段，以方便用户。

什么是模块呢？模块是一个具有独立功能的、能够单独进行编译的源文件。这就是说，模块是程序所包含的独立的组成部分。一个程序至少含有一个模块。每个模块作为一个独立文件存入磁盘。一个程序中，有一个主模块。主模块就是包含有程序开始运行的入口点的模块。

程序模块化后，相互独立的模块可以为许多不同的程序共享，即模块可以当作积木块，在不同的程序中使用。

二、过程

过程也是将一个程序分成较小的逻辑单元的编程技术。过程是这样一种功能块，它可以被程序的其他部分多次调用，从而避免在程序中多次重复完成该功能的程序段代码。过程的调用如图

4.6.1 所示。

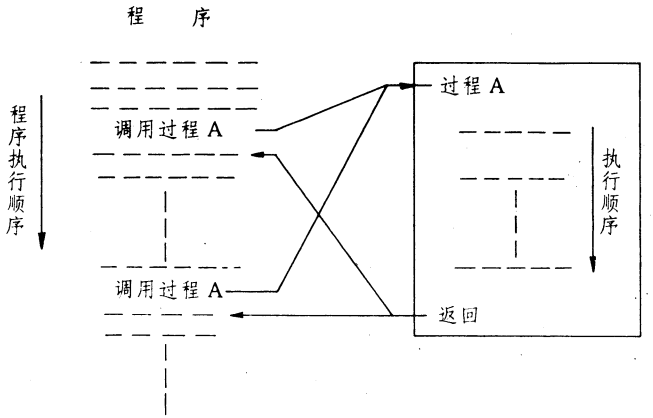


图 4.6.1　过程调用示意图

由图 4.6.1 可以看出，在程序的运行中，当需要完成过程 A 的功能时，便调用过程 A。于是开始执行过程 A 的程序。过程 A 执行完毕，又返回到调用它的程序块，继续运行。

由于把程序分解为小逻辑单元，必然使程序的编制变得更容易、更方便。完成一定功能的过程，经过很小的修改，甚至不修改，就可以移植到其他程序中去。

BASIC 语言提供了两种结构的过程：子程序和函数。在“过程与程序模块”一章，将详细介绍过程的设计和应用。

三、用户程序库

过程也可作为一个独立的模块来处理。这时，它与任何其他模块没有什么区别，也是一个程序中的模块之一，如以上所述。

但是，如果把一些过程模块放在一起，组成一个独立的文件，并且能为其他程序所调用，那么，这个文件便叫作库，或称用户程序库。

显然，库技术的应用，为程序的开发带来了许多方便。用户使用库就像使用一个 BASIC 语句一样简单。

小　　结

本章讲述了构成 BASIC 语言的基本元素，从最小的元素字符讲到程序模块。通过本章的学习，要着重掌握以下几个问题：

1. BASIC 语言的基本符号

BASIC 语言的基本符号包括三个方面：字母、数字和一些特殊字符。

2. BASIC 的数据类型

BASIC 有两大数据类型：基本数据类型和用户定义数据类型。基本数据类型中又分为数值类型和字符串类型两大类。

字符串类型又分为定长字符串类型和变长字符串类型两种。

数值数据类型又分为整数，长整数，单精度浮点数和双精度浮点数四个数据类型。

3. BASIC 常数

BASIC 常数是程序中固定不变的量。BASIC 中有两类常数，数值常数和字符串常数。数值常数有整型、长整型、单精度和双精度四种数据类型。我们要掌握好它们的表示方法。

BASIC 常数可以用符号来代表，这能给程序设计带来一定的好处。符号常数在使用之前，要用 CONST 语句来定义。

4. BASIC 变量

学习好变量，是学好 BASIC 程序设计的第一步，也是关键的一步，一定要掌握好各种数据类型变量的表示方法和定义方法。

本章介绍了下列各种类型变量：整型变量；长整型变量；单精度变量；双精度变量；定长度字符串变量；变长度字符串变量；用户定义类型变量（记录变量）。

本章还介绍了三种定义变量的方法：用变量类型符(后缀)%，

&,!，#和$；用DIM…AS…语句；用DEF Type语句。

5. BASIC标准函数

BASIC标准函数也叫BASIC内部函数。大量的BASIC内部函数，给用户提供了很大的方便。这一章我们只介绍了其中最常用的十几个函数，还有一些常用的函数，在以后的章节中陆续介绍。本章介绍的这些函数，应该很好掌握。

本章介绍的函数有SIN(X)，COS(X)，TAN(X)，ATN(X)，EXP(X)，LOG(X)，ABS(X)，SQR(X)，SGN(X)。FIX(X)，INT(X)，CINT(X)共12个函数。

6. BASIC的运算与表达式

BASIC可以进行四类运算：算术运算，关系运算，逻辑运算和字符串运算。本章只介绍了前三种运算。

算术运算包括7种运算。依照优先级由高到低的次序，这些运算是：乘幂(^)，乘与除(*，/)，整除(\)，模运算(MOD)，加与减(+，-)。

关系运算有6种。它们是相等(=)，不等(<>)，小于(<)，大于(>)，小于等于(<=)和大于等于(>=)。这些运算的优先级相同。

逻辑运算有6种。它们是逻辑非（NOT），与（AND），或（OR），异或（XOR），相等（EQV）和蕴涵（IMP）。这6种运算是按优先级由高到低列出的。

上述三类运算，以算术运算的优先级最高，其次是关系运算，逻辑运算的优先级最低。

用运算符和圆括号连接常数、变量或内部函数，便组成了BASIC表达式。BASIC表达式是用来表示所进行的运算操作的，是程序的基本组成部分，必须熟练掌握。

7. 模块与过程

在本章的最后，简要地给出了模块与过程的概念，在第十章我们还要深入学习这方面的内容。

习　　题

4-1　各种不同类型的常数和变量是怎样区别的？

4-2　将下列数学式写成 BASIC 表达式：

(a) $3+\dfrac{4}{7+9}$　　(b) $\dfrac{3^{12}}{5+6^2}$

(c) $\dfrac{1}{7^2}(\dfrac{5}{11})^3$　　(d) $\pi\ln(a+\sqrt{b^2+c})$

(e) $\sin^2x+e^x$　　(f) $\cos^2(x+y)+\sin(x+y)^2$

(g) $\dfrac{1-x}{1-\dfrac{y}{1-z}}$　　(h) $a\sqrt{1+\sin a}$

4-3　改正下列 BASIC 表达式中的错误：

序　号	数　学　式	表　达　式
(a)	$\dfrac{3+5}{4+7}$	3+5/4+7
(b)	$\dfrac{3}{4\times7}$	3/4*7
(c)	$\left(\dfrac{x+y}{4z}\right)^2$	(x+y/4*z)^2
(d)	$(A\cdot B)^3$	A*B^3
(e)	$\left(\dfrac{A}{B}\right)^{N-1}$	A/B^N-1

4-4　写出下列 BASIC 表达式的数学式：

(a) 5^2/4+6　　(b) 4^4/(7+6)

(c) 8+3*2*4/5−1　　(d) 3.14*R*R

(e) A+B/C/D　　(f) 1/N^(1/N)

(g) EXP(A+B)*(A+B)

(h) N*(N−1)*(N−2)/(2*3)

4-5　给出下列函数的值，已知 A=4.5，B=3.4，C=−2.7：

INT(A)　　CINT(A)　　FIX(A)

INT(B)　　CINT(B)　　FIX(B)

INT(C)　　CINT(C)　　FIX(C)

4-6 已知 X＝－4.7，Y＝6.2，计算下面表达式的值：

Y＊SGN(X－Y)＋ABS(INT(X))

4-7 写出下列各式的运算结果：

(a) 12 AND 7　　(b) 7 OR 12

(c) NOT(7 OR 12)　　(d) (NOT2)＋1

4-8 已知 A＝12，B＝7，C＝6，计算下式的值：

A＞C AND B＜C

第五章　简单程序设计

简单程序是这样一种结构的程序，它的执行自始至终按着语句的顺序，由前向后进行，并且每个语句必执行一遍，且只能执行一遍。

本章从由一、二个语句组成的最简单程序入手，学习简单程序设计方法，熟悉几个常用的BASIC语句和函数，解决如何给变量赋值和如何将数据（计算结果）输出的问题。

第一节　变量的赋值——赋值语句

一、最简单的BASIC程序

首先看两个例子。

例5.1.1　计算一个学生三门课程的平均分。这三门课程的考分是：数学97分，物理83分，化学81分。

程序如下：

```
PRINT (97+83+81) /3
END
```

例5.1.2　计算x=2时的双曲线函数shx。

程序如下：

```
PRINT (EXP (2) -EXP (-2)) /2
END
```

上面的两个程序，仅由两个语句组成。

第一个语句是在显示器上输出（显示）数据PRINT。当执行

这个语句时，首先计算算术表达式（97+83+81）/3 的值，然后把结果显示在屏幕上。

第二个语句是结束语句 END。当程序执行这个语句时，停止程序的运行，返回 BASIC 环境。

当程序执行到最后一行，没有 END 语句，程序也会正常结束。因此，在上面的两个例子中，END 语句还可以省略。

为了使程序容易阅读和便于交流，常在程序里写一些说明文字。说明文字必须写在注释语句 REM 里。REM 语句的简写形式是“'”（撇）。下面的例子说明 REM 语句的用法。

例 5.1.3　给例 5.1.1 的程序加上说明文字。

把说明文字写在程序的第一行：

```
REM Compute Average
PRINT (97+83+81) /3
END
```

把说明文字写在被说明语句的后面：

```
PRINT (97+83+81) /3 : ' Compute Average
END
```

注释语句对于程序的运行没有影响，这样的语句叫非执行语句。而象 PRINT，END 这样的语句，则叫作可执行语句。

二、赋值语句 LET

不使用变量的程序，如例 5.1.1 和例 5.1.2 的程序，不能解决复杂的问题，至多象个一般的计算器。计算机程序的特点之一，就是用变量来解决计算问题。变量不仅存储已知数据，还用来存储计算的中间结果和最终结果。

程序中使用变量时，必须给变量赋值。给变量赋值是用赋值语句实现的，赋值语句的格式是：

[LET] 变量＝表达式

它的功能是把表达式的值赋给变量。语句的关键字 LET 用方括号

括起来，表明关键字 LET 是任选项，即在赋值语句中可以省略。语句中的等号“=”叫作赋值号。

例 5.1.4 一个物体以 $a=10m/s^2$ 的加速度从静止状态开始作匀加速直线运动。试计算 5 秒钟后此物体移动的距离，编写 BASIC 程序。

计算公式：　　$S=at^2/2$

式中 a 为加速度，t 为时间，S 为路程。

按计算公式编写 BASIC 源程序如下：

```
a=10：t=5（或写 LET a=10：LET t=5）
S=a*t*t/2（或写 LET S=a*t*t/2）
PRINT S
END
```

程序的第一行写了两个语句。这时，在两个语句之间必须使用冒号“:”在这个程序中，使用了三个赋值语句。

对于赋值语句的应用，必须注意以下几点：

(1) 赋值语句的赋值号与数学中的等号，虽然在形式上是相同的，但其含义是有差别的。

例如

$$X=X+1$$

作为数学式，它是没有意义的，因为 X 不能等于 X+1。但作为赋值语句，它是有意义的。这个语句表示，将变量 X 的当前值加上 1，然后将其结果再赋给变量 X。设变量 X 的当前值等于 8，执行赋值语句 X=X+1 后，变量 X 的当前值由 8 变为 9。

又如

$$Y=-Y$$

作为数学式，只有 Y=0 时才有意义。而作为赋值语句，它表示的是改变变量 Y 当前值的符号。如果 Y 的当前值是−5，那么执行了赋值语句 Y=−Y 后，变量 Y 的当前值变为+5。

(2) 赋值号的左侧必须是变量，不能是表达式。例如，下面

的语句是错误的：

$$2 * A = X + Y$$

$$35 = 2 * X$$

（3）程序中的变量在赋值以前，对于数值型变量来说，它的值为零；对于字符串变量来说，它的值是长度为零的字符串。

三、数据类型转换

由前面的程序举例可知，赋值语句中等号的右侧，在一般情况下，是进行若干种运算的表达式。这表明赋值语句的功能，不仅仅是赋值，而且还有计算功能。同样，PRINT 语句的输出项，也可能是进行若干种运算的表达式，因此，输出语句 PRINT 也具有运算功能。

在进行表达式的运算过程中，不同类型数据之间，会发生类型转换问题。

BASIC 按以下规则把数据从一种类型转换到另一种类型。

（1）将一种类型的数值赋给另一种类型的数值变量时，数据将按变量的类型存储。

例如

```
A%=23.42
PRINT A%
```

程序的输出是 23。小数部分按四舍五入处理。

又如，表达式 7/9 是按单精度计算的。将其单精度结果，赋给双精度变量 X# 时，虽然按双精度存储，但其实际精度并不会超过单精度。例如，程序

```
X#=7/9
PRINT X#
```

的输出是 0.7777777910232544。小数点七位以后的数字（910232544）是没有意义的。而表达式 7#/9# 是按双精度运算，其结果也是双精度的。因此，程序

```
X#=7#/9#
PRINT X#
```

的输出是 0.7777777777777778。

但是，如果将一个数值量赋给字符串变量，或将一个字符串量赋给数值变量，都会产生类型不匹配（Type Mismatch）的错误信息。

（2）在计算表达式的值时，算术运算或关系运算中的操作数，将转换成与其中精度最高的操作数相同的精度，运算的结果为最后的精度。

例如，表达式 X% /Y! *Z#。X%/Y! 的运算是单精度的（运算操作数的最高精度），其结果也是单精度值。这个结果与双精度的 Z# 相乘时，转换为双精度运算，其最后结果也是双精度数。正如在前面的例子所看到的那样，在将单精度的操作数转换为双精度时，并不能得到真正的双精度，本例的计算结果虽然表示为双精度，但它只有单精度的准确度。下面的程序将说明这个问题：

```
X%=2  Y!=1.5  Z#=100
PRINT X%/Y!*Z#
```

它的输出是133.3333373069763。

要想得到真正的双精度的准确值，上例可作如下修改：

```
X%=2  Y#=1.5  Z#=100
PRINT X%/Y#*Z#
```

这时程序输出的是133.3333333333333。

第二节　输出语句

绝大多数程序，需将它的运算结果输出到显示器屏幕或打印机，以便能直接看到程序的运行结果，这就需要输出语句。本节详细介绍输出语句 PTINT，LPRINT，PRINT USING 和 LPRINT US-

ING 的格式和功能，另外，还要介绍与输出有关的两个函数 TAB 和 SPC。

一、PRINT 和 LPRINT 语句

语句 PRINT 用于在屏幕上显示数据，语句 LPRINT 用于在打印机（设备符 LPTI）上打印数据。两个语句的书写格式完全一样。因此，在下面的叙述中，可以不必区分是 PRINT 还是 LPRINT。

这两个语句的格式是：

PRINT［表达式组］［｛；｜,｝］

LPRINT［表达式组］［｛；｜,｝］

语句中第一个任选项表达式组是输出的内容，它可以是一个或若干个表达式，如果没有表达式，则输出一个空白行。表达式可以是数值的或字符串的表达式。

PRINT（LPRINT）语句输出的单精度数和双精度数有两种格式：定点格式和浮点格式。

如果能用7位或更少位的数字来表示一个单精度数，而不失其精度，则用定点格式，否则，用浮点格式。例如，单精度数$1.1\cdot10^{-6}$用定点格式输出，即为0.0000011，而$1.1\cdot10^{-7}$则要用浮点格式输出，即为1.1E－7。

同样，对于双精度数，如果能用16位或更少位的数字表示，且不失其精度，就用定点格式表示，否则，用浮点格式表示。如，双精度数$1.1\cdot10^{-15}$的输出为定点数0.0000000000000011，而双精度数$1.1\cdot10^{-16}$的输出，则为浮点数1.1D－16。

语句格式中的第二项也是任选项。大括号和竖线表示在两个或多个中必须选择一个，在这里就是选用分号或者逗号。它们的作用是分隔每一个输出项并说明其输出位置。

BASIC 把一行分成长度为14个字符的若干输出（显示或打印）区域。分隔符逗号能使其后的表达式的值，输出在下一个输出区域开始的地方，每个区域上至多输出一个数据。

PRINT（LPRINT）语句以逗号结束时，下一条 PRINT（LPRINT）语句，将在上条语句的输出行上继续输出。如果语句的结尾没有逗号，则在输出完该语句的数据后，下一条 PRINT（LPRINT）语句将重新开始一行的输出。

当 PRINT（LPRINT）语句的输出项数，大于一行的输出区域个数时，它能自动地接着在下一行输出。

例 5.2.1 分析下面程序中 LPRINT 语句的打印机输出形式：

```
Chr1$ = "One": Chr2$ = "Two": Chr3$ = "Three"
Chr4$ = "Four": Chr5$ = "Five"
A! = 1.1E-6:    B! = 1.1E-7
A# = 1.1D-15: B# = 1.1D-16

LPRINT Chr1$, Chr2$, Chr3$, Chr4$,
LPRINT Chr5$
LPRINT A!, B!, A#, B#
LPRINT "End"
END
```

这个程序的打印输出如图5.2.1所示。由图可以看出，输出打印的格式，完全符合上述关于逗号分隔符输出格式的原则。

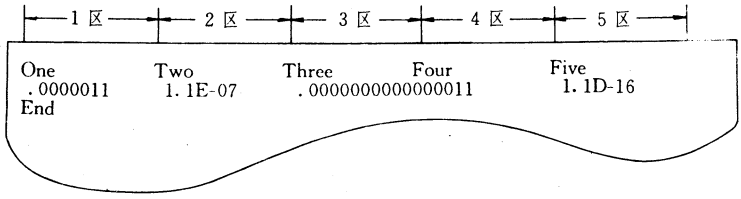


图5.2.1 例5.2.1程序的打印输出格式

由于在 PRINT（LPRINT）中使用了逗号，使得每个数据，必须都在一个输出区域的头上开始打印或显示，因此，输出的数据

呈表格样式。上述输出形式，也叫作标准格式。

下面介绍在PRINT(LPRINT)语句中采用分号时的输出格式。

分号能使下一个要显示或打印的值，紧跟着前一个显示或打印出来。正号不显示（打印）但保留其位置（空格），数据之后留一空格，但字符串量除外。表达式之间的空格也起分号的作用。

如果PRINT（LPRINT）语句以分号结束，则下一条PRINT（LPRINT）语句，继续在同一行上前一个数据之后输出。如果语句PRINT（LPRINT）不用标点符号结束，则输出完成后，产生一回车换行。

如果PRINT（LPRINT）语句要求输出的内容比屏幕或打印宽度还要宽，则自动在下一行继续输出。

例 5.2.2 观察下面程序的输出格式：

```
Chr1$ = "One": Chr2$ = "Two": Chr3$ = "Three"
Chr4$ = "Four": Chr5$ = "Five"
A! = 1.1E-6: B! = 1.1E-7
A# = 1.1D-15: B# = 1.1D-16
PRINT Chr1$; Chr2$; Chr3$; Chr4$;
PRINT Chr5$
PRINT A!; B!; A#; B#
PRINT "End"
END
```

这个程序的显示如图5.2.2所示。

```
One Two Three Four Five
 .0000011      1.1E-07 .0000000000000011    1.1D-16
End
```

图5.2.2 例5.2.2程序的输出显示格式

图5.2.2所示的输出格式，也叫紧凑格式。

在一个 PRINT（LPRINT）语句中，同时使用分号和逗号，可以实现两种格式的输出。这种结合，有时是很有用的。例如，可使数据的输出形式清楚和明了。

例 5.2.3 改写例5.1.4程序中的输出显示语句 PRINT。

修改后的程序如下：

```
a=10: t=5
S=a*t*t/2
PRINT "a=";a,"t=";t,"S=";S
END
```

这时，程序有如下的显示形式：

```
a=10      t=5       S=125
```

显然，这种输出形式，要比只简单的输出三个数值醒目和容易阅读。

二、TAB 函数与 SPC 函数

TAB 和 SPC 函数只能在 PRINT 和 LPRINT 语句中使用，它们的功能是指定数据显示或打印的位置。

显示器和打印机的行宽是一定的。例如，显示器通常是每行可显示80个字符，或说显示器的宽度是80列，每行打80个字符的打印机，行宽也是80列。当然它们的行宽（列数），可以用相应的命令和语句加以改变，但在以下的叙述中，总是假设显示器和打印机列数的缺省值即为80。

1. TAB 函数

TAB 函数的格式是：

TAB（column）

参数 column 是数据要显示（或打印）列数的数值表达式，列数1是最左边的输出位置，当 column＜1时，输出位置为第1列。

例如，语句

PRINT TAB（5）；A；TAB（25）；B

表示变量A的值从第5列开始显示，变量B的值从第25列开始显示，列数不能超过32767。

这种由TAB函数控制的输出格式，也叫作特定格式，特定格式主要用于组织表格和绘制简单图形。

如果当前显示（打印）位置超出函数要求的列数，那么，TAB函数把输出位置移到下一行的相应列。

如果TAB函数给定的列数大于显示器或打印机当前工作的行宽d（设d=80），则实际的输出位置（列数），应按下式计算：

Column MOD d

例如，TAB（1287）的实际输出位置是

1287 MOD 80=7（列）

下面的例子展示出TAB函数在不同参数值情况下的效果。

例5.2.4 观察TAB函数的效果，屏幕行宽为80列。

```
PRINT "12345678901234567890123456789O" TAB (20) "one"
PRINT TAB (1287) "two"
PRINT TAB (255) "three"
PRINT TAB (-5) "four"
```

这个程序的输出显示如图5.2.3所示。

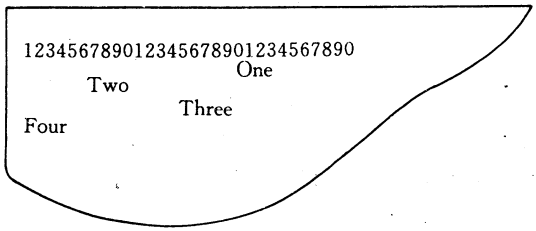


图5.2.3 例5.2.4程序的输出显示格式

2. SPC 函数

SPC 函数的格式是

SPC（n）

参数 n 是0—32767范围内的数、变量或表达式。SPC 使用在 PRINT 和 LPRINT 语句中，它的功能是从当前显示或打印位置起，跳过 n 个空格。

例 5.2.5 本例说明 SPC 函数的功能。

```
PRINT "123456789012345678901234 5"
PRINT "Over"; SPC (15); "Theve"
END
```

执行上列程序的第二个 PRINT 语句时，首先在1—4格（列）的位置显示 "Over"，然后跳过15个格，即跳过5—19列，在20—24格（列）的位置上显示 "There"，程序的输出如图5.2.4所示。

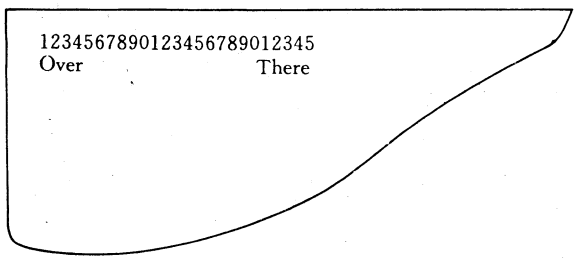


图5.2.4 例5.2.5程序的输出

如果参数 n 大于规定的显示、打印宽度 width（如大于80），则取 n MOD width 的值作为函数的实际参数，这与 TAB 函数类似。

三、按给定格式输出数据

上面我们介绍了 PRINT（LPRINT）语句的基本输出格式和如何使用这两个语句中的 TAB、SPC 函数。此外，还要介绍具有能按着给定的格式显示和打印的特殊输出格式的输出语句，PRINT

USING 语句和 LPRINT USING 语句。这两个语句功能基本一样，其差别仅仅是前一个语句在显示器上输出，后一个语句在打印机上输出。

这两个语句的书写格式是：

PRINT USING formatstring；expressionlist [{，|；}]

LPRINT USING formatstring；expressionlist [{，|；}]

其中 formatstring 是特定的字符串或字符串变量，这些特定的字符串或变量决定了要显示或打印的信息的格式，我们可以把这种特定的字符串叫作格式字符串。下面还要对格式字符串作具体介绍。

expressionlist 是要显示或打印的数据的一些表达式，表达式之间用逗号或分号分隔开。

语句末尾也可以使用逗号或分号，这些标点符号的作用和前面介绍的 PRINT 语句一样，这里就不重复了。

当 PRINT（LPRINT）USING 语句用于显示（打印）字符串型数据时，有三种特殊输出格式。这三种格式相应的字符串及其意义列于表5. 2. 1中。

表 5. 2. 1

格式字符串	说　　明
!	显示或打印所给定字符串的第一个字符。
\ \	显示或打印所给定字符串的前 n＋2个字符，其中 n 是两个斜号间的空格数，如果字符串长度大于 n＋2，则多余的字符忽略不计，如果字符串长度小于 n＋2，则在字符串尾部用空格补到 n＋2个字符。
&	输出字符串的全部字符

例 5. 2. 6　应用 PRINT USING 语句显示字符串型数据的例子。

```
' using the three string-formating characters to
' modify the appearance of printed output.
```

```
A$ = "Look": B$ = "Out": V$ = "&"
PRINT USING "!"; A&; B$
PRINT USING "\  \"; A$; B$ ' Two spaces bettween
                          ' back-slashes.
PRINT USING "\   \"; A$; B$; "!!" ' Three spaces
PRINT USING "!"; A$;
PRINT USING V$; B$
END
```

这个程序的输出是：

```
LO
Look Out
Look Out !!
LOut
```

当用 PRINT（LPRINT）USING 语句显示（打印）各种类型的数值量时，格式字符串 formatstring 由表5.2.2的10种字符（串）组合而成。

表 5.2.2

字符(串)	说　　明
#	它代表一个数字的位置。例如"##.##"表示输出格式为两位整数(包括负号)和两位小数。整数部分的位数少于#规定的位数时，数前用空格补足；超过时，数前输出标记%。小数部分的位数少于#规定的位数时，用0补齐；超过时，四舍五入后，丢掉多余位。
.	显示(打印)小数点，规定小数点的位置。
+	使要显示(打印)的数的符号(加号或减号)放在这个数的前面(如果加号出现在格式字符串的前面)或者后面(如果加号出现在格式字符串的后边)。
−	减号在格式字符串的末尾时，显示(打印)的负数的后面带有减号。
* *	格式字符串以两个星号开头，使输出数字前的空格，用星号补齐。星号本身还有#的作用。

字符(串)	说　　明
$ $	格式字符串以两个美元符号开头，可使输出的字冠以符号$。$ $在格式字符串中也起两个#的作用。
* * $	以符号* * $开头格式字符串，具有* * 和$ $ 两种情况的功能。
^^^^	四个乘幂符号放在数字位置符号#之后，说明数据要用指数形式输出，指数为两位数字。用五个乘幂符号时，给出为三位指数。
,	如果逗号在格式字符串中小数点的左边，可使数字的整数部分从低位开始，每3位输出一个逗号。如果逗号出现在格式字符串的末尾，则它作为字符串的一部分被输出出来。逗号在指数格式中不起作用。
—	格式字符串中的下横线，表示它后面的一个字符须直接输出。

在格式字符串中，除了使用表5.2.2中字符组合外，还可以包含所想要显示或打印的任何字符串常数。例如，语句

```
PRINT USING "This is example —# #"; 1
```

的输出是

```
This is example #1
```

下边的例子给出了用于输出数值量的各种格式字符串。

例 5.2.7　观察下面程序中，不同格式字符串在输出格式中产生的效果。

```
' Format and print numeric data
PRINT USING "# #. # #"; .78
PRINT USING "# # #. # #"; 987.654
PRINT USING "# #. # #     "; 10.2, 5.3, 66.789, .234
PRINT USING "+# #. # #   "; -68.95; 2.4; 55.6; -.9
PRINT USING "# #. # #-   "; -68.95, 2.4; 55.6; -.9
PRINT USING "* * #. #     "; 12.39, -.9, 765.1
PRINT USING "$ $ # # #. # #"; 456.78
```

```
PRINT USING "**$##.##"; 2.34
PRINT USING "####.##,     "; 1234.5;
PRINT USING "#####, .##"; 1234.5
PRINT USING "##.##^^^^"; 234.56
PRINT USING ".####^^^^-"; -888888
PRINT USING "+.##^^^^"; 123
PRINT USING "+.##^^^^^"; 123
PRINT USING "_!##.##_!"; 12.34
PRINT USING "##.##"; 111.22
PRINT USING ".##"; .999
END
```

请仔细对照程序，分析每一个 PRINT USING 语句的输出格式。这个程序的输出如下：

```
 0.78
 987.65
 10.20      5.30      66.79     0.23
 -68.95     +2.40     +55.60    -0.90
 68.95-     2.40      55.60     0.90-
 *12.4      *-0.9     765.1
   $456.78
 *** $234
 1234.50,   1, 234.50
 2.35E+02
 .8889E+06-
 +.12E+03
 +.12E+003
 !12.34!
 %111.22
 %1.00
```

还必须指出，上述程序中 PRINT USING 语句的格式字符串，可以用字符串变量来代替。语句中的常数数据，也可以用变量、符号常量或表达式。

第三节　READ 语句和 DATA 语句

赋值语句 LET 能为变量赋值，并且有计算功能，是程序中最基本的语句之一，应用频率也最高。然而，在 BASIC 语言中，还有其他方法为变量赋值。

当所要赋给变量的值都是常数(包括数值常数和字符串常数，但符号常数除外)，并且无需进行计算时，便可使用 READ/DATA 语句为变量赋值。尤其是当数据比较多时，用这种语句给变量赋值，会感到更方便。下面我们逐个介绍有关的语句和应用。

一、DATA 语句

使用 READ/DATA 语句给变量赋值的基本思想是，把要赋给变量的数据，按照它们在程序中赋给变量的先后顺序，依次写在一个或若干个 DATA 语句中。DATA 语句按数据的顺序，将它们置入内存的数据区。DATA 语句叫作置数据语句。程序运行过程中，当需要这些数据区内的数据时，即将它们取出，赋给相应的变量。

置数据语句 DATA 的格式如下：

DATA constant1[, constant2] …

其中 constant1，constant2等是常数，常数之间用逗号分隔。当字符串常数中含有逗号、冒号、空格时，应该用双引号把这个常数括起来。

DATA 语句中可以含有空数据项，例如：

DATA 1，2，，4，5，6

在这个语句中，第三个数据是空数据项。当空数据项赋给数字变量时，其值为零。当空数据项赋给字符串变量时，其值为空字符串，

即为“”。

DATA 语句是非执行语句，它的功能是在输入这个语句时实现的。例如，输入下面的语句：

DATA 12.1，1E+7，ABC，“ABC,”，5%，1.2345#

其中的数据便依次存入内存的数据区，如图5.3.1所示。当程序运行时，数据已经在数据区中，需要时随时可取出。因此，DATA 语句写在程序的什么地方是没有关系的，而且，这些数据既可以写在一个 DATA 语句中，也可以写在几个 DATA 语句中，如图5.3.1所表示的那样。

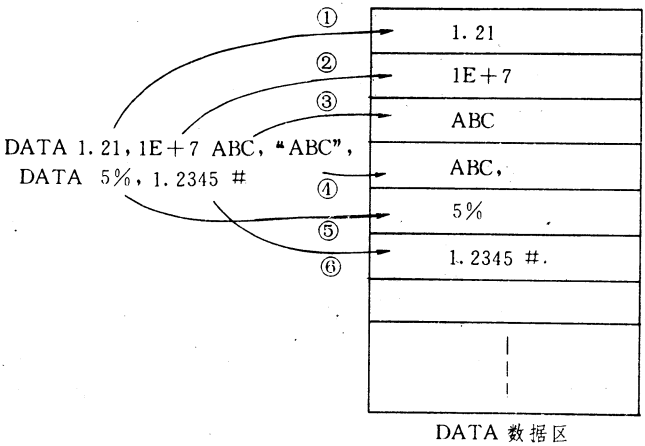


图5.3.1 DATA 语句中的数据存入内存数据区

二、READ 语句

READ 语句叫作读数据语句，它的功能是读取由 DATA 语句建立的数据区中的数据并赋给变量。读数据语句 READ 的格式如下：

READ variablelist

其中 variablelist 是用逗号分隔的 BASIC 变量，变量可以是任何类型的，但必须与 DATA 语句中相应数据的类型一致。

例如，下面语句中的数据类型是匹配的。

```
DATA 12.1, 1E+7, ABC, "ABC," 5%, 1.2345#
READ A, B, C$, D$, E%, F#
```

READ 语句执行的结果，将使 DATA 语句中的数据依次赋给 READ 语句中的变量，如图5.3.2所示，这相当于执行了6个赋值语句：

```
A=1.21:B=1E+7:C$="ABC"
D$="ABC,":E%=5:F#=1.2345
```

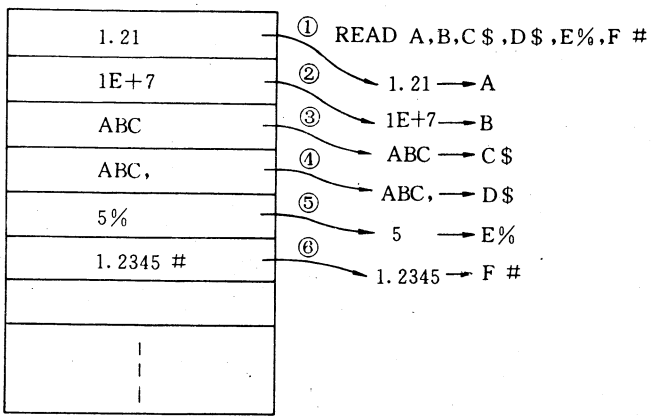


图5.3.2 READ 语句功能示意图

如果读一个字符串赋给数字变量，在程序运行时会出现语法错误信息。如果读一个数值赋给字符串变量，则不会出现错误信息，而是把这个数值看作字符串。

如果读一个单精度数或双精度数赋给整型数变量，则赋值前要进行四舍五入。

当读一字符串赋给定长字符串变量时，如果字符串过长，则

读出的字符串要截尾；如果字符串长度小于字符串变量的长度，则以左侧对齐并用空格填充在串的后边。

一个 READ 语句可以从一个或多个 DATA 语句读取数据，几个 READ 语句也能从一个 DATA 语句读取数据，但一定是顺序地读。

如果 variablelist 中的变量个数超过了 DATA 语句中的数据个数，会产生 Out of Data 的错误信息。DATA 语句提供的数据个数，多于 READ 语句中的变量个数时，多余的数据不产生影响。

下面是一个应用 READ/DATA 语句编程的例子。

例 5.3.1 编写进行 (a+ib)(c+id) 复数运算的 BASIC 程序。已知 a=1，b=2，c=3和 d=4。要求计算结果以如下形式显示在屏幕上：

␣␣␣␣The result is （实部值）+i（虚部值）

(符号␣表示一个空格)。

计算公式：

(a+ib)(c+id) = (ac−bd) +i (ad+bc)

程序如下：

```
DATA 1, 2, 3, 4
READ a, b, c, d
Real=a * c- b * d
Imaginary=a * d+b * c
PRINT TAB (5); "The result is";
PRINT "(";Read;")+i(";Imaginary;")"
END
```

这个程序的输出是：

```
The result is (−5) +i (10)
```

三、恢复数据语句

从以上的讨论可知，DATA 语句中的数据是顺序的读出，并且

每读完一个数据，指针便指向下一个数据，即指针永远指示着下一个要读的数据。这样，数据只能顺序读，并只读一次，对某些程序可能是不方便的，因此，我们希望能在某种程度上控制指针的移动，从而使数据的读取灵活些。

数据恢复语句 RESTORE 可能完成我们所希望的功能。RESTORE 语句的格式是：

RESTORE [{linenumber|linelabel}]

RESTORE 语句有一个任选项，当采用这一项参数时，选用 DATA 语句的行号，或者选用 DATA 语句的标号。

执行没有行号或标号的 RESTORE 语句后，下一个 READ 语句便从第一个 DATA 语句的第一个数据开始读取数据，执行带有行号或标号的 RESTORE 语句后，下一个 READ 语句则从行号或标号指定的 DATA 语句的第一个数据开始读取数据。

有了 RESTORE 语句后，DATA 语句中的数据（全部数据或部分数据）就可以被多次读取了。

例 5.3.2 分析 RESTORE 语句在程序中的作用。对照程序的输出，检查你的分析是否正确。

```
RESTORE There
READ A, B, C
PRINT "A1="; A, "B1="; B, "C1="; C
RESTORE
READ A, B, C
PRINT "A2="; A, "B2="; B, "C2="; C
READ A, B, C
PRINT "A3="; A, "B3="; B, "C3="; C
DATA 57, 66, 32
There: DATA 11, 33, 55
END
```

程序的输出如下：

A1＝11　　B1＝33　　C1＝55

A2＝57　　B2＝66　　C2＝32

A3＝11　　B3＝33　　C3＝55

不难看出，在这个程序中，RESTORE There 语句在程序的一开始就把数据区的指针移到标号为 There 的 DATA 语句中的第一个数据，因此，第一个 READ 语句，读的是 There 行 DATA 语句的三个数据。接着，第二个 RESTORE 语句，又把指针移至数据的头上，于是后边的两个 READ 语句，从头开始，顺序地读取了6个数据。

第四节　用键盘输入方式提供数据

用键盘输入方式提供程序所需要的数据，是在程序运行过程中，通过键盘操作，把数值或字符串常数赋给相应的变量。这是一种人机对话方式输入数据。

实现上述提供数据方式的语句，是键盘输入语句（INPUT）和行键盘输入语句（LINE INPUT）。在程序运行中，遇到 INPUT 或 LINE INPUT 语句时，它提出要求：提供数据给指定变量，并等待操作人员从键盘打入数据。操作人员打入数据后，程序继续往下运行。

在程序运行过程中，由操作人员临时输入数据，而不是事先把数据写在程序里，是一种灵活的提供数据的方法。这种方法可使程序具有通用性，也就是说，程序可以在各种不同数据条件下运行，而不需修改程序。

下面我们介绍有关用键盘输入数据的语句和函数。

一、INPUT 语句

INPUT 语句的格式是：

INPUT [;] ["prompt" {; |,}] variablelist

其中 variablelist 是用逗号分割开的一组变量，用于接收从键盘输入的常数，这一项是必须有的。

prompt 是用户设置的提示输入数据的字符串。它是任选项，选用这一项后，当运行这个语句时，屏幕上将显示用户设置的字符串，起着提醒用户输入适当数据的作用。

紧跟在 INPUT 后面的分号也是任选的。它的作用是：用户输入完数据并按回车键后，使光标仍停留在原来的行，从而使以后所要显示的内容接在当前行的后面。

在提示字符串和变量之间，要选用一个分号或者一个逗号。分号的作用是：在提示字符串后面显示一个问号（提示符）；逗号的作用是：不在提示字符串后面显示问号。

INPUT 语句的功能是暂停程序的运行，根据语句格式显示提示字符串及提示符，表示等待用户输入数据，这时，用户可以从键盘上输入所需数据。输入的数据由 INPUT 语句依次地赋值给变量表中的变量。

下面是几个 INPUT 语句的例句：

```
INPUT A，B，C
INPUT “values of A，B and C”；A，B，C
INPUT；“Radius=”，R
```

执行第一个 INPUT 语句，显示器上出现提示符——问号，等待输入。这时用户要输入三个数据，数据之间用逗号分隔开，输入完最后一个数据，按回车键，于是三个输入的数据依次赋给变量 A，B 和 C，程序继续往下运行。

执行第二个 INPUT 语句，显示器屏幕上显示出提示字符串 values of A，B and C，在提示字符串后显示提示符（问号），即有如下的显示：

```
values of A，B and C?
```

显然，这个提示信息是提醒用户正确输入数据。

执行第三个 INPUT 语句时，在提示字符串 Radius 后面没有提示符（问号），因为提示字符串后面是逗号，而且，输入完数据并按回车键之后，光标仍停留在该输入行，因此，如果程序中有

PRINT 语句，它将继续在本行输出。这是 INPUT 后面分号的作用。前两个 INPUT 后面没有分号，因此，后面 PRINT 语句的输出将在另一行上显示。

例 5.4.1 编写一个计算 X^2的程序，X 值在程序运行中临时输入。

使用最简单的 INPUT 语句，程序如下：

```
INPUT X
PRINT X "SQUARED IS" X ^ 2
END
```

程序开始运行后，显示出问号，便是提示输入 X 值。例如求5^2，打入5并按回车键，于是程序继续运行，显示出运算结果。屏幕的显示如下：

```
? 5
5 SQUARED IS 25
```

例5.4.1中的 INPUT 语句没有使用提示字符串，在出现问号后，可能一时想不起：程序要求什么数据、几个数据，特别是在程序中有多个 INPUT 语句时，这个问题就更突出。有了提示字符串，就解决了这个问题。

例 5.4.2 计算不同半径时的圆面积，试编写一通用程序。

INPUT 语句是实现通用程序的方便手段。程序如下：

```
PI=3.14
INPUT "what is the radius"; R
A=PI * R ^ 2
PRINT "The area of the circle is"; A
END
```

下面是输入半径 R 的值为7.4时的显示情况：

```
What is the radius? 7.4
The area of the circle is 171.9464
```

现在，我们改变一下 INPUT 语句的格式和 PRINT 语句的输出

形式，可得到下面的程序：

```
PI=3.141593
PRINT "Enter radius."
INPUT; "If radius=", R
A=PI * R ^ 2
PRINT ", the area of the circle=" A
END
```

程序运行举例（用户输入半径等于3）：

```
Enter radius
If radius=3, the area of the circle=28.27434
```

在给 INPUT 语句输入数据时，输入数据的个数必须与语句中的变量数一致，数据类型与变量类型一致。否则，会给出错误信息：

```
Redo from start
```

要求用户再次从头输入数据，并且再次显示提示字符串和提示字符。

每输入一个数据后，都要按逗号键，输入完最后一个数据，必须按回车键。

在输入字符串常数时，一般可不必用双引号把字符串括起来，但是，字符串常数头或尾有空格时，则必须用双引号把字符串括起来。字符串常数本身不能含有双引号。

二、LINE INPUT 语句

行键盘输入语句 LINE INPUT 的格式是：

LINE INPUT [;] ["prompt" {; |,}] stringvariable

这个语句的功能是把一个整行（最多255个字符）字符串通过键盘输给一个字符串变量 stringvariable。

语句中的提示字符串 prompt 和紧跟在 LINE INPUT 后面的分号，具有 INPUT 语句中相应部分相同的作用。在提示字符串后，可用分号或逗号，但都不显示提示符（问号）。在提示字符串之后到

回车符之间的所有的输入字符，全部赋给字符串变量。

例 5.4.3 修改例5.4.1的程序。这次用 LINE INPUT 语句来输入打印结果时所需的字符串。

程序如下：

```
PI=3.141593
INPUT "Radius=";R
LINE INPUT "Your Words:";WS$
A=PI*R^2
PRINT
PRINT W$;A
END
```

运行此程序：通过键盘输入，R 赋值为3，WS$ 赋值为 "Area is"，这时，屏幕的显示情况如下：

```
Radius=3
Your Words: Area is
Area is 28.27434
```

三、键盘输入函数 INKEY$

键盘输入函数 INKEY$ 能够从键盘读入一个字符。它的格式是：

```
INKEY$
```

当执行这个语句时，正好键盘输入了一个字符，于是变量接收这个字符。如果用户没有从键盘上输入字符，与 INPUT 语句不同，INKEY$ 不等待用户输入字符，而是取空串作为函数值并继续往下运行程序。

用户从键盘上输入字符时，无需按回车键。用户所输入的字符，不在屏幕上显示。INKEY$ 函数可以接收除下列字符以外的所有字符：

Ctrl+Break

Ctrl＋Alt＋Del

Ctrl＋Numlock

Prtsc

由于函数 INKEY＄不等待键盘输入，为了使它能“捕捉”到键盘输入的字符，就必须反复调用函数去接收字符，这是使用 INKEY＄函数的特点。

在下一章，我们将看到应用 INKEY＄函数的程序例题。

第五节　简单程序举例

例 5.5.1　编写计算图5.5.1所示阴影部分面积的 BASIC 程序。

计算方法：先分别计算出正方形和圆的面积，然后将两个面积相减，所得之差，即为所求面积。

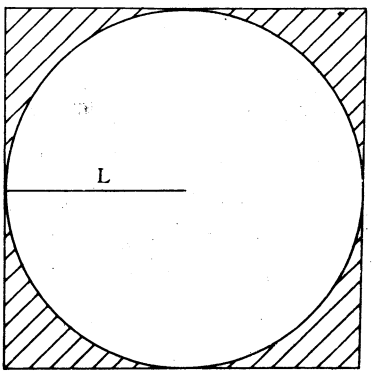


图5.5.1　例5.5.1题图

程序如下：

PI＝3.141593

L＝10

```
Cir=PI * L ^ 2
Squ=4 * L * L
Area=Squ-Cir
PRINT "The Area is"; Area
END
```

程序的输出为如下形式：

```
The Area is 85.8407
```

在上面的程序中，变量 L 的数值是用赋值语句提供。下面的程序方案是应用 READ/DATA 语句为变量 L 提供数据：

```
PI=3.141593
READ L
Cir=PI * L ^ 2
Squ=4 * L * L
Area=Squ-Cir
PRINT "The Area is"; Area
DATA 10
END
```

如果用 INPUT 语句给变量 L 输送数据，这个程序就可以不加修改的计算不同 L 值时的面积。这是一种通用程序的设计方法。下面的程序是采用 INPUT 语句设计的，而且还要求计算结果为整数，输出形式也有所改变。修改后的程序和输出如下：

```
INPUT "Enter the radius:", L
PI=3.141593
Cir=PI * L ^ 2
Squ=4 * L * L
Area=CINT (Squ-Cir)
LPRINT "When L="; L; ", the Area is"; Area; "."
END
```

When L=10, the Area is 86.

下面的程序使用了 PRINT USING 语句输出计算结果，但要求取小数点后两位：

```
INPUT "Enter the radius;", L
PI=3.141593
Cir=PI * L ^ 2
Squ=4 * L * L
Area= (Squ-Cir)
PRINT USING "When L=##.##"; L;
PRINT USING ", the Area is +##.##_."; Area
END
```

When L=10.00, the Area is +85.84.

我们还可以用函数 INT 实现，取小数点后两位精度。具体方法是先将 Area 乘以10^2，加上0.5，解决四舍五入问题，最后再被10^2除，即

```
Area=INT (Area * 1E+2+0.5) /1E+2
```

例 5.5.2 编写一个 BASIC 程序，其功能是将华氏温度(F)换算为摄氏温度 (C)。

计算公式

$$C=\frac{5}{9}(F-32)$$

我们用 INPUT 语句设计成通用程序。程序如下：

```
INPUT "F=", F
C=5 * (F-32) /9
PRINT "F="; F; ","; TAB (10); "C="; C
END
```

它的输出形式如下（当 F 输入为68度时）：

```
F=68,        C=20
```

例 5.5.3 已知电感、电容串联电路中的电感 L=0.01亨，电容 C=3×10^{-6}法，试编写一个计算电路振荡频率 F 的 BASIC 程序。

计算公式

$$F=\frac{1}{2\pi}\left(\frac{1}{LC}\right)^{1/2}$$

一个可能的程序方案是由 READ/DATA 语句给定 L 和 C 的值。程序及运行结果如下：

```
PI=3.141593
READ L, C
f=SQR (1/ (L * C)) / (2 * PI)
PRINT "Frequency="; f; "Hz"
DATA 0.01, 0.000003
END

Frequency=918.8814 Hz
```

例 5.5.4 有两个物体，其质量分别为 m_1=1千克，m_2=500千克，它们之间的距离 r=5米。试根据万有引力定律计算这两个物体之间的引力。要求用双精度数进行计算。

计算公式

$$F=G\frac{m_1\cdot m_2}{r^2}$$

$$G=6.67\cdot10^{-11}\frac{\text{牛顿}\cdot\text{米}^2}{\text{千克}^2}$$

程序及运行结果如下：

```
DEFDBL A-Z
READ G, m1, m2, r
Gravitation=G * m1 * m2/r ^ 2
PRINT USING "Gravitation= * * # #.# # # # # # ^ ^ ^ ^ ";Gravitation
```

```
DATA 6.67e-11, 1, 500, 5
END

Gravitation= * 133.40000D-11
```

例 5.5.5. 设计一个BASIC 程序，计算图5.5.2的总电阻 R 和电流 I。

计算公式

$$\frac{1}{R}=\frac{1}{R_1}+\frac{1}{R_2}+\frac{1}{R_3}+\frac{1}{R_4}$$

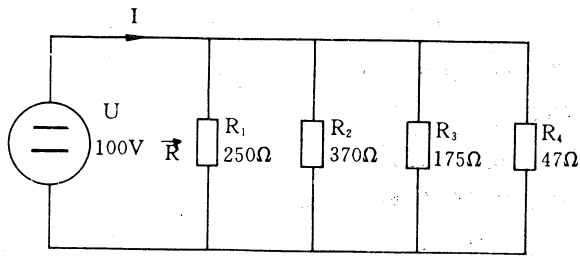


图5.5.2　例5.5.5的题图

$$I=\frac{U}{R}$$

程序及运行结果如下：

```
READ R1, R2, R3, R4
READ U
P=1/R1+1/R2+1/R3+1/R4
R=1/P
I=U/R
PRINT "R="; R; " (OM)"; SPC (10); "I="; I; " (A)"
DATA 250, 370, 175, 47, 100
END

R=29.67924 (OM)        I=3.369358 (A)
```

小　　结

本章以学习简单程序设计方法为主线，介绍了赋值语句、输出语句、注释语句、结束语句、输入数据语句和与输入输出有密切关系的三个函数。

学习程序设计的最好方法是，首先掌握好语句、函数的格式和功能，然后分析研究例题，进一步熟悉语句和函数的格式和功能，体会编制程序的方法，最后，自己试着去设计新程序并上机检验自己所设计的程序。

下面把本章介绍的几个语句的格式归纳如下：

赋值语句：

[LET] 变量＝表达式

其功能是将表达式的值赋给变量。

显示器输出语句：

PRINT [表达式组] [{; |,}]

其功能是将表达式的值显示在屏幕上。

打印机输出语句：

LPRINT [表达式组] [{; |,}]

其功能是将表达式的值在打印机（LPTI）上打印出来。

给定格式显示输出语句：

PRINT USING formatstring; expressionlist [{, |;}]

其功能是按给定的格式将表达式的值输出到显示器屏幕上显示出来。

给定格式打印输出语句：

LPRINT USING formatstring; expressionlist [{; |,}]

其功能是按给定的格式将表达式的值输出到打印机上打印出来。

置数据语句：

DATA constant1 [, constant2] …

其功能是将数据置入内存的数据区，供 READ 语句使用。

读数据语句：

READ variablelist

其功能是读取由 DATA 语句存入数据区的数据并赋给变量。

恢复数据语句：

RESTORE [{linenumber|linelable}]

其功能是将数据区的数据指针移至指定的 DATA 语句或第一个 DATA 语句，并从那里开始读取数据。

键盘输入语句：

INPUT [;] ["prompt" {; |,}] variablelist

其功能是等待和接收用户从键盘为变量输入的数据，然后将它们赋给相应变量。

行键盘输入语句：

LINE INPUT [;] ["prompt" {; |,}] string variable

其功能是等待和接收用户从键盘输入的一行字符串，然后将它赋给字符串变量。

本章还介绍了两个与输出有关，并且只能用在 PRINT 和 LPRINT 语句中的函数，这两个函数是：TAB（n）和 SPC（n）。

函数 TAB（n）可将要显示或打印的数据，从第 n 列的位置开始。

函数 SPC（n）的功能是从当前的显示或打印的位置起始，跳过 n 列再显示或打印下一个数据。

另外，还提到一个与输入有关的键盘输入函数 INKEY $，这个函数用于从键盘上"捕捉"一个用户输入的字符。它的特点是用户输入一个字符后，不要按回车键，输入的字符不显示，这个函数也不使程序停下来等待输入。INKEY $ 函数为"捕捉"到键盘输入的字符，通常要多次重复执行。此函数的进一步应用，将在以后的章节中介绍。

在程序的任何地方，都可以用注释语句给程序加上说明文字。

它有两种格式：

REM 说明文字

' 说明文字

在程序运行结束的地方，要使用结束语句，结束语句 END 的执行，将使程序由运行状态返回 BASIC 环境。如果程序结束运行的地方就是程序的结尾，END 语句可以省略。

习　　题

5-1 按格式写出下列程序的输出。

(1)
```
X=2:X=10
PRINT X
END
```

(2)
```
Y=7:X=Y+3
PRINT X+1, Y+1;
PRINT X * Y
END
```

(3)
```
A=.5:I=3
B=2 * A:B=I+B
A=A/2:I=A
PRINT TAB(5);"a=";A,"b=";B,"i=";I
END
```

5-2 按格式写出下列程序的输出。

(1)
```
A$ = "!" :B$ = "BASIC" :C$ = "Program"
A1$ = "&" :A2$ = "\ \"
A3$ = "This is my &"
PRINT USING A$; B$, C$
PRINT USING A2$; B$, C$
PRINT USING A1$; B$; C$
PRINT USING A3$; C$
```

```
END
```

(2)
```
A=1.23:B=4.56
V$="JOHN++**$###.####-A HOUR"
LPRINT USING V$; A*B+B
```

(3)
```
F=68:C=5*(F-32)/19
PRINT USING "F=+###.##,"; F;
PRINT USING "C=+###.##"; C
END
```

5-3 编写一个输出下面运行结果的程序（标准格式）。

A=3　　B=5　　A+B=8

5-4 编制计算半径 r=4cm 的球体积的 BASIC 程序。计算公式

$$V=\frac{4}{3}\pi r^3$$

5-5 编制计算

$$Z=\frac{e^{ax}-e^{-ax}}{2}\sin(x+a)+a\ln\frac{x+a}{2}$$

的 BASIC 程序。假设 a=0.1，x=2。

5-6 按下列公式编写自由落体位移的 BASIC 程序。

$$S=\frac{1}{2}gt^2tV_0t+S_0$$

已知 g=9.81米/秒^2，S_0=2.0米，V_0=5.0米/秒，t=4.5秒。

5-7 编写一个 BASIC 程序，将角度47°23′15″换算为弧度，结果取三位小数。先用 READ/DATA 语句编写，然后再改用 INPUT 语句，使之成为通用程序。

5-8 试编写一个 BASIC 程序，将摄氏度数换算为华氏度数。要求打印输出的形式为

Centigade ××=Fahrenheit ××

5-9 编写一个 BASIC 程序，其功能是将弧度表示为角度（度、

·分、秒)。

5-10 写出计算下列算式的程序。

(a) $1+\frac{1}{2}+\frac{1}{2\times3}+\frac{2}{2\times3\times4}+\frac{1}{2\times3\times4\times5}$

(b) 1+2+3+4+5

第六章　分支程序设计

分支程序具有比简单程序更为复杂的程序结构，由于它具有逻辑判断能力，从而使它能解决更为复杂的问题。

分支程序不再只是简单地按语句顺序逐句执行。对于结构比较复杂的程序，人们常借助于流程图，直观地表示其结构和功能。本章首先介绍流程图的画法。然后重点介绍与构成分支程序有关的BASIC语句，如GOTO语句，IF语句和SELECT CASE语句。详细地讲解了应用这些语句设计分支程序的方法。

为配合分支程序设计的学习和不断丰富语言手段，本章还介绍了RANOMIZE语句、TIMER函数和RND函数及INKEY$函数的应用。

第一节　流程图

简单程序是按程序语句的顺序逐条执行的。虽然这种程序的结构非常简单，但是，为了顺利的进行设计，减少程序的差错，又快又好地完成程序设计任务，对于比较复杂的问题，在着手编写程序之前，先拟定出准确、详细的计算步骤，并且把它们形象、直观地用图形表示出来，使计算方法和步骤一目了然。这种图形叫作程序框图或流程图（Flow Chart）。

显然，流程图能帮助我们正确地编写程序，也能帮助我们调试程序。

流程图不仅在设计过程中起着重要作用，而且也是一种互相交流的手段。配有流程图的程序，可使别人易读、易懂。

为使程序框图成为程序设计人员相互交流的一种共同语言，必须对程序框图的画法作一些规定。目前虽然尚无完全统一的规定，但在各种文献中，框图的画法差别不大。这里介绍的就是一些比较广泛应用的画法。

程序框图主要包含下列几种类型的框：

(1) 开始框；　　(2) 结束框；

(3) 输入框；　　(4) 输出框；

(5) 运算框；　　(6) 分支框。

上述各种类型框的符号如图 6.1.1 所示。

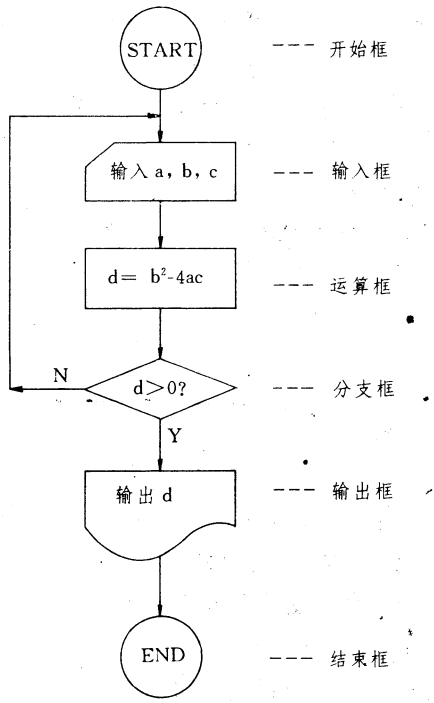


图 6.1.1　流程图各种类型框的符号

程序框图总是以“开始框”开头，以“结束框”结束。这两个框通常用圆圈表示，里面注明“开始”和“结束”字样或相应的英文单词“START”和“END”。框图中带箭头的直线，表示程序的流向，即程序的执行顺序。

运算框表示执行某种确定的运算操作，通常用矩形框表示，框内简明地表示出运算操作内容。其表示方法，既可用数学式也可用文字，文字说明必须简单明了，数学式也可有多种写法。以图6.1.1中的运算框为例，运算框的内容还可以写为

$$b^2-4ac\to d \text{ 或 } d\leftarrow b^2-4ac \text{ 或 } d=b^2-4ac$$

输入框和输出框用来表示程序所进行的输入和输出操作的内容。可以用平行四边形框表示，但是，为了使输入操作和输出操作在框图上更明显地表示出来，输入框往往模仿计算机输入卡片的形状，而输出框模仿一段打印机纸。

分支框用来表示由逻辑判断的结果而产生的不同分支流向。分支框为菱形，里面写着逻辑判断的内容，分支上写着逻辑判断的结果。

现举例说明程序流程图的画法。

例 6.1.1 画出求方程式 $ax^2+bx+c=0$ 的根的程序流程图。

方程式的两个根 x_1 和 x_2 分别为

$$x_1=\frac{-b+\sqrt{b^2-4ac}}{2a}$$

$$x_2=\frac{-b-\sqrt{b^2-4ac}}{2a}$$

并且，当 $b^2-4ac<0$ 时，方程的根为复数。在计算机程序中，一个复数我们用两个数分别表示其实部值和虚部值。

根据以上计算方程式根的公式，不难写出如下的计算步骤：

(1) 给定方程式的系数 a，b 和 c；

(2) 计算 $d=b^2-4ac$；

(3) 计算 $e=-\frac{b}{2a}$；

（4）判断 d 是否为负数；如果是，转去做第 8 步，否则，继续往下做第（5）步；

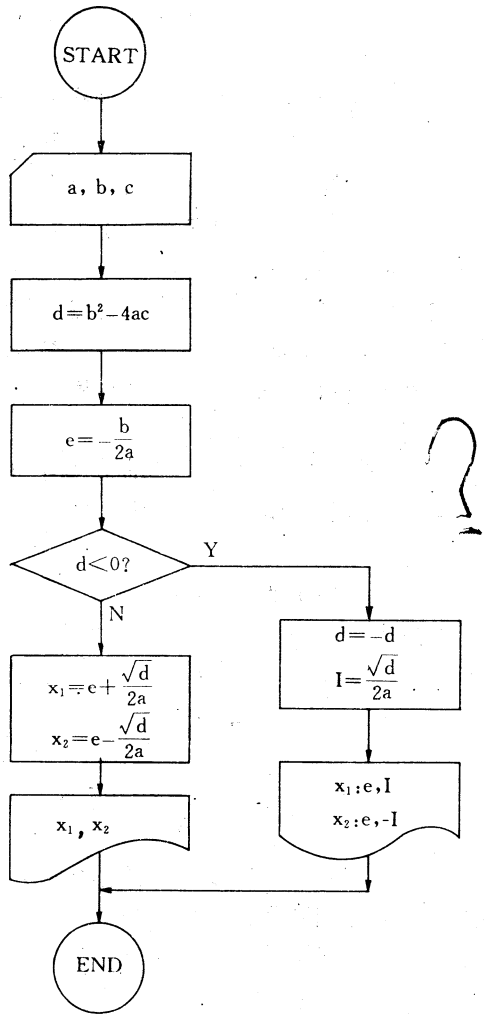


图 6.1.2　例 6.1.1 的程序框图

(5) 计算 $x_1=e+\frac{\sqrt{d}}{2a}$，$x_2=e-\frac{\sqrt{d}}{2a}$；

(6) 输出计算结果 x_1 和 x_2；

(7) 结束或转去做第 10 步；

(8) 计算复根：令 d=−d

$$x_1=e+i\frac{\sqrt{d}}{2a},\quad x_2=e-i\frac{\sqrt{d}}{2a}$$

(9) 输出计算结果：两个复数根；

(10) 结束。

按照上述计算步骤，不难画出图 6.1.2 所示的框图。

一个菱形的分支框，可以表示出两个或三个分支，当有更多的分支时，就要用多个菱形的分支框来表示。下面是一个多分支问题的例子。

例 6.1.2 设有如下函数

$$y=\begin{cases}3x & (0\leqslant x<2)\\ \dfrac{4x+10}{3} & (2\leqslant x<5)\\ \dfrac{3}{5}x+7 & (5\leqslant x<10)\\ \dfrac{x+68}{6} & (x\geqslant 10)\end{cases}$$

试画出计算函数 y 的分支流程结构。

由例 6.1.1 可知，一个逻辑判断，要用一个菱形分支框来表示，判断结果形成两个分支。在本例中有四个逻辑判断，因此，要用四个菱形分支框，依次进行判断，共可形成五个分支，如图 6.1.3 所示。

为简化分支程序的流程图，特别是在多分支的情况下，还常采用图 6.1.4 所示的画法，这种表示方法既简单又清楚。

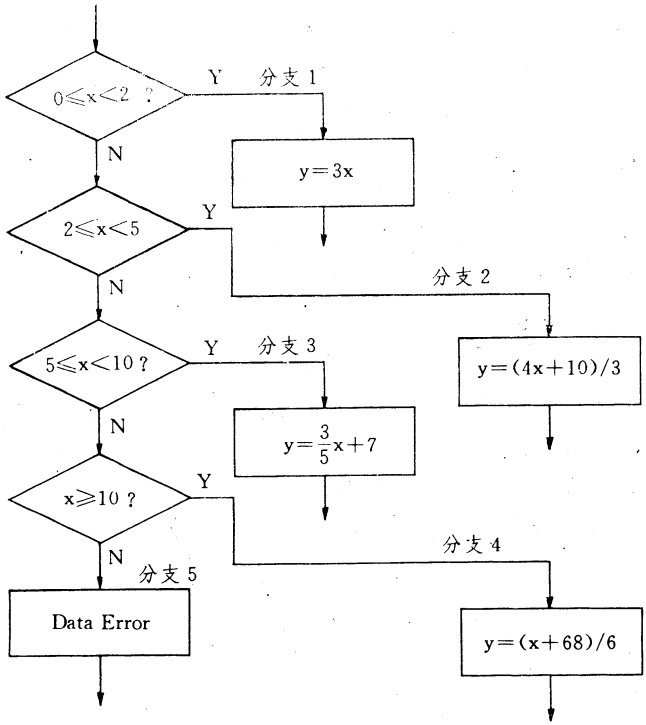


图 6.1.3　例 6.1.2 的分支流程结构（一）

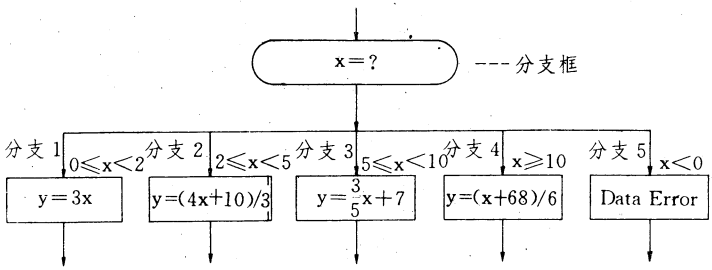


图 6.1.4　例 6.1.2 的分支流程结构（二）

第二节　GOTO 语句和 IF 语句

从上一节的程序流程图可以看出，程序中有了逻辑判断后，可形成程序的分支，根据逻辑判断的结果，决定执行若干程序分支中的哪一个分支，这种结构的程序就是分支程序。

为了在程序上实现分支流程的控制，BASIC 语言提供了具有逻辑判断和控制流程转移功能的语句。在这一节我们介绍这些语句的格式、功能和应用。

一、无条件转移语句 GOTO

这个语句的用途是无条件的转移到指定的程序行去继续运行程序，也就是说改变了简单程序中只能顺序执行各程序行的情况。

语句的格式是：

GOTO {linelabel|linenumber}

在 GOTO 语句中，指定的程序行可用行标号（linelabel）表示，也可以用行号（linenumber）表示。例如，

```
GOTO start
GOTO 150
```

执行第一个 GOTO 语句时，程序转移到标号为 start 的行，从那里继续运行。执行第二个 GOTO 语句时，程序转移到行号为 150 的程序行，从那里继续运行。

例 6.2.1　计算半径 R 分别为 5，7，12 时的圆面积。

```
DATA 5, 7, 12
Start:
   READ R
   PRINT "R="; R,
   A=3.14 * R ^ 2
   PRINT "Area="; A
```

GOTO Start

这个程序把三个数据（圆的半径R）放在DATA语句中。用READ语句读出一个数据，计算圆面积，输出计算结果，然后，用GOTO语句，转移到行标号为Start的程序行，读取下一个数据，计算和输出结果。当第三个数据读出并计算、输出完毕后，再次转移到Start行时，接着又执行READ语句。但是，当DATA语句中，已经没有未被读过的数据时，将出现一个如图6.2.1所示的错误信息对话框，告诉你DATA语句中的数据已读完：

Out of DATA

同时，程序停止运行。

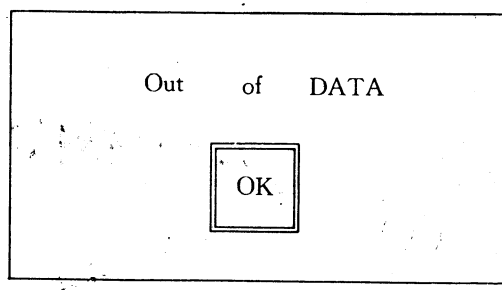


图 6.2.1 超出数据对话框

按回车键、Esc键或空格键，可清除对话框，恢复编辑屏幕，这时还会看到，相应的READ语句是高亮度显示的。

按F4键，可以看到显示在输出屏幕上的计算结果：

R＝5　　Area＝78.5

R＝7　　Area＝153.86

R＝12　　Area＝452.16

图6.2.2是本程序的流程图。

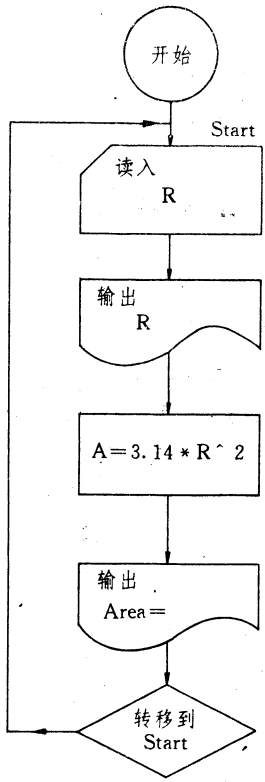


图 6.2.2　例6.2.1程序的流程图

GOTO 语句提供了一个无条件地转移到另一个行语句的手段，但是，GOTO 语句不能进入或退出子程序和函数过程 (SUB 和 FUNCTION)，也不能进入或退出多行 DEFFN 函数，只能在它们的内部使用。

较好的编程方法，是用后面将要介绍的结构化的流程控制语句来代替 GOTO 语句，因为带有许多 GOTO 语句的程序，很难阅读和调试，所以在程序中应尽量少用 GOTO 语句。

二、IF…THEN…ELSE 语句

IF…THEN…ELSE 叫作条件转移语句。这个语句有两种格式：一种是单行格式，一种是块（Block）格式，现分述如下。

1. 单行格式的 IF…THEN…ELSE 语句

其语句格式为

IF boolean expression THEN thenpart [ELSE elsepart]

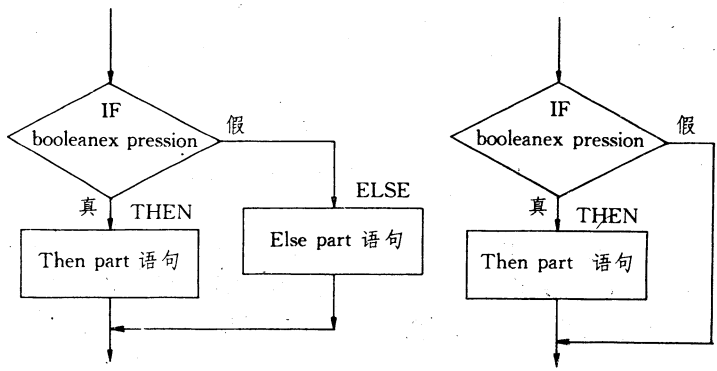
语句中 boolean expression 是任意可求值为真（非零）或假（零）的布尔表达式（逻辑表达式）。

thenpart 是当上述表达式的值为真时所执行的语句或分支。

elsepart 是当上述表达式的值为假时所执行的语句或分支。

ELSE 部分是任选的。若语句中没有给出 ELSE，当 booleanexpression 为假时，控制就转移到下一个语句行。

上述的 IF…，THEN…ELSE 语句的功能，还可以用图6.2.3所示的框图表示出来。



(a) IF … THEN … ELSE 语句　　(b) IF … THEN 语句

图 6.2.3　IF…THEN…ELSE 语句的功能

例 6.2.2　有两种规格不同的纸张 A 和 B，A 纸每张价格为 0.05元，B 纸每张价格为0.055元。为了某种用途，若买 A 纸，需

1200张，若买B纸，需1100张。试编一BASIC程序，确定买哪种纸较为经济。

首先计算出1200张A纸的价钱A和1100张B纸的价钱B，然后比较A和B两数的大小，输出其中较小者。其程序框图如图6.2.4所示。

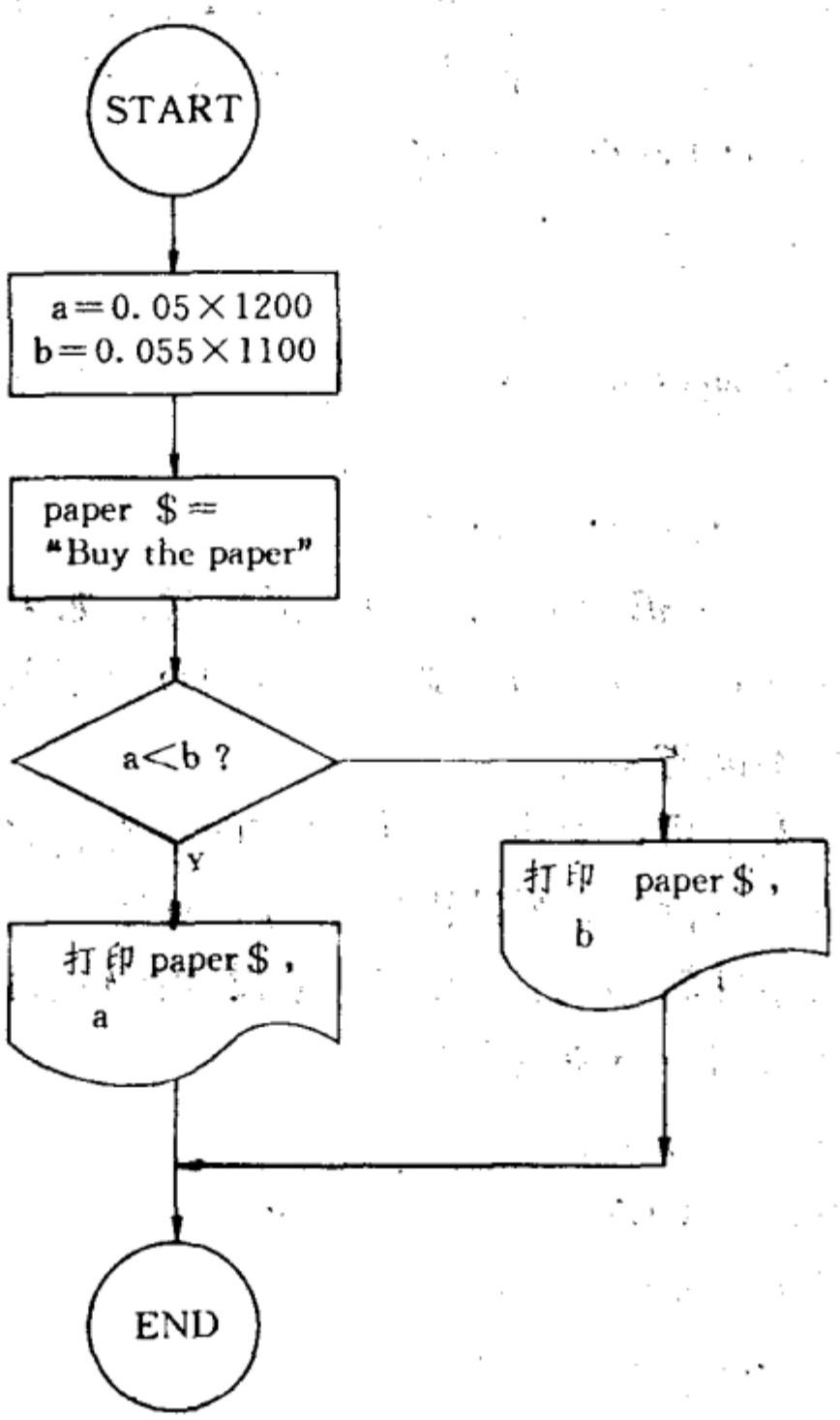


图 6.2.4　例6.2.2程序的流程图

按框图写出程序如下：

```
a=.05*1200:b=.055*1100
paper $ = "Buy the paper"
IF a < b THEN PRINT paper $ ; "A =" ; a ELSE PRINT
```

```
paper $ ; "B="; b
    END
```

程序的运行结果是：

```
Buy the paper A=60
```

在上边这个程序中，IF…THEN…ELSE 语句很长，但避免了使用 GOTO 语句。下面将语句改为 IF…THEN 形式，则有

```
    IF A<B THEN GOTO 100
    PRINT paper $ ; "B="; b
    GOTO 10
100 PRINT paper $ ; "A="; a
10 END
```

这里我们不得不使用 GOTO 语句。

在 THEN 或 ELSE 之后用 GOTO 语句时，如果 GOTO 之后是行号，则关键字 GOTO 可以省略，但 GOTO 之后是行标号时，关键字 GOTO 是不能省略的。

从上面的例子可以看出，单行格式的条件转移语句，对于复杂的判断是不太方便的，而且程序结构不清楚、不易读。下面将要介绍块结构的条件转移语句，可容易地设计出较好的结构化程序。

2. IF…THEN…ELSE 语句块

语句块的格式如下：

```
IF condition1 THEN
        [statementblock－1]
[ELSEIF condition2 THEN
        [statementblock－2]]
          ⋮
[ELSE
        [statementblock－n]]
END IF
```

语句中的 condition1，condition2等等，都是 BASIC 表达式，可以是数值表达式，也可以是逻辑表达式。当它们是数字表达式时，非零值为真，零值为假。当它们是逻辑表达式时，真是－1，假是0。

语句中 statementblock－1，statementblock－2等等，是一组 BASIC 语句。

在这个条件转移语句块中，除了 IF，THEN 和 END IF，其余部分括在方括号中，表示它们都是任选部分。END IF 表示语句块的结束。

语句的工作过程（即功能）是：当开始执行 IF 语句块时，首先测试 IF 后面的第一个条件（condition1），如果其值为真，则执行 THEN 后面的一组语句（statementblock－1），然后，跳出语句块，继续执行 END IF 后面的语句。如果其值为假，则顺序检查每个 ELSE IF 后面的条件（condition2，condition3…），当发现一个真条件时，就执行其 THEN 后面的语句组，然后，跳出语句块，继续执行 END IF 后面的语句。如果没有一个条件为真，就执行 ELSE 后面的一组语句，然后退出语句块，继续执行 END IF 后面的语句。如果没有 ELSE 语句，就跳出语句块，继续执行 END IF 后面的语句。

在块格式的 IF 语句中，可以根据需要，加入任意多个 ELSE IF。

BASIC 通过查看 THEN 后面的内容，来确认这个 IF 语句是否是块结构。如果在 THEN 后面有除注释以外的其他内容，则认为这是一个单行格式的 IF 语句。

一个最简单的 IF 语句块，至少包含一个 IF…THEN 和一个 END IF。例如，

```
IF x<100 THEN
   PRINT x
   Number=Number+1
END IF
```

如果 x 的值小于100，则做 THEN 后面的两个语句，否则，退出 IF

语句块，执行 END IF 后面的语句。

所有单行格式的 IF 语句，都可以用 IF 语句块来表示。

例 6.2.3 用 IF 语句块改写例6.2.2的程序。

这是一个具有两个分支的分支程序。用 IF 语句块描述时，一般用一个 IF，THEN 和 ELSE 就够了。修改后的程序如下：

```
a=.05*1200:b=.055*1100
paper$="Buy the paper"
IF a<b THEN
    PRINT paper$;"A=";a
ELSE
    PRINT paper$;"B=";b
END IF
END
```

程序中的 IF 语句块，用一个框标了出来。显然，这个语句所表达出的分支结构，要比例6.2.2中的单行格式的结构清晰。

当程序有两个以上分支时，可以通过在语句中增加 ELSE IF 子句的办法解决。

例 6.2.4 编写计算方程式 $ax^2+bx+c=0$根的 BASIC 程序。要求分别考虑下述三种情况：

(1) 两个不同的实根；

(2) 两个相等的实根；

(3) 两个复数根。

对图6.1.2稍加修改，就可得到本例题的流程图，如图6.2.5所示。

方程的系数 a，b 和 c 用 INPUT 语句输入。要求程序输出打印系数 a，b，c 的值和方程式的根 x_1，x_2。

下面是按流程图编写的程序。

```
INPUT "a, b, c=";a, b, c
```

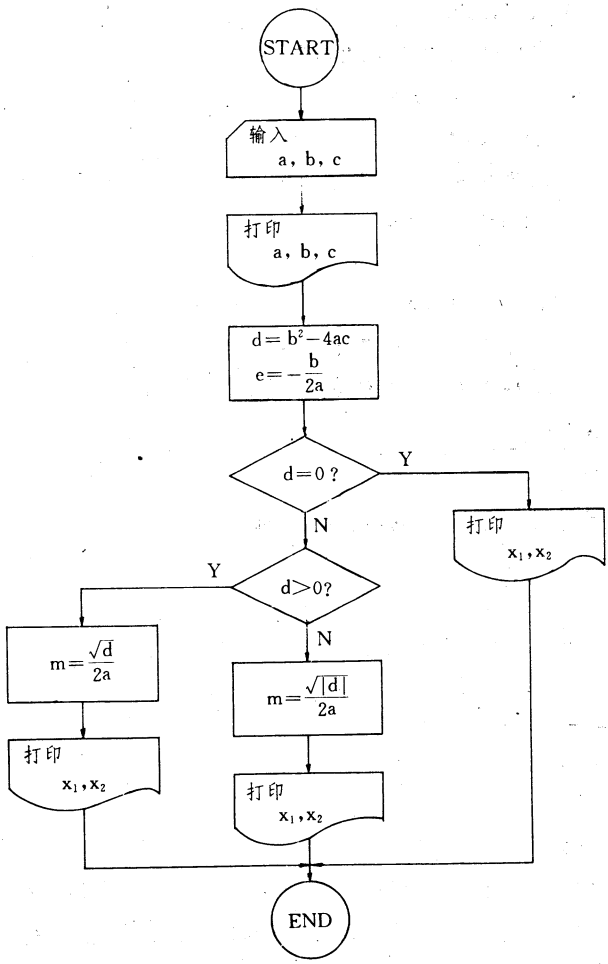


图 6.2.5　例6.2.4程序的流程图

```
LPRINT "a="; a; "b="; b; "c="; c
d=b*b-4*a*c
e=-b/(2*a)
```

```
IF d=0 THEN
      LPRINT "x1=x2="; e
ELSEIF d>0 THEN
      m=SQR (d) / (2*a)
      LPRINT "x1="; e+m
      LPRINT "x2="; e-m
ELSE
      m=SQR (ABS (d)) / (2*a)
      LPRINT "x1="; e; "+i("; m;")"
      LPRINT "x2="; e; "-i("; m;")"
END IF
END
```

程序对三种不同根的输出，有如下的形式：

```
a=22 b=33 c=1
x1=-3.094125E-02
x2=-1.469059
a=1 b=1 c=2
x1=-.5+i (1.322876)
x2=-.5-i (1.322876)
a=1 b=2 c=1
x1=x2=-1
```

块格式的 IF 语句，相对于单行格式的语句，有许多优点，如：

(1) 块格式比单行格式设计出的程序更结构化、更灵活；

(2) 可以方便地测试非常复杂的条件；

(3) 用块格式语句设计出的程序易于阅读、调试和修改。

IF 语句块内还可以包含 IF 语句块，形成多层的嵌套结构。

例 6.2.5 比较输入的任意两个整数 a 和 b，试分析它们属于下列四种情况的哪一种：

(1) a≤50，b≤50；　　(2) a≤50，b>50；

(3) a>50，b≤50；　　(4) a>50，b>50。

完成上述功能的 BASIC 程序如下：

```
INPUT "Enter two integer a and b"; a%, b%
IF a%<=50 THEN
    IF b%<=50 THEN
        PRINT "a<=50, b<=50"
    ELSE
        PRINT "a<=50, b>50"
    END IF
ELSE
    IF b%<=50 THEN
        PRINT "a>50, b<=50"
    ELSE
        PRINT "a>50, b>50"
    END IF
END IF
END
```

在这个程序中，有两个并列的 IF 语句块被嵌套在另一个 IF 语句块中，结构是非常清楚的。

如果把 IF 语句块中的判断条件稍加修改，本例题的程序完全可以用一个单纯的 IF 语句块实现。修改后的程序如下：

```
INPUT "Enter two integer a and b"; a%, b%
IF a%<=50 AND b% <=50 THEN
        PRINT "a<=50, b<=50"
ELSEIF a% <=50 AND b% >50 THEN
        PRINT "a<=50, b>50"
ELSEIF a% >50 AND b% <=50 THEN
        PRINT "a>50, b<=50"
```

```
ELSE
        PRINT "a>50, b>50"
END IF
END
```

第三节　SELECT CASE 语句

SELECT CASE 语句也是一个构成分支程序，特别是构成多分支程序的块结构语句。

SELECT CASE 语句有如下的格式：

```
SELECT CASE testexpression
    CASE expressionlist1
        [statementblock1]
    [CASE expressionlist2
        [statementblock2]]
        ⋮
    [CASE ELSE
        [statementblockn]]
END SELECT
```

语句中 testexpression 是被测试的内容，它可以是数值的或字符串的表达式。

expressionlist 是一个或多个用逗号隔开的表达式，它们可以有以下三种不同的格式：

（1）expression [, expression] …

（2）expression TO expression

（3）IS relation-operate expression

其中 expression 可以是任何数值的或字符串的表达式，但其类型必须与被测试的表达式 testexpression 一致。格式（1）表示所要测试的值，格式（2）给出所要测试值的范围，格式（3）给出所要测试

值的关系式。关系式中允许使用下列关系操作（relation-operate）

<，<=，>，>=，<>，=

如果使用 TO 来表示值的范围，那么，较小的值必须写在前面（第一个）。

在一个 expressionlist 中，可以同时出现上述三种格式。

执行 SELECT CASE 语句时，依次测试每个 CASE 后面的表达式。如果 testexpression 与某个 CASE 相匹配，则执行这个 CASE 后面的语句组 statementblock。然后控制转到 END SELECT 后面的语句。

使用 CASE ELSE 时，当 testexpression 与其他任何 CASE 的 expression 都不匹配时，就执行 CASE ELSE 后面那组语句。

在不使用 CASE ELSE 的情况下，没有一个 CASE 表达式符合 testexpression 的值，即都不匹配，则会出现运行期间的错误信息：

CASE ELSE expected

下面是一个 SELECT CASE 例句：

```
SELECT CASE Testvalue
    CASE 1，3，5，7，9
      PRINT "Odd"
    CASE 2，4，6，8
      PRINT "Even"
    CASE IS<1
      PRINT "Too low"
    CASE IS>9
      PRINT "Too hight"
    CASE ELSE
      PRINT "Not an integer"
END SELECT
```

这个 SELECT CASE 语句测试表达式 Testvalue 的值，产生五个分支。当 Testvalue 的值等于1，3，5，7或9时，输出字符串 "Odd"；

当表达式 Testvalue 的值等于2，4，6或8时，输出字符串“Even”；当 Testvalue 的值小于1时，输出字符串“Too low”；当 Testvalue 的值大于9时，输出字符串“Too hight”；当 Testvalue 的值不属于上述四种情况的任何一种时，执行 CASE ELSE 后面的语句，即输出字符串“Not an integer”。

SELECT CASE 语句的功能，可表示为图6.3.1所示的流程图。

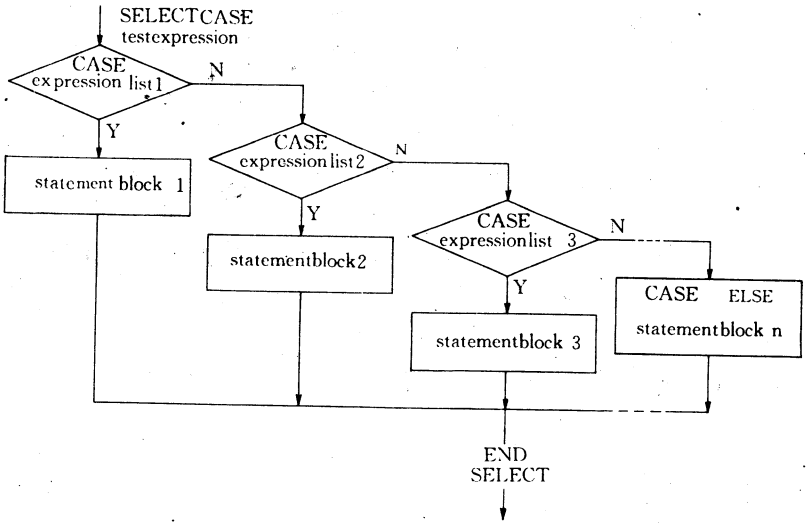


图 6.3.1 SELECT CASE 语句的功能

例 6.3.1 用 SELECT CASE 语句编写例6.2.4的程序。令方程式系数 a，b，c 为整数。

程序如下：

```
INPUT "a, b, c="; a%, b%, c%
d=b% * b%-4 * a% * c%
e=-b%/ (2 * a%)
SELECT CASE d
    CASE 0
      PRINT "x1=x2="; e
```

```
        CASE IS>0
          m=SQR (d) / (2*a%)
          PRINT "x1="; e+m
          PRINT "x2="; e-m
        CASE ELSE
          m=SQR (ABS (d)) / (2*a%)
          PRINT "x1="; e; "+i ("; m;")"
          PRINT "x2="; e; "-i ("; m;")"
    END SELECT

    END
```

例 6.3.2 用 SELECT CASE 语句编写计算例6.1.2函数的BASIC 程序。

图6.1.3和图6.1.4给出了本例题程序框图的核心部分，框图的开头，增加了一个输入变量 x 值的输入框，在每个分支下面，再增加一个输出 y 值的输出框，便可得到一个完整的程序框图。这个任务留给读者来完成。

下面是根据流程图编写的 BASIC 程序：

```
INPUT "Enter x"; x
SELECT CASE x
    CASE 0 TO 2
      y=3*x
      PRINT y
    CASE 2 TO 5
      y= (4*x+10) /3
      PRINT y
    CASE 5 TO 10
      y=3/5*x+7
      PRINT y
```

```
    CASE IS >=10
      y= (x+68) /6
      PRINT y
    CASE ELSE
      PRINT "Data error."
END SELECT

END
```

在这个程序中，若没有 CASE ELSE 部分，当 x 的值为负数时，就会出现错误信息：

CASE ELSE expected

告诉你，需要有 CASE ELSE 这部分语句。

SELECT CASE 语句，类似于 IF 语句块，都能方便地构成多分支程序。凡是能用 SELECT CASE 语句的地方，IF 语句块也能用。

两个语句的主要区别在于，SELECT CASE 语句是对单个表达式求值，根据其结果执行不同的分支语句。而 IF 语句块可以计算非常复杂的表达式。

第四节　RANDOMIZE 语句和 TIMER，RND 函数

前面几节，我们学习了有关构成分支程序的语句。这一节，配合分支程序设计，学习 RANDOMIZE 语句、函数 TIMER 和 RND。同时，还要进一步熟悉学习的 INKEY $ 函数的应用。

TIMER 函数的格式是：

TIMER

它的功能是取开机以来经过的秒数。

它可以用来在程序中计时。例如，要计算某段程序的执行时

间，可以这样做：

（1）在这段程序的开头记录开始时间：

start＝TIMER

（2）在这段程序的最后记录结束时间：

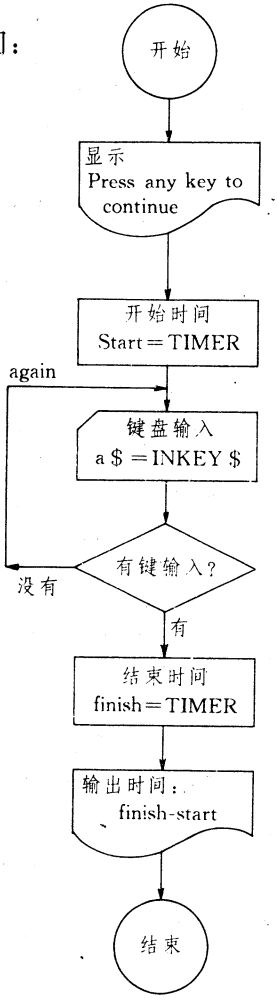
finish＝TIMER

（3）以上两个时间之差，就是该段程序的运行时间：

Finish－start

例 6.4.1 要求用户从键盘输入任意一个字符，用函数 INKEY $ 读取。为了捕获用户输入的字符，要反复调用 INKEY $ 函数。这个例子将打印出从要求输入到用户打入字符所经过的时间。

程序框图如图6.4.1所示。



程序如下：

```
        LPRINT "Press any key to continue."
        start=TIMER
again:
        a$=INKEY$
        IF a$="" THEN GOTO again
        finish=TIMER
        LPRINT "Time=";finish-start
        END
```

下面是这个程序的一次运行结果：

```
Press any key to continue.
Time=7.090088
```

图 6.4.1 例6.4.1流程图

RND 函数能产生0到1之间的单精度随机数，随机数的统计规

律服从均匀分布。RND 函数的格式为：

RND（n）

n＜0时，始终给出一个相同的数。

n＝0时，给出上次产生的那个随机数。

n＞0或省略时（简单表示为 RND），按顺序取下一个随机数。

即使 n＞0，当每次运行程序时，都产生相同的随机数序列。例如，表6.4.1列出前20个随机数，当每次程序运行时，调用函数 RND，它都按这个顺序给出随机数。

表 6.4.1

.7107346	.99058	.8523988	.3503776
.0436358	.0897767	.5111076	.7553547
.935394	.3444337	.2073416	.5360375
.2351914	.7695048	.2087399	.9520086
.9767802	.8262324	.4440745	.2794519

如果想获得新的随机数序列，可使用 RANDOMIZE 语句。这个语句的格式为：

RANDOMIZE［expression］

表达式 expression 省略时，程序运行到该语句时便停止运行，通过显示下列信息，要求输入随机数发生器的初值：

Random Number seed（－32768 to 32767）

当使用自变量 expression 时，BASIC 就使用该值（这个值也叫作种子）初始化随机数发生器。

不同的种子，随机数序列也不同。初始化随机数发生器的一个便利方法，是使用 TIMER 函数：

RANDOMIZE TIMER

每次运行程序时，函数 TIMER 都提供一个新的种子，从而获得一个新的随机数序列。

虽然 RND 函数本身给出的是0到1之间的数，但是，通过下式可得到任意 a 到 b 范围的随机数整数：

INT（（b－a＋1）＊RND＋a）

例如，要想得到0到99的整数随机数，则其表达式为：

INT（100＊RND）

例 6.4.2 编制一个两位正整数以内的加法测验程序，程序运行时，显示出题目，如

35＋26＝

并等待从键盘输入答案，如果答案正确，则显示“Good!”，否则显示“Wrong!”。例如，

35＋26＝61　Good!

接着显示：

Press C to continue.

Press any other key to end.

这时，用户按键C，可以得到一个新的题目。按C以外的任何键，程序结束。

在所要设计的程序中，将要用到本节着重介绍的RANDOMIZE语句，函数TIMER，RND和INKEY$。

图6.4.2是本例题的程序框图。

下面是按照图6.4.2编制的程序。

```
start:
    RANDOMIZE TIMER
    a%=INT（100＊RND）:b%=INT（100＊RND）
    C%=a%+b%
    PRINT a%; “+”; b%; “=”;
    INPUT ; “  ”, answer%
    IF answer%=c% THEN
        PRINT “ Good!”
    ELSE
        PRINT “ Wrong!”
    END IF
```

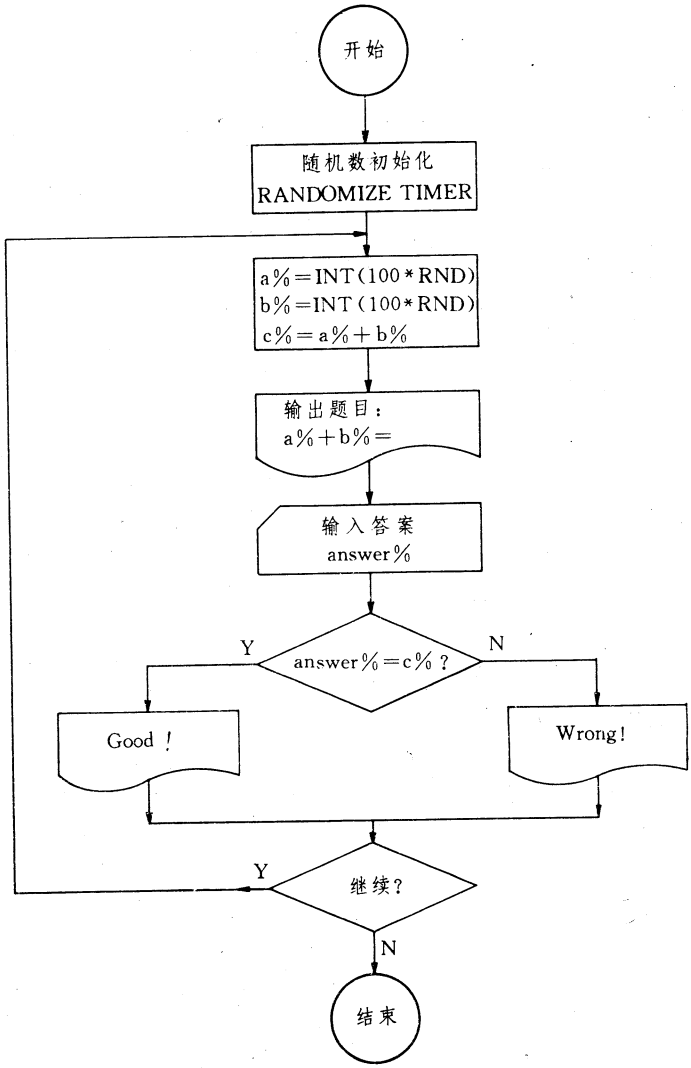


图 6.4.2　例6.4.2程序的流程图

PRINT “Press C to continue.”

PRINR “Press any other key to end.”

```
WaitKey:
    a$=INKEY$
    IF a$=" " THEN GOTO WaitKey
    IF a$="C" OR a$="c" THEN GOTO start
    END
```

小　　结

这一章介绍了分支程序的结构及流程图的画法，讲解了用于构成分支程序的 BASIC 语句，并通过例题讲述了分支程序的设计方法。

在构成分支程序时，主要使用下列两个语句：

(1) 单行格式和块格式的 IF 语句；

(2) SELECT CASE 语句。

在结构化的程序设计中，一般不用 GOTO 语句，或很少使用。

在 IF 语句的使用上，为使程序结构清楚，应尽量使用 IF 语句块。

为了丰富程序设计手段，在本章最后，结合分支程序的学习，我们又介绍了新的语句 RANDOMIZE 和函数 TIMER 及 RND，还进一步介绍了 INKEY$ 函数的应用。

习　　题

6-1 什么是分支程序?用什么语句可以实现分支程序?说明这些语句的格式和功能。

6-2 什么是程序流程图?它有什么作用?

6-3 画出下面程序的流程图，按格式写出程序的输出结果：

```
A=1:B=2:C=3
IF A*B>C THEN
    C=C+1
```

```
ELSE
    B=A+B
END IF
IF A+B>C THEN
    C=C+A
ELSE
    A=A+1
    B=B+1
END IF
END
```

6-4 写出实现下列条件分支的程序段：

(a)仅当 x=5时，函数 Y=sin(x)，否则，Y=cos(x)。

(b)当0<x<5时，y=lnx，否则，y=0。

(c)若 A 和 B 两个逻辑量不同时为真，则打印出 A 和 B 的值，然后程序结束，否则，打印“A and B are true”，然后程序继续执行。

6-5 画出下列程序的流程图，说明程序的功能，写出程序的输出：

```
A=15:B=20
IF A=B THEN
    PRINT "a=b"
ELSE IF a>b THEN
    PRINT "a>b"
ELSE
    PRINT "a<b"
END IF
END
```

6-6 输入两个整数，比较它们的大小，输出比较的结果，试编制程序。

6-7 已知三个数 x，y 和 z，试画出流程图并编写程序，要求

当三个数的平均值大于83时，打印出“A”字；当等于83时，打印出“B”字；当小于83时，打印出“C”字。三个数x，y和z，用INPUT语句输入，分别用IF语句块和SELECT CASE语句编写出两个程序。

6-8 火车托运站用下列公式计算托运费：

$$y=\begin{cases}0.2x & (x\leqslant 20)\\ 0.2\times 20+0.3\ (x-20) & (x>20)\end{cases}$$

其中x是托运物件的重量（公斤），y是托运费（元），试画出程序流程图和编写程序。x用INPUT语句输入，每次计算完一个托运费后，程序询问，是否还要计算，如果还要继续计算，程序要求输入新的x值，否则，程序结束，停止运行。

6-9 已知a_{11}，a_{12}，a_{21}，a_{22}，c_1和c_2，解二元一次方程组：

$$\begin{aligned}a_{11}x+a_{12}y&=c_1\\ a_{21}x+a_{22}y&=c_2\end{aligned}$$

要求用INPUT语句，READ/DATA语句和赋值语句分别设计三个程序。

提示：

$$x=\frac{\begin{vmatrix}c_1 & a_{12}\\ c_2 & a_{22}\end{vmatrix}}{\Delta},\quad y=\frac{\begin{vmatrix}a_{11} & c_1\\ a_{21} & c_2\end{vmatrix}}{\Delta}$$

$$\Delta=\begin{vmatrix}a_{11} & a_{12}\\ a_{21} & a_{22}\end{vmatrix}=a_{11}a_{22}-a_{21}a_{12}$$

若$\Delta=0$，则方程组无解。

6-10 由键盘输入三个数x，y和z，将三个数按由小到大的顺序打印出来。

6-11 参照例6.4.2编写一个含有负数的三位以内的整数加法程序。

6-12 应用INKEY$函数编写一个BASIC程序，使键盘和显示器（或打印机）在一起，模仿打字机的工作。

6-13 向 RC 串联电路充电时，电容上的电压为

$$U=U_0\ (1-e^{-t/RC})$$

式中 U_0 为直流电源电压，R＝100kΩ，C＝1μf，求在不同时间 t 时的 U/U_0。当输入 t 值为负数时，程序要求重新输入 t，当输入的 t 值为0时，程序结束。

6-14 有分段函数

$$y=\begin{cases}0 & 0\leqslant x<2\\ \sin x & 2\leqslant x<4\\ \cos x & 4\leqslant x<6\end{cases}$$

试编写计算函数 y 的 BASIC 程序。

6-15 用 SELEC CASE 语句和 SGN 函数，重新编写例6.2.4的程序。

第七章　循环程序设计

循环程序是应用极为广泛的基本程序结构。循环程序结构的特点是，它的程序段有可能重复地执行多次，这是与简单程序结构和分支程序结构不同之处。

本章主要介绍实现循环程序结构的语句及下列几种循环程序的设计方法：

IF…THEN 循环结构；

FOR…NEXT 循环结构；

WHILE…WEND 循环结构；

DO…LOOP 循环结构。

最后，还介绍了多重循环程序的结构和设计方法。

除循环语句，本章还将涉及另外两个新语句：

CLS 语句和 TIME$ 语句。

第一节　循环程序结构

无论是简单程序还是分支程序，它们有一个共同的特点，就是程序中的每一个语句，最多只能执行一次。但是在很多实际问题中，要求对一个或若干个语句执行多次。

例 7.1.1　计算 1 至 100 自然数的和 S，即求

$$S=1+2+3+\cdots+100$$

如果用简单程序结构，编写计算 S 的程序，是很不方便的，因为这个表达式太长。

解决表达式太长的计算多采用逐个数进行累加的方法。这时语句有如下的形式：

```
A=A+1
S=S+A
```

程序有如下的形式：

```
A=0 : S=0
A=A+1 : S=S+A
A=A+1 : S=S+A
A=A+1 : S=S+A
.....
PRINT  S
```

（A=A+1 : S=S+A 共100行）

这个程序的缺点是要重复地写（输入）100行A=A+1和S=S+A语句。程序长而重复。这种程序结构，对所要解决的问题，显然是不可取的。

最好的程序结构应当是，程序中只写一次A=A+1和S=S+A两个语句，但使它们能重复地执行100次，这样的程序结构简单、清楚，很容易用我们学过的IF语句和GOTO语句来实现。

用IF语句和GOTO语句实现本例题的流程图，如图7.1.1所示。

根据流程图7.1.1不难写出如下的程序：

```
a%=0: s%=0
Counter%=0
100
IF Counter%<100 THEN
    a%=a%+1: s%=s%+a%
    Counter%=Counter%+1
    GOTO 100
ELSE
    PRINT "Sum="; s%
END IF
END
```

现在我们来分析一下流程图 7.1.1 的程序结构特点。

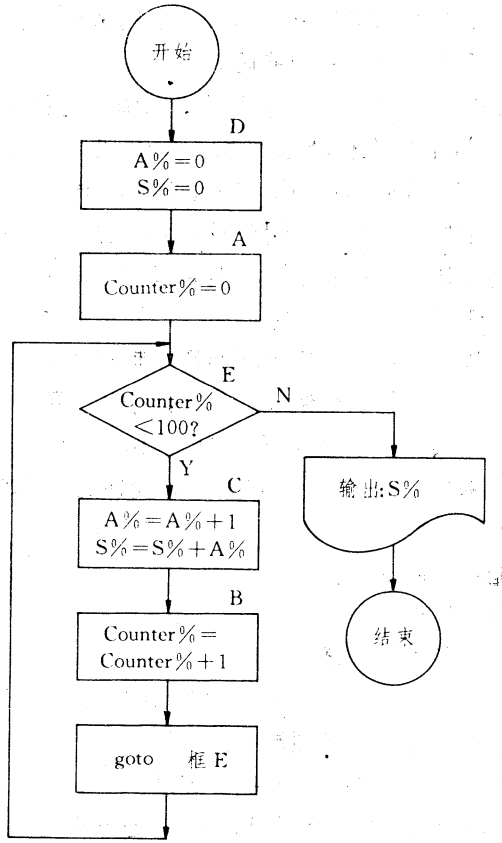


图 7.1.1 例 7.1.1 程序流程图

变量 counter%的作用是控制程序的循环，称为循环控制变量或循环条件变量，它的当前值决定程序继续循环还是终止循环。图 7.1.1 中框 A 的作用是为循环（条件）变量赋初始值，而框 B 的作用则是修改循环（条件）变量的值，对循环条件进行某种运算（本例中为加一），使它的值随着循环的进行而变化。

框 C 所表示的程序段，是需要进行多次重复运算的内容，即计算自然数的累加和，这部分称为循环体。在循环体之外的框 D，

是为了在进行循环运算之前，为循环体内有关的变量，如a%和s%赋初值。由于BASIC在开始运行一个程序时，先把变量置成0，所以，在本例中，框D可以省略。

框E的作用是根据循环变量的当前值，判定循环条件是否成立，从而决定是否继续循环执行程序，所以，框E是循环的控制框。

由以上分析可见，循环结构是这样一种程序结构，它可以重复地执行一个称为循环体的程序段，直到达到给定的次数，或者直到指定的表达式（循环条件）是真（或假）为止。

例7.1.1还表明，一个循环结构，要由置循环（条件）变量初值、循环体、循环条件修改和循环控制四个部分组成。

在这循环程序的四个组成部分中，循环（条件）变量赋初值部分，必须在其他三个组成部分之前，循环体、循环条件计算和循环控制三部分的相对位置，则是可以变换的。例如，在图7.1.1中，它们的顺序是：

（1）循环控制部分；

（2）循环体部分；

（3）循环条件计算部分；

除此之外，还有一些其他可能的循环结构，例如：

循环条件计算→循环控制→循环体；

循环体→循环条件计算→循环控制。

无疑，这两种结构，同样可以用IF语句并配合以GOTO语句来实现。

仍以例7.1.1为例，结构为“循环条件计算→循环控制→循环体”的程序如下：

```
    a%=0: s%=0
    counter%=0
100 counter%=counter%+1
    IF counter%<=100 THEN
    a%=a%+1
```

```
        s%=s%+a%
        GOTO 100
    ELSE
        PRINT "Sum=";s%
    END IF
    END
```

下面是以“循环体→循环条件计算→循环控制”的循环结构编写的例 7.1.1 的程序。

```
    counter%=0
100 a%=a%+1
    s%=s%+a%
    counter%=counter%+1
    IF counter%<100 THEN
        GOTO 100
    ELSE
        PRINT "Sum=";s%
    END IF
    END
```

这个程序有一个特点，就是循环次数是已知的确定值。这种循环程序是通过循环变量（例中的 counter%）来记录循环次数的。另一类循环程序的循环次数是未知的，这种程序中循环变量的值，表示的是循环条件，当满足某种给定的条件时，循环结束。

例 7.1.2 用牛顿迭代法，按给定的精度计算正数 N 的开方。精度要求到小数点后 E 位，计算结果也要求输出到小数点后 E 位。

牛顿迭代法的开方公式是

$$G_{n+1}=\frac{(G_n+\frac{N}{G_n})}{2} \quad n=0,\ 1,\ 2,\ \cdots$$

$$G_0=\frac{N}{2}$$

当 $|\frac{G_{n+1}-G_n}{G_{n+1}}|\leqslant 1\cdot 10^{-E}$ 时，迭代过程结束，这时有

$$\sqrt{N} \approx G_{n+1}$$

迭代多少次才能达到给定的精度，事先不能知道，因此，程序的循环次数是未知的，只能通过检查计算精度是否已达到要求，来确定循环是否结束，程序框图如图 7.1.2 所示。这个流程图采用了“循环体→循环条件计算→循环控制”的循环结构。

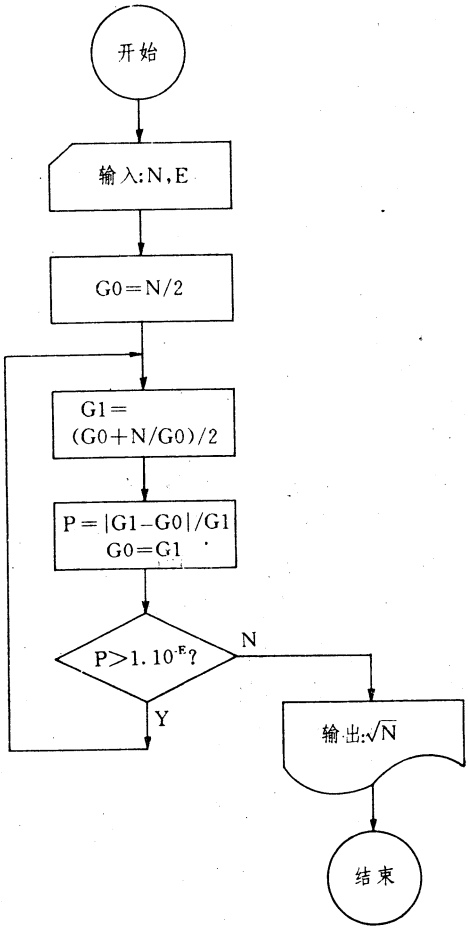


图 7.1.2　例 7.1.2 的程序流程图

下面是按图 7.1.2 编写的 BASIC 程序：

```
    DEFDBL  A－Z
    INPUT "Enter N and E:", N, E
    G0=N/2
There: G1= (G0+N/G0) /2
    P=ABS ( (G1－G0) /G1)
    G0=G1
    IF P<=10^ －E THEN
       G=INT (G1*10^ E+.5) /10^ E
       PRINT "SQR ( "; N;") ="; G
    ELSE
        GOTO There
    END IF
    END
```

令 N=2，E=8，程序将给出如下的结果：

```
Enter N and E; 2     8
SQR (2) =1.41421356
```

以上两种类型的循环程序，都使用了 GOTO 语句，这对结构化的程序设计是不利的。BASIC 语言提供了专门用于设计循环程序的语句，使用这些语句，可以更方便地设计出结构清晰的、合乎结构化程序要求的循环程序。

第二节　循环语句及其应用

一、FOR…NEXT 循环

FOR…NEXT 循环结构，能将一个程序段重复执行给定的次数，因而适用于循环次数为已知的情况。FOR…NEXT 循环包括三个语句：FOR 语句，NEXT 语句和 EXIT FOR 语句。语句格式如下：

FOR counter=start TO end [STEP stepsize]

[statementblock]

[EXIT FOR]

NEXT [counter]

语句中的 counter 是用于循环计数的数值变量，相当于第一节中提到的循环变量。它可以是整型的，单精度的或双精度的。

start 是循环计数的初始值表达式。

end 是计数的终值表达式。

stepsize 是每次循环时，counter 的计数增量（步长）表达式，它可以是正数，也可以是负数。FOR 语句中省略 STEP stepsize 时，步长为 1。

statementblock 是循环体。

语句 EXIT FOR 的功能是退出 FOR…NEXT 循环，继续执行 NEXT 语句后面的语句。EXIT FOR 语句是任选的。

FOR…NEXT 语句的逻辑功能，可用图 7.2.1 所示的流程图来说明。

由图 7.2.1 可以看出，FOR…NEXT 具有“循环控制→循环体→循环条件计算”的循环结构。当执行 FOR 语句时，首先将循环计数的初始值赋给变量 counter，然后，将变量 counter 的当前值与终值 end 比较。在步长为正值的情况下，如果 counter 的当前值大于 end 值，则循环结束。在步长为负值的情况下，如果 counter 的当前值小于 end 值，则循环结束。循环结束后，退出循环，继续执行 NEXT 语句后面的语句。

如果不满足上述循环结束的条件，则执行 FOR 语句和 NEXT 语句之间的各语句，即循环体。如果循环体中含有 EXIT FOR 语句，则执行到该语句时，终止循环，转去执行 NEXT 后面的语句。

执行完循环体的各语句后，执行 NEXT 语句。在这里用步长 stepsize 修改（计算）循环变量 counter 的值，然后控制转向判断循环是否结束。

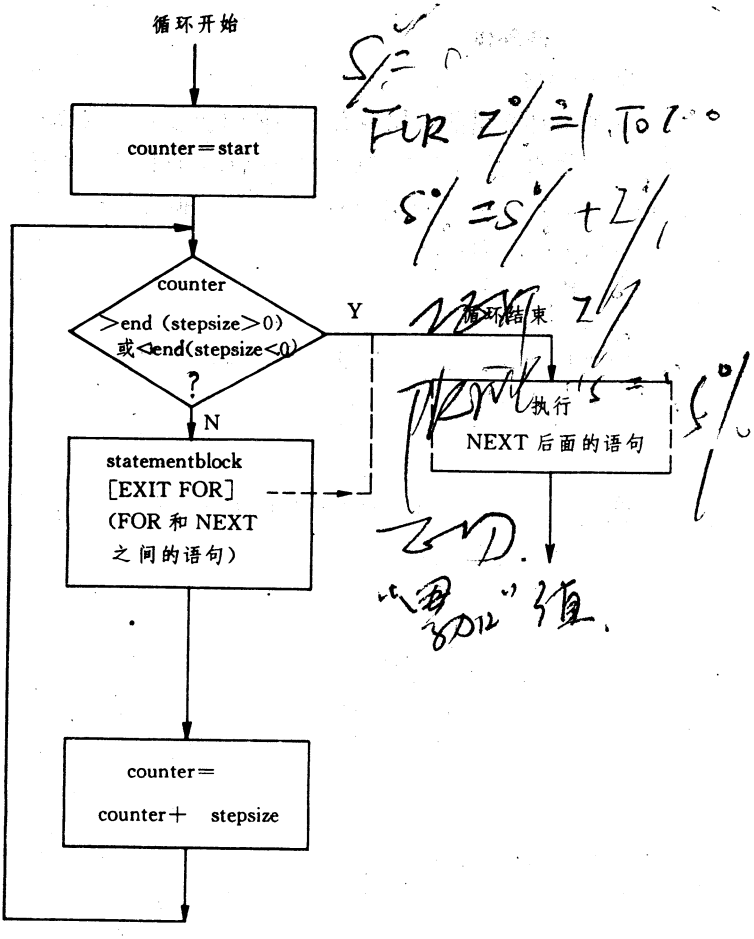


图 7.2.1 FOR…NEXT 语句逻辑功能

例如，下面的 FOR…NEXT 语句，循环执行循环体 10 次：

```
FOR I%＝1 TO   10
      循环体
NEXT I%
FOR I%＝10   TO   1   STEP－1
```

循环体

NEXT I%

如果循环变量的初始值大于终值（正步长）或循环变量的初始值小于终值（负步长），则不执行循环，直接执行 NEXT 后面的语句。例如，下面的 FOR…NEXT 语句，循环执行零次：

```
FOR I=3  TO  2
    PRINT I
NEXT I
```

NEXT 语句中的循环变量可以不写，例如，

```
FOR I=1 TO  10
    PRINT I
NEXT
```

例 7.2.1　用 FOR…NEXT 循环重新编写例 7.1.1 的程序。程序框图如图 7.2.2 所示。

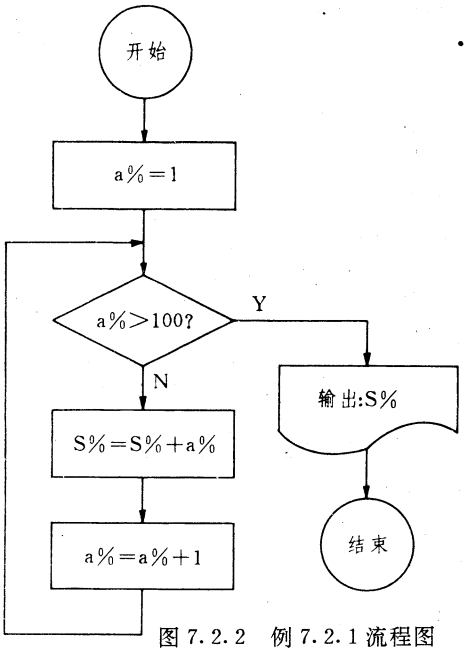


图 7.2.2　例 7.2.1 流程图

程序如下：

```
s%=0
FOR a%=1 TO 100
    s%=s%+a%
NEXT a%
PRINT "Sum=";s%
END
```

在例 7.2.1 程序中，FOR 语句的步长为+1，故语句中省略了"STEP1"。如果给它添上"STEP 2"，则这个程序就能计算 1 至 100 的奇数和。修改后的程序如下：

```
FOR i%=1 TO 100 STEP 2
    s%=s%+i%
NEXT i%
PRINT "OddSum=";s%
END
```

有的循环程序不一定要执行完 FOR 语句确定的循环次数，而提前退出循环，在这种情况下，就需要使用语句 EXIT FOR。

例 7.2.2 要求计算一批数量不定的整数数据的和，数据写在 DATA 语句中，数据结束的标志，是最后一个数为 0，用 FOR…NEXT 循环编写程序。

我们可以编一个循环次数足够大的循环程序，来累加各个数据，当遇到等于 0 的数据时，退出循环。输出数据的数目和这些数据的和。按着这个思路，画出如图 7.2.3 所示的流程图。

下面是例 7.2.2 的程序和运行结果：

```
DATA 1, 2, 3, 4, 5, 6, 7, 8, 9, 0
s%=0: k%=0
FOR i%=1 TO 1000
    READ a%
    IF a%=0 THEN EXIT FOR
```

```
    s%=s%+a%: k%=k%+1
NEXT i%
PRINT "Items=";k%,"Sum=";s%
END
```

Items=9 Sum=45

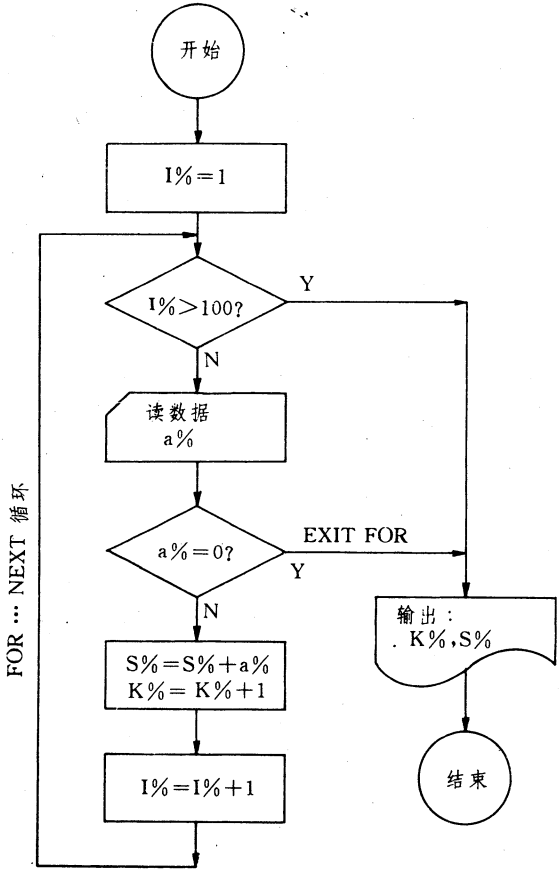


图 7.2.3　例 7.2.2 的流程图

例 7.2.3　编写一个计算 n! 的 BASIC 程序。n 的值由键盘输

入，用长整型数计算。

计算公式：

$$n! = 1 \cdot 2 \cdot 3 \cdot 4 \cdots (n-1) \cdot n$$

这个式子可写成：

$$0! = 1$$

$$i! = (i-1)! \cdot i \qquad i=1, 2, 3, \cdots$$

由上式可见，计算 n! 时，只要对上式计算 n 次（i=1，2，…，n）运算就可以了。

根据上述计算原理，我们可以设计出如下的循环程序，来实现 n! 的计算：

```
DEFLNG A-Z
INPUT "Enter a integer number"; n
a=1
FOR i=1 TO n
    a=a * i
NEXT i
PRINT n; "! ="; a
END
```

令 n=8，运行这个程序，得

8! =40320

例 7.2.4 编写一个程序，计算并输出 0 至 90 度，间隔为 10 度的 sin 值表。

这个题目是以不同的自变量反复计算 sin 的值，重复次数一定（10 次），最适合用 FOR 循环结构实现程序。程序及其运行结果如下：

```
PRINT "x                    sin(x)"
PRINT "-------------------------------------------"
FOR degree%=0 TO 90 STEP 10
    x=3.14159/180 * degree%
```

```
    y=sin(x)
    PRINT USING "##          #.  ######";
degree%, y
  NEXT degree%
  END
```

x	sin(x)
0	0.000000
10	0.173648
20	0.342020
30	0.500000
40	0.642787
50	0.766044
60	0.866025
70	0.939692
80	0.984808
90	1.000000

例 7.2.5 用梯形法则计算定积分

$$I = \int_0^{\pi/2} = \cos(x)\mathrm{d}x$$

首先简要介绍一下计算方法 —— 梯形法则。

设要求计算积分

$$I = \int_a^b f(x)\mathrm{d}x$$

函数 $f(x)$ 的曲线如图 7.2.4 所示，它的积分就是计算图中由曲线 $f(x)$ 和 AabB 所包围的面积。准确地计算这个面积，有时是很困难的，一个近似的计算方法是，像图 7.2.4 所示，将 ab 分成若干个相等的区间，每个区间宽度为 h，然后用梯形面积 aACx_1，x_1CDx_2，x_2DBb 等的和代表所求面积。显然，区间个数越多，或 h 越小，所计算出的梯形面积之和就越接近准确值。这种计算积分的方法，叫作

梯形法则。

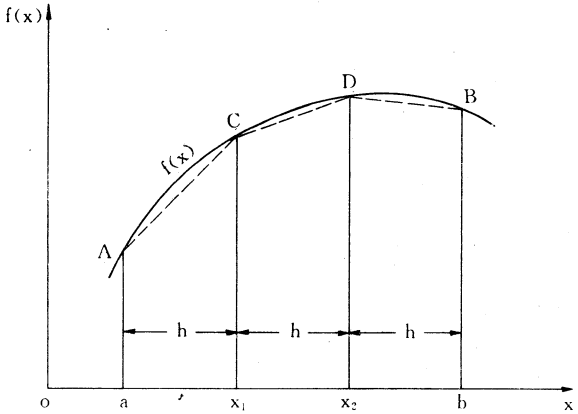


图7.2.4　梯形法则原理

设积分区间（a，b）分为 n 个子区间，则每个区间的宽度

$$h=\frac{b-a}{n}$$

各区间在 x 轴上的分界点

$$x_k=a+kh \quad (k=0,\ 1,\ 2,\ \cdots,\ n)$$

并且有

$$x_0=a$$

$$x_n=b$$

积分的近似值，即 n 个梯形面积之和

$$I_n=\sum_{k=1}^{m}\frac{h}{2}[f(x_{k-1})+f(x_k)]$$

$$=\frac{h}{2}[f(a)+2\sum_{k=1}^{n-1}f(x_k)+f(b)]$$

上式中的 $\sum_{k=1}^{n-1}f(x_k)$ 需要用 FOR 循环来实现。

按上式的算法，便可画出计算本例题的程序流程图，如图7.2.5所示。

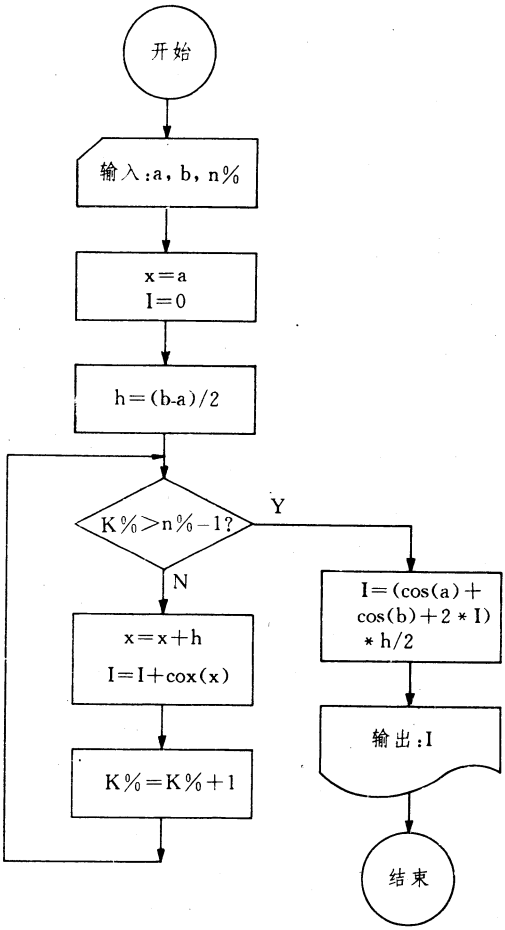


图 7.2.5 例7.2.5的流程图

下面是按照框图写出的程序，给定 a，b 和 n，便可输出积分值。

```
INPUT "Enter a, b and n"; a, b, n%
I=0
x=a
h= (b-a) /n%
```

```
FOR k%=1 TO n%-1
    x=x+h
    I=I+cos(x)
NEXT k%
I=(cos(a)+cos(b)+2*I)*h/2
PRINT "Integration=";I
END
```

下面是a=0，b=π/2=1.570795，n=100时的运行结果（理论值为1）：

```
Enter a, b and n; 0, 1.570795, 100
Integration=.9999789
```

二、WHILE…WEND 循环

当事先知道需要执行循环的确切次数时，用 FOR…NEXT 循环是很有效的。而当不能预先算出执行循环的准确次数，只知道结束循环的条件时，可用 WHILE…WEND 语句方便地实现循环。

WHILE…WEND 语句的格式是：

```
WHILE condition
    [statementblock]
WEND
```

WHILE…WEND 语句的 condition 是执行循环的条件的表达式。当 condition 的值为真（即非零）时，执行 WHILE 与 WEND 之间语句 statementblock，然后再返回到 WHILE，检查 condition，重复上述过程，直到它不为真，则从 WEND 语句后面的语句开始执行。

上述 WHILE…WEND 的循环功能，可表示为图7.2.6所示的逻辑功能图。

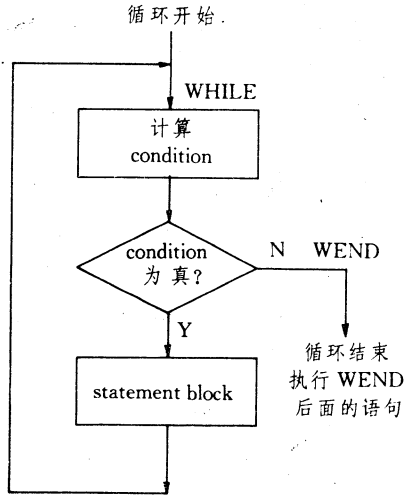


图7.2.6 WHILE…WEND 逻辑功能图

下面是几个应用 WHILE…WEND 循环的例子。

例 7.2.6 给定正整数 N，计算 N!。用 WHILE…WEND 循环语句编写程序。

在例7.2.3中我们曾用 FOR…NEXT 循环实现了计算 N!的程序，在那个程序中，我们是用循环变量 i 控制循环的。当用 WHILE…WEND 循环时，则循环条件应为 “i<=N”。

下面给出程序：

```
DEFDBL A-Z
INPUT "Positive integer number N=", N
F=1
i=1
WHILE i<=N
    F=F*i
    i=i+1
WEND
```

```
PRINT N; "!="; F
END
```

这个程序表明，用 FOR 语句实现的循环程序，也可以用 WHILE 语句来实现。

WHILE…WEND 循环，主要还是用在循环次数未知的情况下，通过判断条件实现循环。

例 7.2.7 用 WHILE…WEND 循环结构，编写例7.1.2的开方程序。

使用 WHILE…WEND 编写循环程序，必须先确定判定循环条件表达式。通过这个例题，我们介绍两种设置循环条件的方法。

第一种方法是，设置一个初始值为1的变量 Flag，当 Flag 的值变为0时，循环结束。变量 Flag 的值由题目所要求的精度来控制，当计算结果达到要求的精度时，将 Flag 置为0，否则，继续进行迭代过程。

下面给出程序：

```
DEFDBL A-Z
INPUT "Enter n and e;", n, e
G0=n/2
Flag=1
WHILE Flag
    G1= (G0+n/G0) /2
    IF ABS ( (G0-G1) /G1) <1/10^ e THEN
        Flag=0
    ELSE
        G0=G1
    END IF
WEND
G1=INT (G1 * 10^ e+.5) /10^ e
PRINT "SQR ( "; n;") ="; G1
```

```
END
```

第二种方法是，以所要求的计算精度作为 WHILE 语句的循环条件。例如，要求精度为10^{-6}，则循环条件可写为

$$\left|\frac{G0-G1}{G0}\right|\geqslant 10^{-6}$$

相应于这种循环条件的程序如下：

```
DEFDBL A-Z
INPUT "Enter n and e;", n, e
G0=n/2
G1= (G0+n/G0) /2
WHILE ABS ( (G0-G1) /G1) >=1/10^ e
    G0=G1
    G1= (G0+n/G0) /2
WEND
G1=INT (G1 * 10^ e+.5) /10^ e
PRINT "SQR ( "; n;") ="; G1
END
```

显然后一程序有更简明的程序结构。

例 7.2.8 利用近似公式

$$e\approx 1+\frac{1}{1!}+\frac{1}{2!}+\frac{1}{3!}+\cdots+\frac{1}{n!}+\cdots$$

计算 e 的值，由键盘输入所要求的计算精度。例如，要求计算到10^{-p}的项时，结束运算并输出到小数点后 p 位，试编制 BASIC 程序。

用 t 表示当前所计算项的值，即

$$t=\frac{1}{n!}$$

则 WHILE 语句的循环条件为

$$t\geqslant 10^{-p}$$

各项 t 的值，在循环体内采用如下的迭代计算方法：

$t_0=1$

$t_n=\frac{t_{n-1}}{n}$　　$(n=1, 2, 3, \cdots)$

用变量 e 累加各项 t 值：

$$e=e+t$$

而且每循环一次，n 值加1。

程序如下：

```
DEFDBL A－Z
INPUT "p="; p
e=1: t=1: n=1
WHILE t>=10^ －p
    t=t/n
    e=e+t
    n=n+1
WEND
PRINT "e="; INT (e*10^ p+.5) *10^ －p
END
```

程序运行举例。当 p=10时，程序输出为

e=2.7182818285

三、DO…LOOP 循环

WHILE…WEND 循环的特点是，把循环条件的测试放在循环的前面，即先测试，后循环；而且只能是测试条件的“真”，即以“真”为循环条件。这两个特点可以从图7.2.6清楚地看到。

与 WHILE…WEND 循环一样，DO…LOOP 也是依据循环条件进行循环，而不是按照给定的循环次数循环。但是，DO…LOOP 循环与 WHILE…WEND 循环不同，DO…LOOP 既能测试循环条件的“真”，也能测试循环条件的“假”，既可以把测试放在循环体之前，也可以把测试放在循环体之后。而且，它还有用于从循环体中退出

循环的 EXIT DO 语句。因此，DO…LOOP 循环是比 WHILE…WEND 灵活得多的一种循环结构。

1. **测试在循环体之前，测试“真”条件的 DO…LOOP 循环**

语句格式如下：

```
DO [WHILE condition]
    [statementblock]
    [EXIT DO]
LOOP
```

语句的逻辑功能示于图7.2.7。比较图7.2.6和图7.2.7，可以看出，这两个语句的功能是一样的。把 WHILE 换成 DO WHILE，WEND 换成 LOOP，那么，例7.2.6——例7.2.8三个例子的程序就改造成 DO…LOOP 循环了。

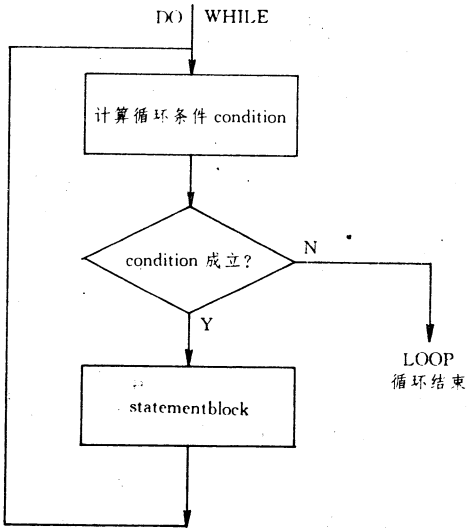


图7.2.7　DO WHILE…LOOP 逻辑功能

例 7.2.9　把例7.2.8计算 e 值的 WHIEL…WEND 循环改为 DO…LOOP 循环。

修改后的程序如下：

```
DEFDBL A－Z
INPUT "p="; p
e=1: t=1: n%=1
DO WHILE t>=10^ －p
    t=t/n%
    e=e+t
    n%=n%+1
LOOP
PRINT "e="; INT (e*10^ p+.5) *10^ －p
END
```

2. 测试在循环体之前，测试“假”条件的 DO…LOOP 语句

与上述第一种情况的唯一区别是，当循环条件表达式为“假”时，执行循环体的语句，否则，退出循环，执行 LOOP 后面的语句。

语句格式如下：

```
DO [UNTIL condition]
    [statementblock]
    [EXIT DO]
LOOP
```

语句的逻辑功能如图 7.2.8所示。

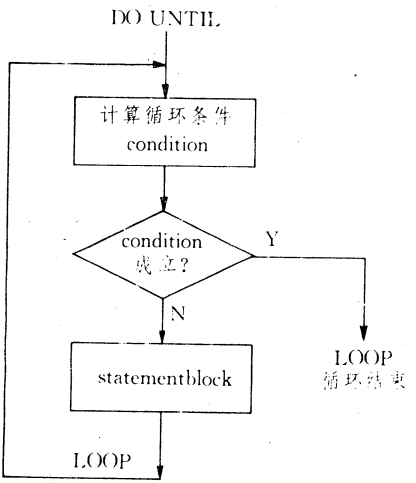


图 7.2.8　DO UNTIL…LOOP 逻辑功能

由语句的以上定义可知，DO UNTIL 的测试条件正好与 DO WHILE 相反。因此，用 DO WHILE 编写的程序，也可以用 DO UNTIL 编写。

例 7.2.10　用 DO UNTIL 改写例7.2.9的程序。

程序如下：

```
DEFDBL A－Z
INPUT “p＝”；p
e＝1：t＝1：n%＝1
DO UNTIL t＜10^ －p
    t＝t/n%
    e＝e＋t
    n%＝n%＋1
LOOP
PRINT “e＝”；INT (e＊10^ p＋.5) ＊10^ －p
END
```

使用 DO WHILE 还是 DO UNTIL 实现循环，在许多情况下，是没有什么区别的，如例7.2.9就是如此。但有时，根据问题的具体情况，用户可能觉得用 DO UNTIL（或 DO WHILE）更方便些。

例 7.2.11 编写一循环程序，用来等待用户从键盘上输入任意一字符。

这是一个没有循环体的空循环。循环条件就是测试用户是否按下了任意一个键。用户所按键的内容，可用函数 INKEY $ 来捕捉。

程序如下：

```
CLS
PRINT “press any key to continue.”
PRINT “Waiting…”
DO UNTIL INKEY $ ＜＞ “  ”
LOOP
```

在这个程序中出现了一个新语句：

```
CLS
```

这个语句的功能是清除屏幕上的文本和图形。

循环程序与函数 TIMER 和语句 TIME $ 配合，常用来构成软

件时延。

我们已经知道，用 TIMER 函数可以取得系统的当前时间。它给出的是从0开始的秒数，并且24小时后，回到0，重新开始计时。系统的当前时间，可用 TIME$ 语句任意地设置。这个语句的格式是：

TIME$ ＝字符串表达式

字符串表达式的格式可以是：

hh	设置小时，分和秒缺省为00；
hh:mm	设置小时和分，秒缺省为00；
hh:mm:ss	设置小时、分和秒。

使用24小时制的时钟。例如，把当前时间设置为下午8点，语句应写成：

TIME$ ＝ “20:00:00”

现在我们来设计时延程序。

例 7.2.12 由键盘输入所要求的时延时间(以秒为单位)，用循环程序实现给定的时延长度。

最简单的方法是，在开始计时时，先把当前时间置为零，程序开始循环。当 TIMER 的值达到给定的延时时间 DelayTime 时，循环结束。用这种方法实现延时的程序如下：

```
INPUT “DelayTime＝”, DelayTime
TIME$ ＝ “00：00：00”
DO UNTIL TIMER＞＝DelayTime
LOOP
```

如果系统时钟还有其他用途，则不能随便用 TIME$ 语句改变系统时钟的计时，这时，就不能用上边的程序设计方法实现时延。

如果当前系统时间加上要求的时延时间，即延时结束时刻，用 LoopFinish 表示，不超过时钟的最大秒数24×60×60，则上述循环程序仍然是正确的。但是，如果延时的开始时间是在系统时钟的0秒之前，延时结束时间是在0秒之后，即跨跃了午夜时刻，则上述

的循环是错误的。这时，应以0秒为界，对0秒前和0秒后两段时间，分别用循环程序延时。

下面是这个延时程序：

```
INPUT "DelayTime", DelayTime
CONST SecondsInDay=24&*60&*60&
' TIMER is a BASIC fuction that returns the number
' of seconds past midnight. The DelayLoop is over
' when TIMER is greater than LoopFinish.
LoopFinish=TIMER+DelayTime
' IF the pause begins befor midnight and lasts
' nutil after midnight:
IF LoopFinish>SecondsInDay THEN
        LoopFinish=LoopFinish-SecondsInDay
        ' Wait until the TIMER has gone past midnight
        ' and is reset to zero:
        DO UNTIL TIMER<=LoopFinish
        LOOP
END IF
' Pause until the TIMER is less than the end time:
DO UNTIL TIMER>=LoopFinish
LOOP
END
```

3. 测试在循环体之后，测试“真”条件的 DO…LOOP 语句

这种循环结构的特点是，循环体至少要执行一次。条件表达式 condition 为“真”时，继续循环。

语句格式为

```
DO
    [statementblock]
    [EXIT DO]
```

LOOP [WHILE condition]

语句功能如图7.2.9所示。逻辑功能图表明，循环开始后，立即执行循环体，然后检查循环条件。条件为“真”时，继续循环，否则，执行 Loop 后面的语句。

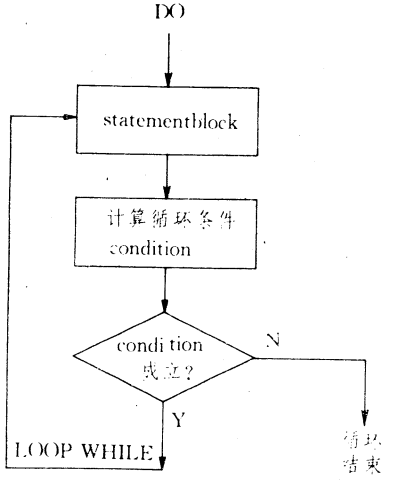


图 7.2.9 DO…LOOP WHILE

例 7.2.13 用 DO…LOOP WHILE 循环实现例7.2.7的开方程序。

```
DEFDBL A－Z
INPUT “Enter n and e:”, n, e
G0=n/2
DO
    G1= (G0+n/G0) /2
    G00=G0
    G0=G1
LOOP WHILE ABS ( (G00－G1) /G1) >10^ －e
G1=INT (G1*10^ e+.5) /10^ e
PRINT “SQR ( “; n;”) =”; G1
END
```

例 7.2.14 编一 BASIC 程序，求解方程2x－1－cos(x)＝0在区间(0,1)内的根。

有多种方法可以求解方程 f(x)＝0的根，本例题使用“二分法”。

二分法适用于求方程 f(x)＝0的实根。

假设方程 f(x)＝0在区间(a,b)内有且仅有一个实根 x^*，我们用其中点 x＝(a＋b)/2将区间分为两个部分，则根 x^* 必在其中一个区间内，如图7.2.10所示。若 f(x)与 f(a)同号，则 x^* 在右半个区间内，否则，x^* 在左半个区间内。于是，我们把根 x^* 所在的区间，减小了一半。

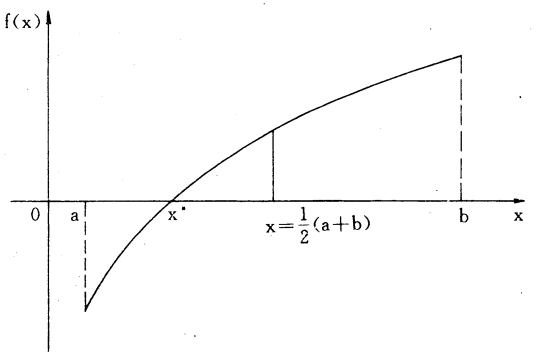


图7.2.10 二分法求方程 f(x)＝0根的原理

设 x^* 在左半个区间 (a, b_1)，这里 b_1 就是中点 x，将区间 (a, b_1) 一分为二，确定出根 x^* 所在的新半个区间，包含 x^* 的区间又缩小了一半。如此反复二分下去，含 x^* 的区间不断缩小，最终必收缩于一点 x^*，这就是所求的根。

但是，在实际计算时，不可能无止境地二分下去，也没有这种必要。在计算机上进行数值计算，总是带有一定的误差，方程的根 x^* 与其近似值 x_i 的误差为

$$\varepsilon=|x^*-x_i|\leqslant\frac{1}{2}\ (b_i-a_i)$$

我们只要计算到给定 ε 值，就可以结束计算。由于 x^* 是个未知数，因此，在实际中，当包含 x^* 的区间足够小时，即满足

$$|x-b_i|<\varepsilon$$

ε 是给定的足够小的数，运算结束。

图7.2.11给出了这个方法的流程图。

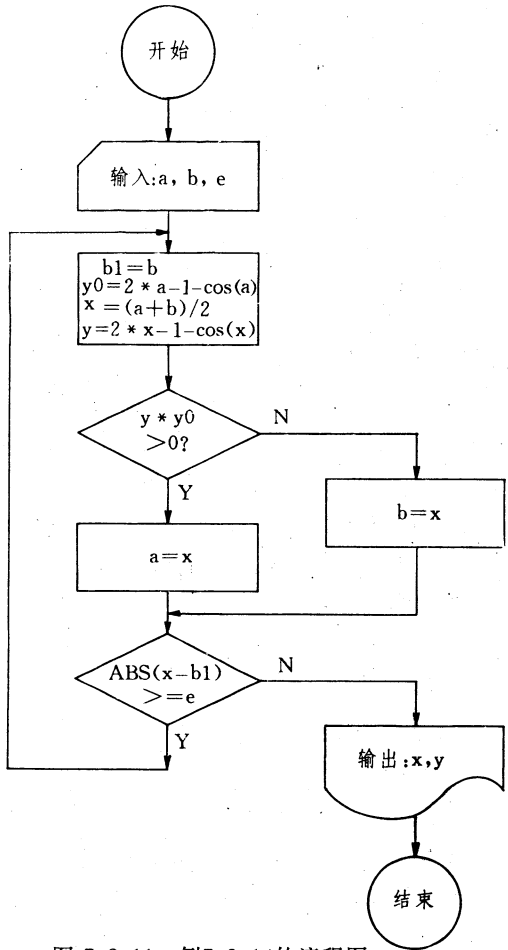


图 7.2.11　例7.2.14的流程图

下面是根据流程图写出的程序。

```
INPUT "a, b and e"; a, b, e
DO
    b1=b
    y0=2*a-1-cos(a)
    x=(a+b)/2
    y=2*x-1-cos(x)
    IF y*y0>0 THEN
        a=x
    ELSE
        b=x
    END IF
LOOP WHILE ABS(x-b1)>=e
PRINT"x=";x,"y=";y
END
```

当输入 a=0，b=1，e=0.0000001时，程序给出如下结果：

x=0.8354296　　y=5.960465E-08

4. 测试在循环体之后，测试“假”条件的 DO…LOOP 语句

这种循环结构的特点是，循环体至少要执行一次。条件表达式 condition 为“假”时，继续循环，否则退出循环。

语句的格式为

```
DO
    [statementblock]
    [EXIT DO]
LOOP [UNTIL condition]
```

语句的逻辑功能，如图7.2.12所示。

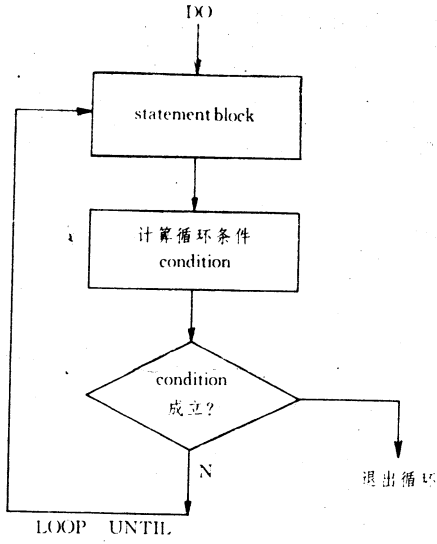


图 7.2.12. DO…LOOP UNTIL 逻辑功能

例 7.2.15 用 DO…LOOP UNTIL 循环结构编写例7.2.2的程序。

程序如下：

```
DATA 1, 2, 3, 4, 5, 6, 7, 8, 9, 0
DO
    READ a%: k%=k%+1
    s%=s%+a%
LOOP UNTIL a%=0
PRINT "Items="; k%-1, "Sum="; s%
END
```

5. 无循环条件的DO…LOOP 循环结构

从上述四种 DO…LOOP 循环语句的格式可知，WHILE condition 和 UNTIL condition 都是任选项。当语句中省略它们时，便构成了无循环条件的 DO…LOOP 循环结构。程序进入循环后，将无休止的循环，除非遇到 EXIT DO 语句。

这种循环结构的语句格式为：

```
DO
    [statementblock]
    [EXIT DO]
LOOP
```

其逻辑功能如图7.2.13所示。

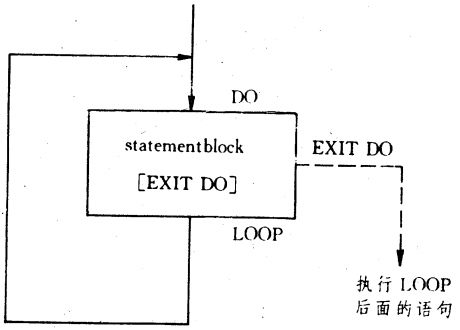


图 7.2.13　DO…LOOP 逻辑功能

例 7.2.16　用不带循环条件的 DO…LOOP 循环结构，重新设计例7.2.7的开方程序。

程序如下：

```
DEFDBL A－Z
INPUT "Enter n and e:", n, e
G0=n/2
DO
    G1= (G0+n/G0) /2
    IF ABS ( (G0－G1) /G1) <1/10^ e THEN
        EXIT DO
    ELSE
        G0=G1
    END IF
```

```
LOOP
G1=INT (G1 * 10 ^ e+.5) /10 ^ e
PRINT "SQR ( "; n;") =";  G1
END
```

例 7.2.17 选票统计问题。有 Zhang-Lin，Li-Ping 和 Wang-Lan 三个候选人，试统计每人的票数。

我们可以把选票写在 DATA 语句中，并用 "end" 表示数据结束。程序的任务就在于反复从 DATA 语句中读选票，统计每位候选人的票数，直读到 end 为止，输出统计结果。我们用 SELECT CASE 语句进行选票统计，用 DO…LOOP 语句进行循环读数据。程序框图如图7.2.14所示。

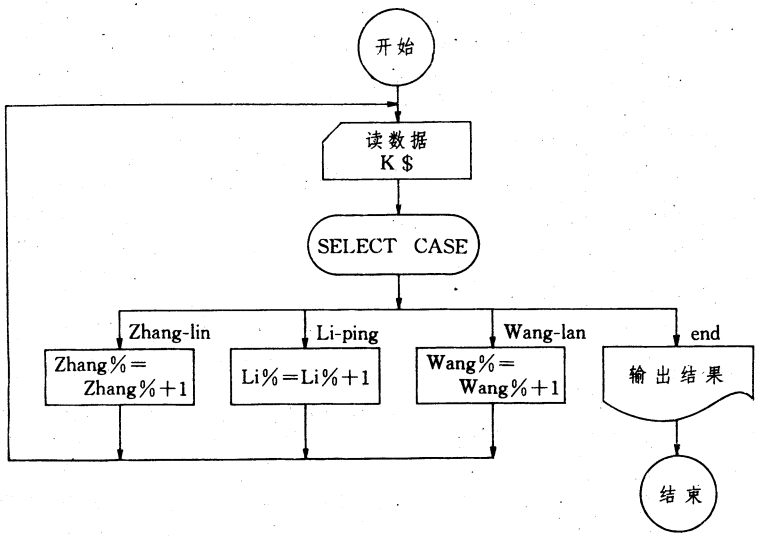


图 7.2.14 例7.2.17流程图

程序和输出如下：

```
DATA Zhang-lin, Li-Ping, Wang-Lan, Li-Ping, Zhang-Lin
DATA Wang-Lan, Li-Ping, Li-Ping, Zhang-Lin, Li-Ping
DATA end
```

```
DO
    READ k $
        SELECT CASE k $
            CASE "Zhang-Lin"
                Zhang% = Zhang% + 1
            CASE "Li-Ping"
                Li% = Li% + 1
            CASE "Wang-Lan"
                Wang% = Wang% + 1
            CASE "end"
                EXIT DO
        END SELECT
LOOP
PRINT "Zhang-Lin…"; Zhang%
PRINT "Li-Ping…"; Li%
PRINT "Wang-Lan…"; Wang%
END
Zhagn-Lin…3
Li-Ping……5
Wang-Lan…2
```

第三节　多重循环程序

如果在一个循环程序的循环体内，还包含有一个或几个循环程序段，形成循环套循环的嵌套程序结构，则把这样的程序叫作多重循环程序。多重循环程序的结构，可以形象地用图7.2.14表示，图中是以 FOR…NEXT 语句为例来说明的，使用其他循环语句也是一样。

图7.2.14的例子表示，循环 A 中包含了 B 循环，在 B 循环中，

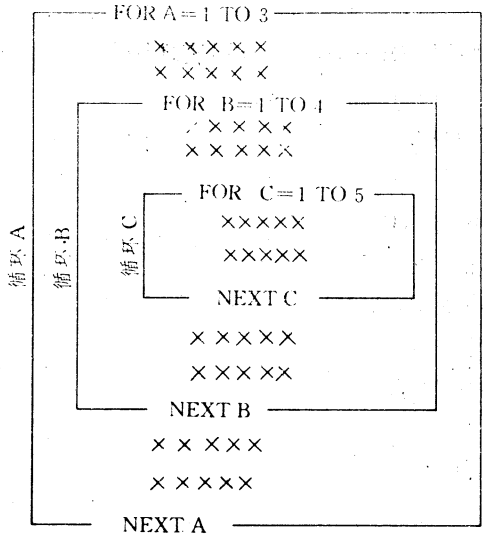


图 7.3.1　多重循环程序的嵌套结构

又包含了C循环。多重循环程序，必须遵守这种嵌套结构。

违反嵌套原则的多重循环程序，是错误的程序，这时，各循环之间会出现如图7.2.15所示那样的交叉。

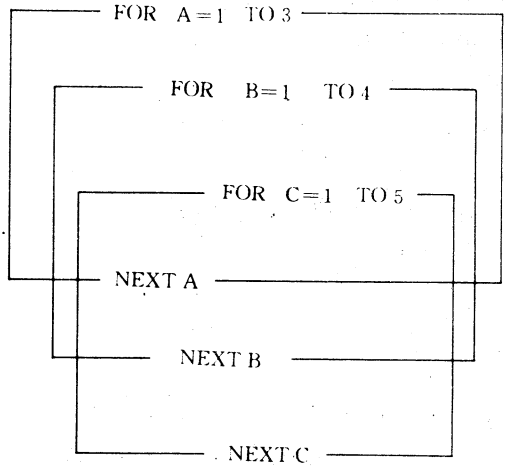


图 7.3.2　错误的多重循环程序结构

现在让我们用 FOR…NEXT 语句实现图7.2.14所示的多重循环程序。按图7.2.14可写出：

```
FOR A=1 TO 3
    ⋮
  FOR B=1 TO 4
      ⋮
    FOR C=1 TO 5
      ⋮
    NEXT C
      ⋮
  NEXT B
    ⋮
NEXT A
```

前面说过，NEXT 后面的变量是可以省略的。因此，上面的程序，又可写成（以下略去用省略号表示的语句）：

```
FOR A=1 TO 3
  FOR B=1 TO 4
    FOR C=1 TO 5
    NEXT
  NEXT
NEXT
```

相邻的若干个 NEXT 语句，可简写成一个 NEXT 语句，例如：

```
FOR A=1 TO 3
FOR B=1 TO 4
FOR C=1 TO 5
NEXT C, B, A
```

所有以上的不同写法，其作用是一样的。

下面的程序段表示的是图7.2.15那样的错误：

```
FOR A=1 TO 3
```

FOR B=1 TO 4

FOR C=1 TO 5

NEXT A，B，C

我们通过一个例子，观察一下多重循环程序的执行过程。

例 7.3.1 观察多重循环程序的执行过程。

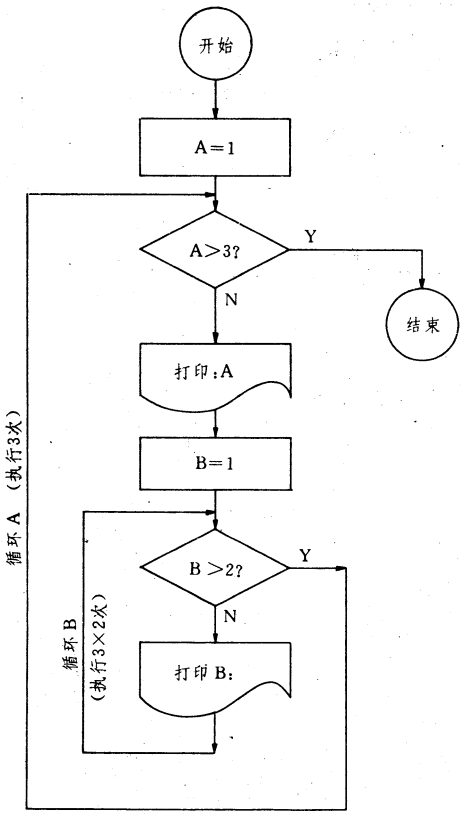


图 7.3.3 例7.3.1的流程图

图7.3.3是一个两重循环的流程图。由图可以看出，在循环A中，嵌套着一个循环B。循环A的循环次数为3，每次循环，打印出循环变量A的值，并进入循环B。循环B的循环次数为2，每次循

环，打印出循环变量B的值。循环B结束后，再次进入循环A，重复上述过程，直到循环A循环3次为止，程序结束。这样，循环A循环3次，循环B循环3×2＝6次。

用FOR…NEXT循环结构实现图7.3.3的程序如下：

```
FOR A=1 TO 3
        LPRINT "A="; A
            FOR B=1 TO 2
                    LPRINT TAB (10); "B="; B
            NEXT
NEXT
END
```

下面是它的输出结果：

```
A=1
                B=1
                B=2
A=2
                B=1
                B=2
A=3
                B=1
                B=2
```

上述输出结果，与我们前面的分析是一致的。

下面举两个多重循环的例子。

例7.3.2 编制一个打印图7.3.4所示三角图形的程序，图形的中心位置，在第14个字符位置上。

图7.3.4是一个由有规律排列的星号组成的三角形图案，适合于用循环程序绘制。本题需要两重循环，外层循环，控制打印的行数和打印星号的起始位置。这个循环的循环次数等于6，因为要求打印6行星号。第一行的打印起始位置是第14个字符位，以后各行

的打印起始位置依次前移一个字符位。如果用变量i%代表打印的行号，则各行的打印起始位置为15－i%。

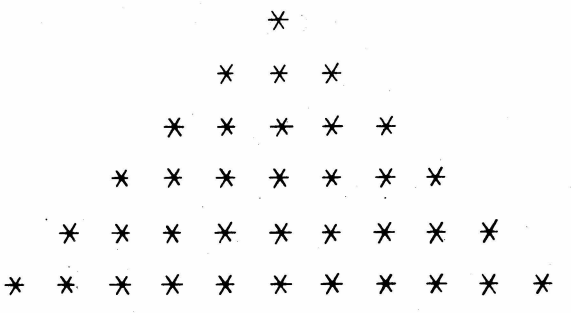


图 7.3.4　例7.3.2题图

内层循环用于打印星号，循环的次数等于要打印的星号数，各行打印的星号数为2i%－1。

根据以上的分析，可写出如下的程序：

```
FOR i%=1 TO 6
    LPRINT TAB (15-i%);
    FOR i%=1 TO 2*i%-1
        LPRINT "*";
    NEXT j%
    LPRINT
NEXT i%
END
```

例 7.3.3　编写一寻找并打印100以内的素数的程序。

验证一个数x是否是素数的简单而又直观的方法是，将它依次除以2，3，4，…，x－1，如果其中哪个数都不能除尽，则这个数x就是素数。

一个数连续地被2，3，4，…，x－1除的过程，可用一个循环程序来实现，而x值要由1变到100，可用另一个循环实现。

可见，解决本例题的程序是一个两重循环程序。外循环提供被

检验的 x，内循环检查 x 是否是素数。程序框图如图7.3.5所示。

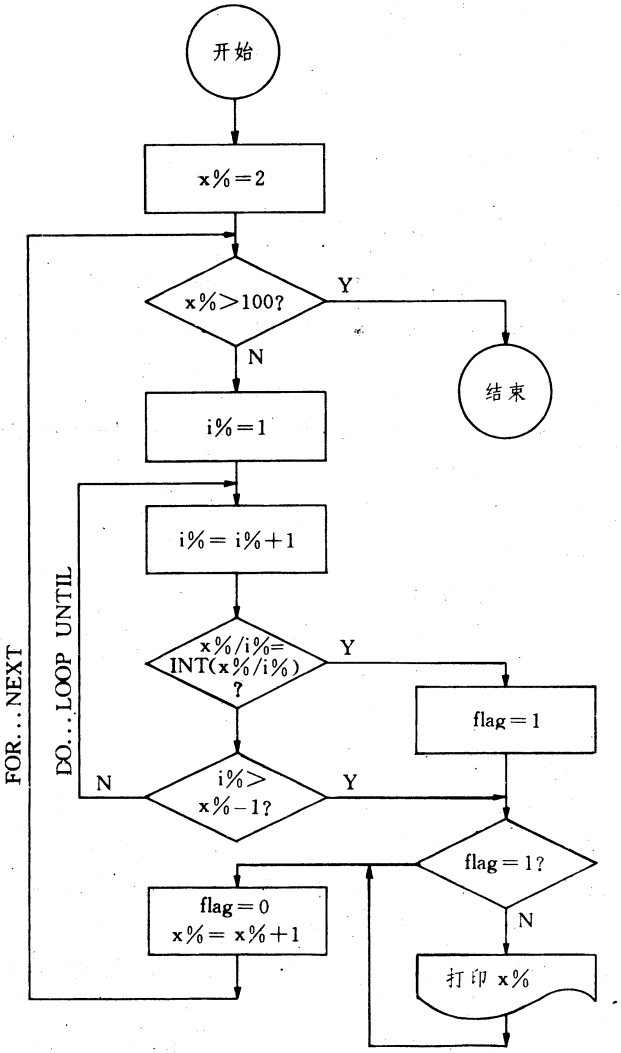


图 7.3.5　例7.3.3的流程图

用 FOR…NEXT 语句实现外循环，用 DO…LOOP UNTIL 实现

内循环。程序和运行结果如下：

```
FOR x%=2 TO 100
    i%=1
    DO
        i%=i%+1
        IF x%  /  i%=INT (x%  /  i%) THEN
            flag=1
            EXIT DO
        END IF
    LOOP UNTIL i%>=x%-1
    IF flag 〈〉 1 THEN
        PRINT x%,
    END IF
    flag=0
NEXT x%
END
```

输出：

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

小　　结

本章中心内容是讲述循环程序的结构，用于构成循环程序的语句和循环程序设计的方法。

首先介绍了循环程序的必要性和循环程序的一般结构。一个循环程序结构，通常要由四个部分组成：

置循环（条件）表达式初值；

循环体；

循环（条件）表达式计算（修改）；

循环控制。

四个部分的相互位置不同时，便组成不同的循环结构。不同的循环结构，其逻辑功能是有所区别的，本章对此作了详细介绍。

结合不同的循环结构，介绍了四种构成循环程序的手段——BASIC 语句。这四种手段是：

IF…THEN 语句；

FOR…NEXT 语句；

WHILE…WEND 语句；

DO…LOOP 语句。

我们既要注意到，用这四种语句构成循环的相同之处，更要注意它们的不同之处。掌握好它们的逻辑结构和语法格式，是正确设计循环程序的关键。

配合 FOR…NEXT 循环结构，有用于退出循环的语句

EXIT FOR

配合 DO…LOOP 循环结构，有用于退出循环的语句

EXIT DO

关于多重循环程序的设计，要特别注意的是多重循环程序的嵌套结构。

除循环语句外，本章结合例题，还介绍了 CLS 语句和 TIME $ 语句的应用。

习　　题

7-1 说明循环程序的基本组成部分。

7-2 说明用于构成循环程序的语句格式、功能和它们之间的区别。

7-3 什么是多重循环程序？它在结构上必须遵守什么原则？

7-4 画出下列程序的框图，按格式写出程序的输出：

(a)
```
FOR X=0 TO 5
    Y=X+5
    PRINT Y
NEXT X
END
```

(b)
```
FOR K=1 TO 14
    PRINT TAB (20-K); "+"
NEXT K
END
```

(c)
```
DO
    INPUT X
    S=S+X: n=n+1
LOOP UNTIL X<0 AND n>10
PRINT "S="; S, "n="; n
END
```

7-5 编写程序，计算自然数1—100之间的偶数和。

7-6 设计一个计算下式的程序：

$$1+\frac{1}{2^2}+\frac{1}{3^2}+\cdots+\frac{1}{10^2}$$

7-7 设计一个计算函数

$$f(x)=7x^2-6x+5$$

当 x=—3，—2，—1，0，1，2，3时的值。

7-8 从下列一组数中，找出最小的数并打印出来：

32，76，—7，10，—12，48

要求使用 DO…LOOP 语句编写程序（附流程图）。

7-9 编写一个打印 $e^{-0.1}\cdot\sin x$ 曲线的程序。x 的取值范围为 0—2π。

7-10 编写一个打印图7.0.1所示图案的程序。

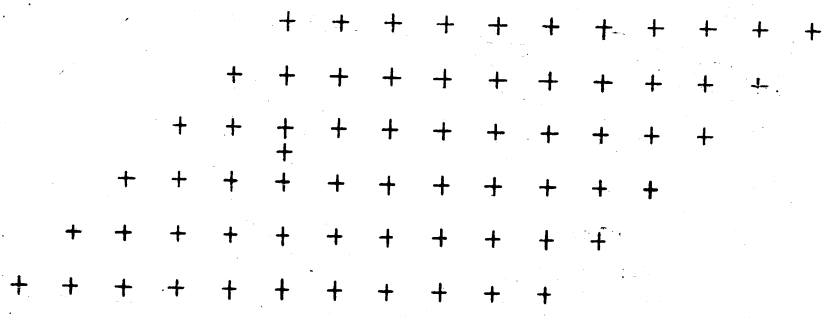


图 7.0.1　习题7-10的题图

7-11 编写程序，计算

$$1+\frac{1}{1\cdot 2}+\frac{1}{2\cdot 3}+\frac{1}{3\cdot 4}+\cdots+\frac{1}{N\ (N+1)}+\cdots$$

直到最后一项的值小于10^{-4}。要求打印出 N 的值，最后一项的值和多项式之和。

7-12 编写计算下式的程序：

$$Y=\sum_{i=1}^{m}\prod_{j=1}^{n}x_{ij}$$

式中 x_{ij}为已知数据，n，m 为已知的常数，这些数据写在 DATA 语句中。

7-13 编写一个计算一系列数之和的通用程序。数据用 INPUT 语句输入，数据个数事先是不知道的。当输入一个特殊的数99999时，表示计算结束，打印计算结果。要求用 DO…LOOP 语句编程。

第八章　下标变量的应用

下标变量是变量的一种类型。在程序设计中，占有重要的地位。下标变量适用于表达有某种关联的数据。使用下标变量，可使程序简洁，给程序的编制，带来很大方便。

本章首先介绍数组与下标变量的概念，数组的定义与再定义，静态数组与动态数组的概念，并讲解了与此有关的语句及与数组有关的两个函数。最后通过一些例子，介绍了下标变量的典型应用。

本章涉及到的语句有：DIM 语句，OPTION BASE 语句，REDIM 语句，ERASE 语句，CLEAR 语句和 SWAP 语句。所介绍的函数是：LBOUND 和 UBOUND。还讲述了两个元命令：$STATIC 和 $DYNAMIC。

第一节　数组与下标变量

一、数组与下标变量的概念

数组是指具有相同变量名的一组数据，数组中的各个单独的数值称为数组元素。在科学计算和事务管理以及数据处理中，常遇到一些相互间有某种关联的数据或数据表格。给这样密切相关的数据一个共同的变量名，对这些数据的运算和处理，会带来很大方便。因此，数组在软件技术中应用很广。

例如，下面的线性方程组：

$$a_{11}x_1+a_{12}x_2+a_{13}x_3=b_1$$

$$a_{21}x_1+a_{22}x_2+a_{23}x_3=b_2$$

$$a_{31}x_1+a_{32}x_2+a_{33}x_3=b_3$$

其中常数 a_{11}，a_{12}，…，a_{33}，b_1，b_2，b_3 都是按一定规律联系在一起的。在数学中，它们常被表示成矩阵或向量的形式，如

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \quad , \quad \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

矩阵 $[a_{ij}]$ 和向量 $[b_i]$，都可以看成是 BASIC 语言中的数组，因而可以给予相应的数组名。如令矩阵 $[a_{ij}]$ 为数组 A，$[b_i]$ 为数组 B。

数组名的命名规则，与简单变量的命名法是完全一样的。

数组元素用数组名和相应的下标来表示，故称为下标变量。例如，数组元素 a_{21} 和 b_3，用下标变量表示，就是

A (2, 1) ， B (3)

变量的下标要用圆括号括起来，各下标间用逗号分隔开，括号内的下标数量叫作数组的维数。数组 A 是二维数组，而 B 则是一维数组。数组的最大维数是 60，每维中的最大元素数是 32768 个。在没有特殊说明的情况下，下标的缺省下限值为 0，上限缺省值为 10，也就是说，数组 A 的下标 i(或 j)最小值为 0，最大值为 10。

下标变量是变量的一种，因此，数组可以具有各简单变量的类型，如整型、长整型、单精度型、双精度型、字符串型和记录型（用户定义的数据类型）。数组类型的说明符和简单变量的说明符一样，如

A% (1, 1) ， B& (3)

也可以用 DEF Type 语句来说明，如

```
DEFINT  A
DEFLNG  B
```

类型说明符总是比 DEF Type 优先起作用，而且，DEF Type 不影响记录类型。记录型数组要用 DIM 语句来说明。

例 8.1.1 设有整型二维数组 A，它有 4×4 个元素。编写程序，将数组的全部元素 A (i, j)，(i=1, 2, 3, 4, j=1, 2, 3,

4）置成 1。

此程序可用双重循环程序实现。程序如下：

```
FOR    I%=1 TO 4
    FOR    J%=1 TO 4
        A% (I%, J%) =1
    NEXT   J%
NEXT   I%
```

对于简单变量，在内存中有相应的空间存储该变量的值，占用内存单元的多少，取决于变量的类型。数组也是如此，它在内存中也占有相应的空间，其占用内存的单元数，除了与类型数据有关，还和数组元素的多少有关。例如，例 8.1.1 中的数组 A%要占用内存的 2×11×11=242 字节，而题目本身实际需要使用 2×4×4=32 字节。数组 A%所占内存空间，可形象地用图 8.1.1 表示。

i \ j	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
1	0	1	1	1	1	0	0	0	0	0	0
2	0	1	1	1	1	0	0	0	0	0	0
3	0	1	1	1	1	0	0	0	0	0	0
4	0	1	1	1	1	0	0	0	0	0	0
5	0	0	0	0	0					0	0
6	0	0	0	0		A%(i, j)=1					0
7	0	0	0	0		(i=1—4					0
8	0	0	0	0		j=1—4)					0
9	0	0	0	0							0
10	0	0	0	0	0	0	0	0	0	0	0

图 8.1.1　数组在内存中的示意图

二、数组的说明

数组在没有说明的情况下，如例 8.1.1 中的数组 A%，下标最小值为 0，最大值为 10。当它在程序中出现时，在编译期间，便开辟了相应大小的存储区。为了有效的利用存储空间，或者需要更大的存储空间，则必须在使用数组之前，对数组加以说明。否则，当数组的下标大于 10 时，由于已超出了所开辟的存储空间而产生错误信息：

Subscript out of range

DIM 语句具有说明数组容量大小的功能，同时也能说明数组的类型。语句的格式是：

DIM Variable（Subscripts）[AS type] [，Variable…

其中 Variable 是数组名；

Subscripts 是数组的下标，可有下列几种格式：

DIM A（8，3）

DIM A（0 TO 8，0 TO 3）

DIM A（−4 TO 10）

一般情况下，下标必须是正数，但使用关键字 TO 时，下标可以是负值（不小于−32768）。上面的第一个和第二个 DIM 语句是完全等效的。

语句中的任选部分 AS type 用来说明数组的类型。type 可以是 INTEGER，LONG，SINGLE，DOUBLE，STRING 和用户定义的记录类型。

例如，例 8.1.1 中的数组说明可写为

DIM A（1 TO 4，1 TO 4）AS INTEGER

这时类型说明符%可以省略。数组这样说明以后，原来多余占用的那些内存空间（见图 8.1.1）就不会出现了。

现在来看一下，如何说明记录型的数组。

假设我们有如下的用户定义类型：

```
Type StudentRec
    StudName      AS STRING * 15
    Mathematics   AS INTEGER
    Physics       AS INTEGER
    Chemistry     AS INTEGER
END Type
```

以上定义的 StudentRec 类型数据，包含四个元素（域）：学生姓名，15 个字节长的定长字符串型变量；数学分数，整型变量；物理分数，整型变量；化学分数，整型变量。

如果一数组 Student 的元素是 StudentRec 类型数据，则数组 Student 为记录型数组。设这个数组共有 1 至 100 个元素，那么，数组应这样说明：

```
DIM Student (1 TO 100) AS StudentRec
```

使用数组中某个记录元素时，它的格式为

变量名．元素名

例如：

```
PRINT Student (i). StudName
PRINT Student (i). Mamthematics
```

在 BASIC 语言中，还有用来专门说明下标的最小值是 1 还是 0 的语句。这个语句的格式为：

```
OPTION BASE n
```

其中 n 或为 0 或为 1。例如：

```
OPTION BASE 1
DIM A (4, 4)
```

则数组 A 的下标范围是 1 到 4，而不是 0 到 4。

数组一旦由 DIM 语句定义后，相应的存储空间便被确定下来。在使用该数组的过程中，如果下标的值超出定义的范围，如上例中，下标若大于 4，便会出现错误信息：

```
Subscript out of range
```

例 8.1.2 有一批关于学生姓名和三门课程（数学、物理和化学）分数的数据，要求把这些数据存入（赋值）记录型数组。设数据写在 DATA 语句中，并以“END”结束数据。数据读入数组后，以表格的形式打印出来。

程序主要由五部分构成：

定义记录数据结构——可使用上面定义的名为 StudentRec 的记录型数据；

定义记录型数组 Student——使用 DIM 语句；

将数据读入数组 Student——循环程序段；

以表格形式输出数据——循环程序段；

数据部分——使用 DATA 语句。

程序清单如下：

```
TYPE StudentRec
    StudName      AS STRING * 13
    Mathematics   AS INTEGER
    Physics       AS INTEGER
    Chemistry     AS INTEGER
END TYPE

OPTION BASE 1
DIM Student (20) AS StudentRec

i%=1
DO
    READ Student (i%). StudName
    IF Student (i%). StudName= "END"
    THEN
        EXIT DO
    ELSE
        READ Student (i%). Mathematics
        READ Student (i%). Physics
```

```
            READ Student (i%). Chemistry
        END IF
        i%=i%+1
    LOOP

    LPRINT "Name", "Mathematics",
    LPRINT "Physics", "Chemistry"
    LPRINT "------------------------";
    LPRINT "------------------------"
    FOR k%=1 TO i%-1
        LPRINT Student (k%). StudName,
        LPRINT Student (k%). Mathematics,
        LPRINT Student (k%). Physics,
        LPRINT Student (k%). Chemistry
    NEXT k%

    DATA Zhang-ping, 50, 60, 70
    DATA Wang-lan, 80, 90, 40
    DATA Li-li, 52, 67, 77
    DATA END

    END
```

程序的输出有如下的形式：

Name	Mathematics	Physics	Chemistry
Zhang-ping	50	60	70
Wang-lan	80	90	40
Li-li	52	67	77

三、静态数组与动态数组

在 Quick BASIC 中，数组分为静态（STATIC）数组和动态（DYNAMIC）数组两种。

静态数组是在编译时开辟存储区的。存储区的大小，由说明语

句决定，在程序中，它的下标范围不能改变。用常数下标在说明语句中定维的数组或隐含值（缺省值）定维的数组，是静态数组。例如，例8.1.1中的数组A%，没有说明语句定义它的维数。它是隐含定义的，所以是静态数组。在例8.1.2中，数组Student是在DIM语句中用常数定义的，因此，它也是静态数组。

动态数组是在程序运行时开辟存储区的。在DIM语句中，用变量下标定义的数组，是动态数组。例如：

```
DIM A (M, N) AS DOUBLE
```

表明，二维数组A是动态数组。

数组还可以使用元命令$STATC说明为静态数组，用元命令$DYNAMIC说明为动态数组。这两个元命令告诉编译器如何给数组分配内存。

元命令的格式是以$开始，放在程序的注释语句中，如

```
REM $DYNAMIC
```

于是，它后面的数组便是动态数组。又如

```
' $STATIC
```

则它后面的数组是静态数组。

如下面程序段

```
REM $DYNAMIC
OPTION BASE 1
DIM a (100, 50)
REM $ STATIC
DIM b (50, 50)
```

说明数组a是动态数组，两个下标的范围分别是1至100和1至50。数组b是静态数组，它的两个下标范围都是1至50。

由于动态数组是在程序运行期间分配内存空间的，因此，在程序中，当不需要某个动态数组时，可将该数组所占用的内存单元收回，从而清除该数组。用于删除动态数组的语句是：

```
ERASE arrayname [, arrayname…]
```

其中 arrayname 是被删除数组的名字。

ERASE 语句对静态数组有不同的作用：它不删除静态数组，而是把数组的全部数值量元素置成0，字符串元素置成空（“ ”），如果是记录数组，则把全部元素置成0。

在用 ERASE 语句删除了某动态数组，释放该数组所占用的内存后，当再次使用这个动态数组时，必须先对该数组重新定义（说明）。除了使用 DIM 语句定义外，还可以使用重新定义语句 REDIM。这个语句的格式为：

```
REDIM Variable (subscripts) [AS type]
        [, Variable (subscripts) [AS type] …
```

与 DIM 语句格式基本相同，不再解释。

要注意的是，REDIM 语句虽然能改变数组下标的范围，但不能改变数组维的个数。例如，程序段

```
REM $DYNAMIC
DIM A (50, 50)
ERASE A
REDIM A (5, 5, 5)
```

是错误的，运行时将给出错误信息：

```
Array already dimensioned.
```

如果在未删除动态数组之前，就用 DIM 语句来重新定义该数组，也会产生“Array already dimensioned”的错误信息。而使用 REDIM 语句来重新定义数组时，可不用 ERASE 语句先进行删除。

也可以用能够清除全部程序变量的语句来代替 ERASE 语句。这个语句的格式是：

```
CLEAR
```

使用此语句时要特别注意，它不仅清除了动态数组，而且把所有数值变量置成0，把字符串变量置成空，把静态数组置成0。

例 8.1.3 编写程序，从一组数据中找出最大的数和最小的数。数据的个数和数据均由键盘输入（用 INPUT 语句）。每处理完

一组数后，程序询问，是否还有数据要继续处理。要求用数组存储输入的数据。

本程序应当采用动态数组，因为数组需要用变量下标来说明，而且每次处理的数据个数可能不同，需要对数组重新定义。

程序流程图如图8.1.2所示。

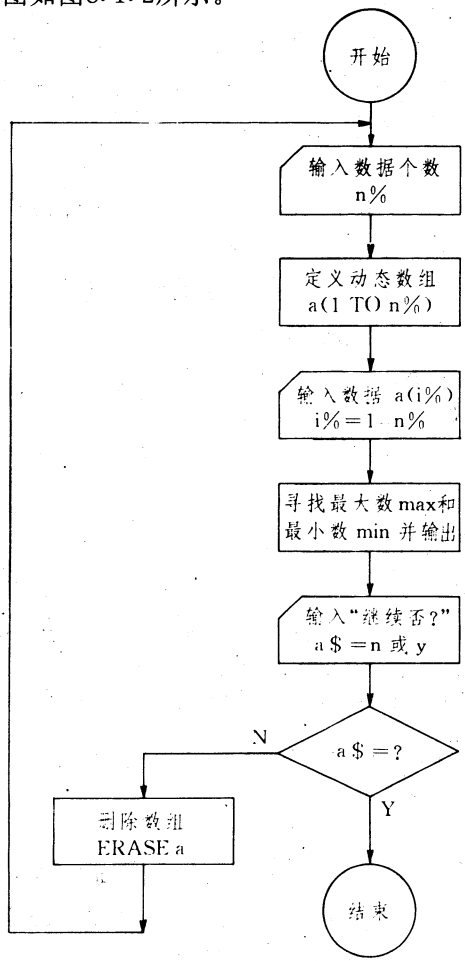


图 8.1.2　例8.1.3的流程图

图8.1.2的流程中，省略了确定一组数中的最大数和最小数的过程。这部分流程示于图8.1.3中。

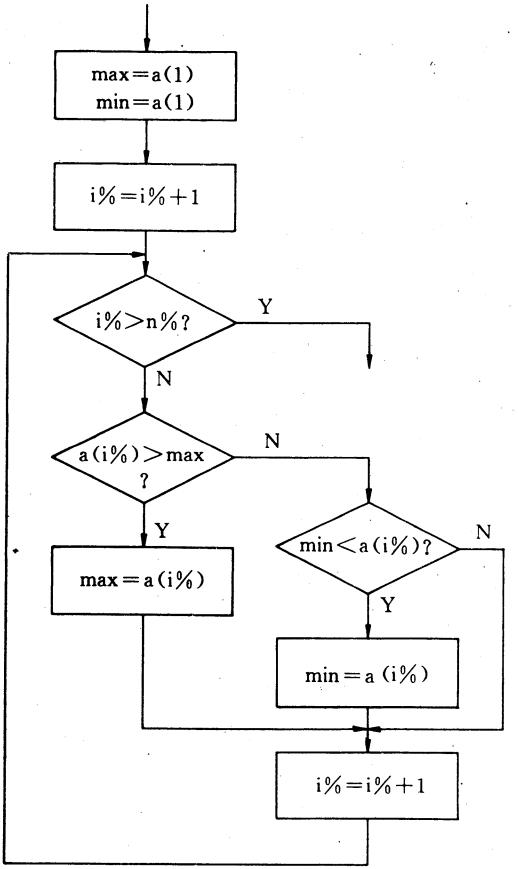


图 8.1.3 确定最大数和最小数

下面给出按图8.1.2和图8.1.3流程所编写的源程序清单：

```
' Find the maximum and minimum of
' Values
DO
```

```
        INPUT "Enter the Number of values", n%
        ' Dimension an array to hold the values.
        REDIM a (1 TO n%)

        ' Get the values.
        FOR i%=1 To n%
            INPUT "Value=  ", a (i%)
        NEXT i%

        ' Find the maximum and minimum values.
        Max=a ( 1)
        Min=a ( 1)
        FOR i%=1 TO n%
            IF a ( i%) >Max THEN Max=a ( i%)
            IF a ( i%) <Min THEN Min=a ( i%)
        NEXT i%

        ' Output
        PRINT "The maxmum is "; Max
        PRINT "The minimum is "; Min

        INPUT "Continue? (y/n) ", a $
        IF a $ = "n" THEN
          EXIT DO
        ELSE
          ERASE a
        END IF
LOOP
END
```

四、LBOUND 和 UBOUND 函数

在 Quick BASIC 中有了两个直接与数组有关的函数。LBOUND 函数可以给出数组下标的下限值，UBOUND 函数可以

给出数组下标的上限值。函数的格式是：

LBOUND (array [, dimension])

UBOUND (array [, dimension])

其中 array 是被查看的数组名；

dimension 是被查看数组的维号。

例如，有下面定义的数组：

DIM A (1 TO 100, 0 TO 50, −3 TO 4)

则函数 LBOUND (A, 1), LBOUND (A, 2), LBOUND (A, 3) 返回的值分别为1, 0, −3。函数 UBOUND (A, 1), UBOUND (A, 2), UBOUND (A, 3) 返回的值，分别为100, 50, 4。

下面是一个应用上述两个函数的简单例子。

例 8.1.4 用随机数函数产生0—2的随机数作为数组 a 下标的下限值，用随机数函数产生3—9的随机数，作为数组 a 的下标的上限值，数组 a 是一维数组。数组 a 被定义后，用0—9的随机数给数组 a 赋值，然后打印出整个数组 a 各元素的值。程序和一次运行结果如下：

```
RANDOMIZE TIMER

Lob%=INT (RND * 3)
Hib%=INT (RND * 7) +3

DIM a (Lob% TO Hib%) AS INTEGER

FOR i%=LBOUND (a, 1) TO UBOUND (a, 1)
    a (i%) =INT (RND * 10)
NEXT i%

PRINT
FOR i%=LBOUND (a, 1) TO UBOUND (a, 1)
    PRINT "a ("; i%; ") ="; a (i%),
NEXT i%
PRINT
```

```
END
```

输出：

```
a (2) =5     a (3) =5     a (4) =9
```

第二节　下标变量应用举例

本节通过一些例子，了解下标变量的应用，掌握下标变量的用法。

例 8.2.1　计算向量$\bar{a}$和向量$\bar{b}$的标量积。已知：$\bar{a}=3i+6j+2k=(a_1, a_2, a_3)=(3, 6, 2)$，$\bar{b}=7i+4j+5k=(b_1, b_2, b_3)=(7, 4, 5)$，计算公式：$c=\bar{a}\times\bar{b}=a_1b_1+a_2b_2+a_3b_3$。

分别用一维数组 a 和一维数组 b 表示向量 a 和向量 b，可得到如下非常简单的程序：

```
DIM a (3), b (3) AS INTEGER
DIM i AS INTEGER

DATA 3, 7, 6, 4, 2, 5

FOR i=1 TO 3
    READ a (i), b (i)
    s=s+a (i) * b (i)
NEXT i

PRINT "c="; s

END
```

例 8.2.2　编制计算多项式 $y=\sum_{i=0}^{n}a_ix^{n-i}$ 的通用程序。系数 a_i，x 和 n 的值用 INPUT 语句由键盘输入。

直接用上式计算，既不方便，运算量也大，且程序不够简洁。比较好的算法是用递推算法，其原理就是把上式变为：

$$y=(\cdots((a_0x+a_1)x+a_2)x+a_3)x+\cdots)+a_n$$

由此可得递推式：

$$y_i = y_{i-1}x + a_i \qquad i=1,\ 2,\ \cdots,\ n$$

$$y_0 = a_0$$

用一维数组 a 的下标变量 a (i) 表示系数 a_i，用循环方法计算上述递推式，便求得 y。

图8.2.1是程序框图。

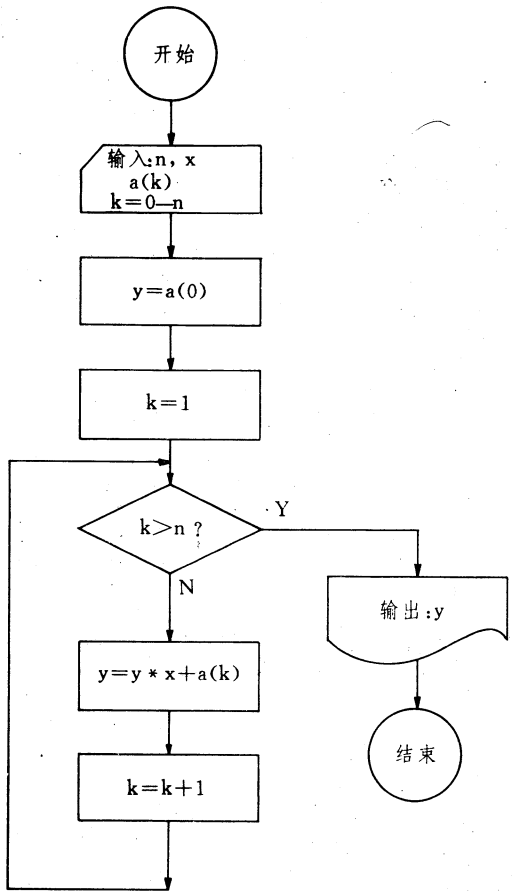


图 8.2.1　例8.2.2流程图

程序如下：

```
REM Polynom
INPUT "Input n and x=  "; n, x
DIM a (n)
DIM k AS INTEGER

PRINT "Enter the coefficients:";
FOR k =0 TO n
    INPUT ; "  ", a (k)
NEXT k
y=a (0)
FOR k=1 TO n
    y=y * x+a (k)
NEXT k
PRINT
PRINT "y="; y
END
```

例 8.2.3 将一组数据按由小到大地顺序排列起来。

这是一类排序问题，在数据处理中，经常遇到。这个问题用下标变量来处理非常方便，先把数据存入数组中，然后对数组的各下标变量值进行比较，通过比较，将较大的数存入下标较大的变量中，将较小的数，存入下标较小的变量中。经过反复比较之后，数据就按变量的下标顺序，由小到大地排列起来。

由上述可知，问题的关键是各下标变量大小的反复比较，必要时，将被比较的两数在数组中的位置交换一下。下面介绍两种反复比较和排序的算法。

算法 I　这种算法的基本作法是：对相邻两数从头到尾地进行比较。若比较结果是后一个数小于前一个数，便将两个数的位置对调。如此反复地比较全部数据，直到没有一对相邻数需要交换位置为止。

图8.2.2详细地描述了这种算法。

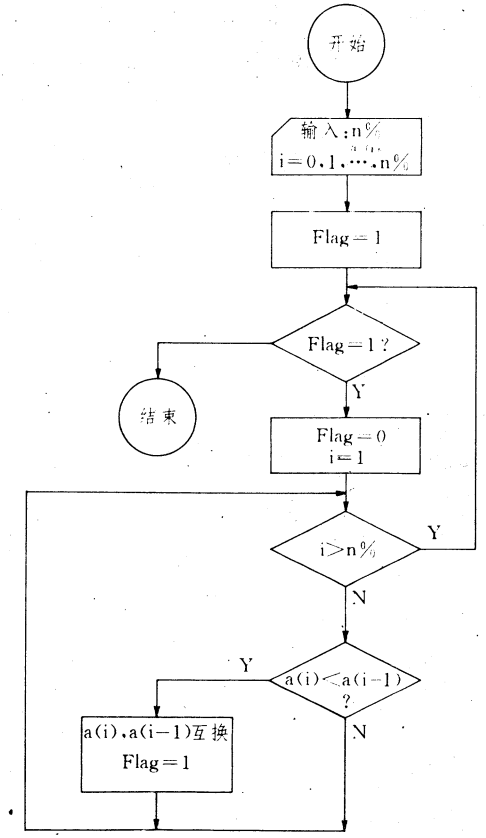


图 8.2.2　例8.2.3算法 I 流程图

两个变量的值互相交换，例如变量 x 的值赋给变量 y，而变量 y 的值赋给变量 x，可以用下面的程序段实现：

```
t=x
x=y
y=t
```

也可以使用一个专门的语句 SWAP 来实现。这个语句的格式是：

```
SWAP variable1, variable2
```

其中 variable1和 variable2是相互交换的两个变量。变量可以是任

何类型，但这两个变量的类型必须相同。

下面是按图8.2.2流程图所示的算法而编写的程序和输出：

```
READ n%

DIM a (1 TO n%) AS SINGLE
DIM i AS INTEGER, Flag AS INTEGER

FOR i=1 TO n%
      READ a (i)
NEXT i

Flag=1
DO WHILE Flag
      Flag=0
      FOR i=2 TO n%
          IF a (i) <a (i-1) THEN
            SWAP a (i), a (i-1)
            Flag=1
          END IF
      NEXT i
LOOP

FOR i=1 TO n%
    PRINT a (i);
NEXT i
DATA 10
DATA 1, 2, 15, 32, 36
DATA 452, 71, 85, 96, 98
END
```

程序的输出如下：

```
1  2  15  32  36  71  85  96  98  452
```

算法Ⅱ　这种算法的基本作法是：依次从数据的前面取数据a (i)，与它后面的所有数据a (i+1)，a (i+2)，…，a (n) 逐个

进行比较，若发现数 a (i) 大于 a (i+k)，就进行 a (i) 与 a (i+k) 的交换。经过这样一轮比较之后，数 a (i) 就一定比它后面的数都小。下一轮再取 a (i+1) 与它后面的数 a (i+2)，a (i+3)，…，a (n) 进行比较、交换。最后一轮是比较 a (n−1) 和 a (n)。经过这样的比较过程之后，数据便在数组 a 中按下标的次序，由小到大地排列起来。

图8.2.3是算法Ⅱ的流程图。

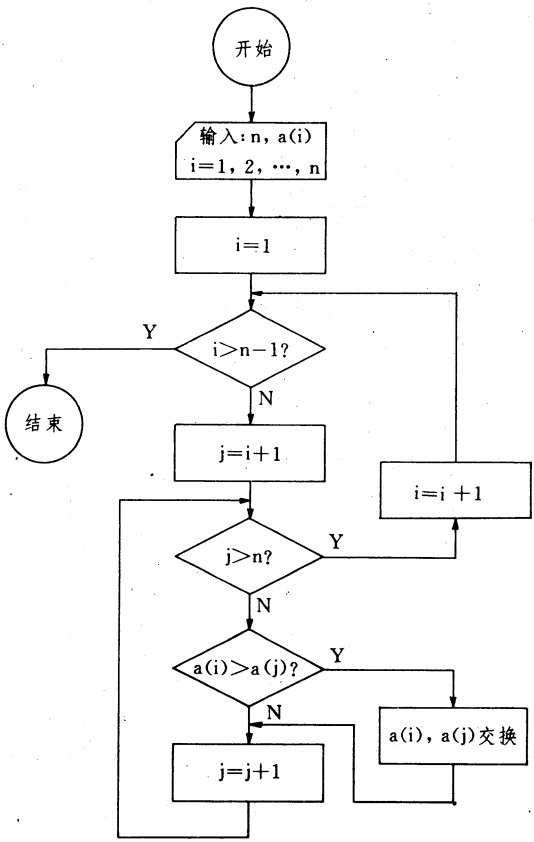


图 8.2.3　例8.2.3算法Ⅱ流程图

这种算法的程序如下：

```
INPUT "Enter number of values", n%

DIM a (1 TO n%) AS INTEGER
DIM i, j AS INTEGER

FOR i=1 TO n%
      PRINT "Value"; i; "=";
      INPUT "  ", a (i)
NEXT i

FOR i=1 TO n%-1
      FOR j=i+1 TO n%
          IF a (i) <a (j) THEN
                SWAP a (j), a (i)
          END IF
      NEXT j
NEXT i

FOR i=1 TO n%
      PRINT a (i);
NEXT i

DND
```

例 8.2.4 用下标变量实现例7.2.17的选票统计程序。

例7.2.17的程序有一个缺点，即当选票数量很大时，输入选票上的人名，是一件很繁重的工作。我们可以改输入姓名为输入编号，这就可以大大减轻劳动，缩短了输入数据的时间。为此，设置一个一维的字符串型的数组，存放候选人的姓名。例如：

```
NameNumber (1) = "Zhang-lin"
NameNumber (2) = "Li-ping"
NameNumber (3) = "Wang-lan"
```

为了方便，可令数组下标号就等于相应候选人的编号。

再设置一个整型一维数组，如名为Vot，记录各候选人所得票数，且数组Vot的下标号等于候选人的编号。这样，Vot（1）中记录1号候选人的票数，Vot（2）记录2号候选人的票数，依此类推，如果读入候选人号为k，则可用下面的语句进行票数统计：

Vot（k）＝Vot（k）＋1

当输入编号k＝0时，表示统计结束，输出结果。

流程图如图8.2.4所示。

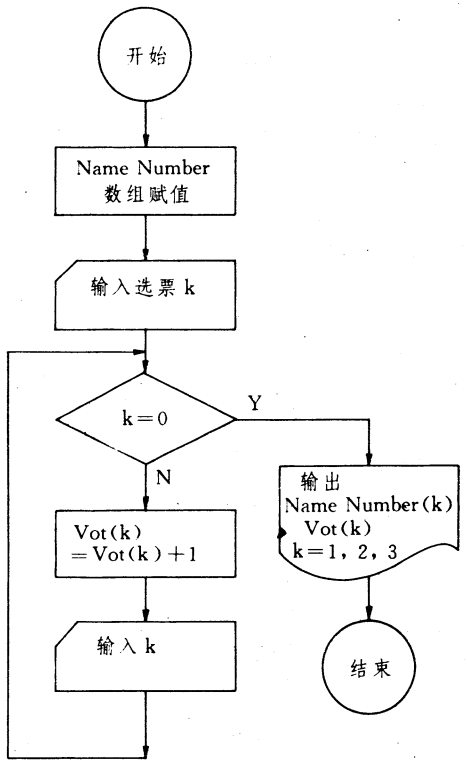


图 8.2.4　例8.2.4流程图

由图8.2.4可以看出，这是一个循环程序。循环体内只有两个语句：一个语句是输入选票(被选人编号)，一个语句是统计票数。循环体与候选人数无关，也就是说，人数的变动，不需要修改循环体，只需修改数组 Vot 的定义和数组 NameNumber 的定义和赋值。这一点也使得本程序优于例7.2.17的程序。

程序清单如下：

```
DIM Vot (3) AS INTEGER
DIM NameNumber (3) AS STRING
DIM k AS INTEGER

NameNumber (1) = "Zhang-lin"
NameNumber (2) = "Li-ping"
NameNumber (3) = "Wang-lan"

INPUT k
DO UNTIL k=0
        Vot (k) =Vot (k) +1
        INPUT k
LOOP

FOR k=1 To 3
      PRINT NameNumber (k), Vot (k)
NEXT k
END
```

我们还可以把上述程序中的两个数组 NameNumber 和 Vot 合在一起，改为记录型数组。为此，首先定义一个记录型变量：

```
TYPE VotRec
    CanName AS STRING * 10
    Number AS INTEGER
END  TYPE
```

然后，把数组 Vot 定义为 VotRec 类型：

```
DIM Vot (3) AS VotRec
```

这样，在 Vot (k). CanName 中存储编号为 k 的姓名，在 Vot (k). Number 中存储编号为 k 者所得的票数。程序如下：

```
TYPE VotRec
  CanName AS STRING * 10
  Number AS INTEGER
END TYPE

DIM Vot (3) AS VotRec
DIM NameNumber (3) AS INTEGER
DIM k AS INTEGER

Vot (1). CanName= "Zhang-lin"
Vot (2). CanName= "Li-ping"
Vot (3). CanName= "Wang-lan"

INPUT k
DO UNTIL k=0
    Vot (k). Number=Vot (k). Number+1
    INPUT k
LOOP

FOR k=1 TO 3
  PRINT Vot (k). CanName, Vot (k). Number
NEXT K

END
```

例 8.2.5 已知矩阵 a 和 b 为：

$$[a]=\begin{bmatrix}3 & 7 & 9 & 10\\ 8 & 9 & 4 & 1\\ 5 & 8 & 6 & 2\end{bmatrix}$$

$$[b]=\begin{bmatrix}15 & 9 & 20 & 36\\ 47 & 32 & 5 & 10\\ 3 & 6 & 22 & 11\end{bmatrix}$$

试编制计算矩阵 a 和矩阵 b 之和的程序。

令 a_{ij}为矩阵 a 的 i 行 j 列元素，b_{ij}为矩阵 b 的 i 行 j 列元素，则上述两矩阵之和 c 矩阵的元素 c_{ij}，可由下式计算：

$$c_{ij}=b_{ij}+a_{ij} \qquad (i=0,\ 1,\ 2;\ j=0,\ 1,\ 2,\ 3)$$

把 a_{ij}和 b_{ij}的值分别输入给数组 a 和数组 b 中，这可以用 READ/DATA 语句实现，或用 INPUT 语句实现。这是一个多重循环程序段。

计算结果 c_{ij}可存入数组中。为计算出数组 c 的所有元素 c_{ij}，也必须用一个多重循环来执行语句。

c (i, j) =a (i, j) +b (i, j)

要求程序的输出具有矩阵的形式。为输出所有的 c_{ij}，也要用一个多重循环结构。

综上所述，整个程序是由三个双重循环结构组成的。程序中用 READ/DATA 语句输入矩阵 a 和矩阵 b 的元素。定义三个数组 a，b，c，它们都是二维的，下标均为0—2和0—3。

程序清单如下：

```
DIM a (0 TO 2, 0 TO 3) AS INTEGER
DIM b (0 TO 2, 0 TO 3) AS INTEGER
DIM c (0 TO 2, 0 TO 3) AS INTEGER
DIM i AS INTEGER, j AS INTEGER

DATA 3, 7, 9, 10
DATA 8, 9, 4, 1
DATA 5, 8, 6, 2
DATA 15, 9, 20, 36
DATA 47, 32, 5, 10
DATA 3, 6, 22, 11
```

```
FOR i=0 TO 2
  FOR j=0 TO 3
      READ a (i, j)
  NEXT j
NEXT i

FOR i=0 TO 2
  FOR j=0 TO 3
      READ b (i, j)
      c (i, j) =a (i, j) +b (i, j)
  NEXT j
NEXT i

FOR i=0 TO 2
  IF i=1 THEN LPRINT " [c] =";
  LPRINT TAB (5); "|";
  FOR j=0 TO 3
      LPRINT TAB (6+5*j); c (i, j);
  NEXT j
  LPRINT "|"
NEXT i

END
```

程序给出如下的运行结果：

```
     |  18  16  29  46  |
[c] =|  55  41   9  11  |
     |   8  14  28  13  |
```

小　结

数组和它的下标变量，在程序设计中占有很重要的地位。它的应用，可以使许多程序设计问题得到简化。下标变量与循环程序相

配合，使程序简洁，结构清楚。

在一般情况下，使用数组时，在程序中先要用数组说明语句定义将要使用的数组。数组说明语句的格式是：

DIM 数组名（下标范围）[AS 类型] [，…

数组名应符合变量名的规则，下标范围为从0到 n 时，可写为“0 TO n”或简单写为“n”。

如果使用语句

OPTION BASE 1

则下标最小值为1，而不是0。

数组分为动态数组和静态数组。静态数组的存储区是在编译时确定的，它的下标范围在程序中不能改变。用常数下标在 DIM 语句中说明的数组或用元命令 $STATIC 说明的数组，都是静态数组。

元命令的格式是，以 $ 开始，并放在注释语句 REM 中。

动态数组的存储区，在程序运行时才分配确定。在 DIM 语句中，用变量说明的数组，或用元命令 $DYNAMIC 说明的数组，都是动态数组。动态数组在程序中，可以用 REDIM 语句重新定义，或称再定义。它的语句格式与 DIM 一样。

动态数组在程序运行中，可以用语句 ERASE 删除，从而收回分配给它的存储空间。

ERASE 语句不能删除静态数组，它只能使数组的所有元素置成0。

本章介绍的另外两个语句是：

CLEAR　置所有变量为0，包括数组；

SWAP 变量1，变量2　使两个变量的值互相交换。

本章介绍的两个与数组有关的函数是：

LBOUND（数组名，维数）——返回指定数组相应维数的下限值；

UBOUND（数组，维数）——返回指定数组相应维数的上限

值。

记录类型数据，在下标变量中经常应用，也就是说，经常会应用到记录类型的数组。对此，本章给予了一定注意。

习　　题

8-1 试说明 DIM 语句的功能和格式。

8-2 按格式写出下列程序的输出：

(a)
```
FOR J=1 TO 3
  C (J) =3*J+1
  PRINT C (J);
NEXT J
END
```

(b)
```
DIM D (3, 3)
FOR M=0 TO 3
    FOR N=1 TO 3
    D (M, N) =M+N
    PRINT "D(";M;",";N;")=";D(M,N)
    NEXT N
NEXT M
END
```

8-3 已知一维数组 A 的元素为1，3，5，7，9。试编写计算该数组所有元素的和、所有元素平均值的程序。

8-4 求数组 B=（32，79，86，30，75，60）中最大元素和它的下标号，画出框图，编出程序。

8-5 给定矩阵

$$A=\begin{bmatrix} 3.75 & 4.21 & 8.36 & -7.2 \\ -3.5 & 6.78 & 9.2 & -1.5 \\ 2.75 & 13.26 & 20.7 & 0.5 \end{bmatrix}$$

求绝对值最大的元素和它所在的行号和列号，画框图和编程序。

8-6 编一程序（包括程序框图）计算矩阵：

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{bmatrix}$$

的各行元素的积和各列元素的和。

8-7 用 RND 函数产生100个随机数，统计这100个随机数中其值在下列10个区间中各有多少个：

[0，0.1)，[0.1，0.2)，[0.2，0.3) … [0.9，1.0)

8-8 已知数组

$$[A] = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \end{bmatrix}$$

编写程序，将数组 A 改为数组

$$[B] = \begin{bmatrix} 1 & 5 \\ 2 & 6 \\ 3 & 7 \\ 4 & 8 \end{bmatrix}$$

要求画出框图。

8-9 求矩阵 A 的转置矩阵 B。已知

$$[A] = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

试画出框图并写出程序（A 矩阵的转置矩阵 B 满足：$a_{ij}=b_{ji}$）。

8-10 修改例8.1.2的程序，使输出的表格含有每个学生的平均分。

第九章　过程、模块和库

在程序设计中，结构化的方法得到广泛的应用。结构化的程序设计方法，就是把一个功能复杂的程序，按其各部分的功能，分成若干个小的逻辑单元，每个小逻辑单元，都是相对独立的程序块。一个完整的程序，便由这些小逻辑单元像积木式的组合起来。由于每个程序块只完成比较简单的任务，且相对独立（与其他部分较少牵连），因此程序的编制和调试都比较容易，程序结构、层次清楚。

本章首先介绍过程的应用，过程是结构化程序设计的最基本的手段，其次介绍多模块程序的构成，最后介绍程序库的使用。灵活运用过程、模块和库，可以设计出质量较高的结构化的程序。

第一节　过程的定义与调用

一、过程的概念

过程是软件技术中应用极为普遍的程序设计手段。

在设计一个比较复杂的程序时，比较好的方法是，按照程序所要实现的功能，将程序分成若干个较小的逻辑功能单元，使其成为相对独立的程序块，必要时，一个程序块可以调用另一个程序块，使其完成相应的功能。这个被调用的程序块称为过程，而调用过程的程序段，相对于过程来说，是主程序。当然，它们的关系是相对的，因为一个过程可以调用另一个过程。一个过程也可以调

用它自己本身，这叫作过程的递归调用。被调用的过程，执行完自己的任务，继续主程序的运行，称为返回主程序。

主程序与过程的调用与返回，遵守嵌套原则，即哪个程序段调用了过程，它还返回那个程序段。图9.1.1表示出主程序与过程之间的调用与返回关系。

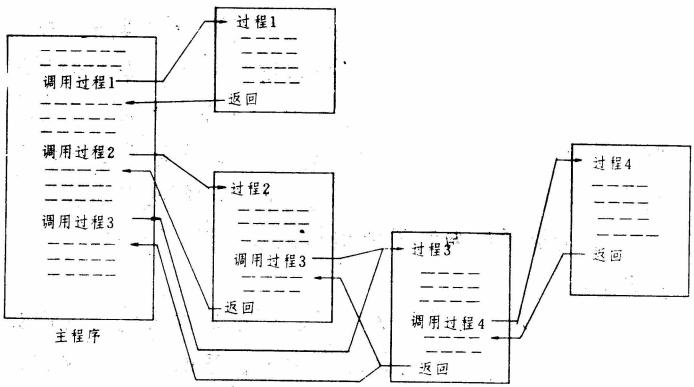


图 9.1.1　过程调用与返回关系示意图

把一些逻辑功能单元，编写成过程，供其他程序使用，这种方法可使复杂程序的逻辑关系明确、结构清晰、易读易懂且便于查错修改。这是一种结构式程序设计方法（或称模块程序设计方法）。过程是结构式程序设计的主要手段，也是节省程序代码和扩展语言能力的主要途径。

实际上，我们已多次使用了过程。所学过的 BASIC 语言函数，就是一些过程。例如，要计算 sinx 的值，程序直接引用 BASIC 函数 SIN（X），就是调用过程 SIN。然而，任何程序设计语言所提供的过程都是数量有限的，但是我们可以自己定义过程，供自己或其他人调用。

下面用一个例子来说明过程的应用。

例 9.1.1　已知正整数 m 和 n，且 m>n，试计算 m 中取 n 的

组合数 C_m^n：

$$C_m^n=\frac{m!}{n!\ (m-n)!}$$

这个题目需要计算 m！，n！和（m－n）！三个数的阶乘，因此，计算 C_m^n 的程序主要是计算三个阶乘。在例7.2.3我们编写过计算任意正整数阶乘的程序，所以，不难写出本例题的程序。程序清单如下：

```
a=1
FOR i=1 TO n
    a=a*i
NEXT i
b=1
FOR i=1 TO m
    b=b*i
NEXT i
C=1
FOR i=1 TO (m-n)
    c=c*i
NEXT i
PRINT b/(a*c)
END
```

这个程序是由结构完全相同、逻辑功能都是计算阶乘的三个程序单元组成。如果我们应用过程的概念，把计算阶乘的逻辑功能单元编写成一个相对独立的过程，那么，计算 C_m^n 的程序只需用不同的数据（n，m 和 m－n）调用三次计算阶乘的过程，便得到同样结果。程序结构如图9.1.2所示。

图9.1.2清楚地表明，应用过程的结果，使程序代码减少，逻辑结构更清楚。这个过程不仅可供本程序调用，而且凡是需要计算阶乘的程序，都可以调用。

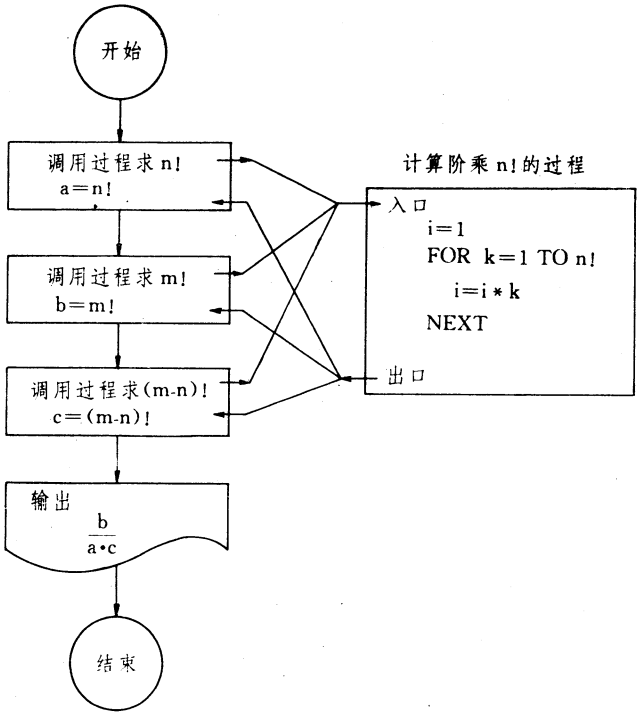


图 9.1.2　例9.1.1的程序结构

在 BASIC 中有两种结构过程：函数过程和子程序过程。下面分别进行讨论。

二、函数过程的定义和调用

1. 函数过程的定义

在编制程序时，用户根据自己的实际需要，可以定义并调用自己编制的函数过程。

所谓定义一个函数过程，就是确定函数过程名、函数过程的参数和函数过程所要实现的功能。函数过程定义语句的格式是：

FUNCTION name [(parameterlist)] [STATIC]

```
    ⋮
    name=expression
    ⋮
    [EXIT FUNCTION]
    ⋮
END FUNCTION
```

其中大写字母为语句的关键字，方括号中的内容是任选项。下面解释一下各项的含义。

FUNCTION　函数过程的开始标志。

name　函数名，它遵循有关变量名的规则，可以使用类型定义符（%，&，!，#或$）。函数名的类型，决定函数返回值的类型。

parameterlist　用逗号分隔的参数表，用来向函数传递自变量的值，是任选项。

STATIC　表示函数的一种属性，是任选项。它的意义和应用以后再作介绍。

expression　函数的返回值，要将它赋给函数名 name。

EXIT FUNCTION　退出函数过程的语句，用户可根据需要选用，以便在执行到 END FUNCTION 之前返回主程序。

END FUNCTION　表示所定义的函数的结束。执行此语句时，返回主程序。

例 9.1.2　定义一个计算 n! 的函数。

令函数名为 Fac#，因而它是一个双精度函数。参数设为整型数 n，即为 n%。根据上面介绍的函数定义语句格式和图9.1.2，便可写出如下的函数定义：

```
FUNCTION Fac# (n%)
    DEFDBL I
    i=1
    FOR k%=1 TO n%
        i=i*k%
```

```
    NEXT k%
    Fac=i
END FUNCTION
```

2. 在 Quick BASIC 环境建立 FUNCTION 过程

过程可在主模块或其他模块中定义和建立，但是，过程只能在调用过程的程序模块中建立。

FUNCTION 过程可按下列步骤建立：

(1) 显示 Edit 的菜单并从中选择 New FUNCTION 命令，这时，屏幕上出现如图9.1.3所示的对话框。

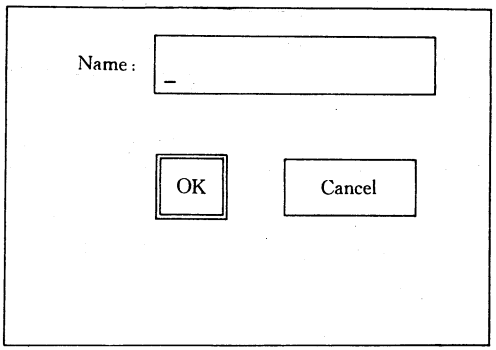


图 9.1.3 建立 FUNCTION 和 SUB 的对话框

(2) 在对话框的 Name 栏内填写所要建立的 FUNCTION 过程名，例如 Fac，然后，按回车键，屏幕上显示如下两条语句：

```
FUNCTION Fac
END SUB
```

(3) 在过程名后面填写过程参数（如果有参数的话）。在 FUNCTION 和 END SUB 两语句之间。输入所要定义的过程的各语句。例如，可以输入例9.1.2定义的 Fac# (n%)。

(4) 返回模块级代码显示

按上述步骤建立 FUNCTION 过程时，当前工作的屏幕（Ac-

tive Window）显示的是过程，如若编辑主程序，则需要将屏幕显示转换到模块级代码的显示。

过程与模块级之间的转换，是通过 View 菜单的 SUBs…命令来实现的。选中这个命令后，出现如图9.1.4所示的对话框。图中假设主模块尚未命名，因而显示〈Untitled〉。用方向键可以选择所要编辑的内容，如模块级〈Untitled〉或过程 Fac＃。

通过图9.1.4的对话框，还可以实现其他一些功能。操作〈Tab〉键，可以选择 Split Window，Move，Delete 和 Cancel。

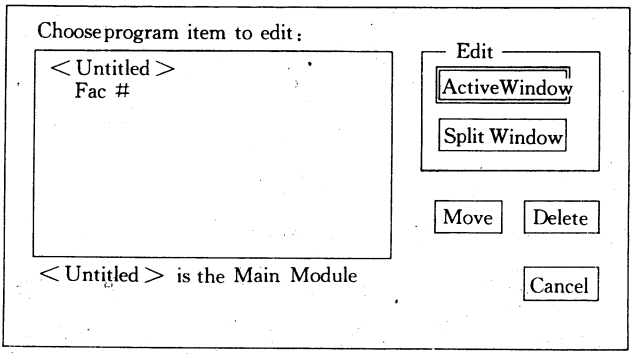


图 9.1.4　View SUBs 对话框

Split Window 命令能使屏幕分为两个窗口，从而可以同时显示两个模块或过程。

Delete 命令用于删除过程。

Move 命令可以将过程从一个模块移至另一个模块。

3. 函数的调用

函数过程一旦被定义，就可以被其他程序调用。调用函数过程和调用 BASIC 内部函数，如 SIN 等，是一样的，即在表达式中直接使用函数名。调用函数时，自变量的类型必须与函数定义中参数表中的相应参数类型一致。例如，函数 Fac＃的参数表中有一个整型参数，那么，在调用 Fac＃函数时，传送给它的自变量也应当是

整型的。函数是把返回值送到它的函数名中，因此，函数名的类型是和它返回值的类型相匹配的。

当函数的参数表中，含有一个以上的参数时，调用该函数时的自变量，必须按照参数表中参数的顺序和类型，一一对应的给出。图9.1.5表示出参数与自变量的这种对应关系。自变量就是按这种对应关系传送到过程中去的。

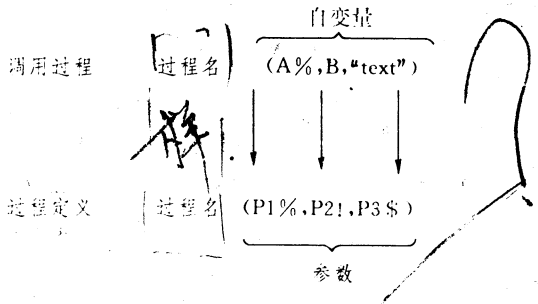


图 9.1.5 参数与自变量

函数被调用后，从 FUNCTION 后面的第一个语句开始执行，直到 END FUNCTION，便退出函数过程，返回调用它的主程序，从被中断的地方继续执行。遇 EXIT FUNCTION 语句时，也能返回主程序。

例 9.1.3 利用在例9.1.2中定义的函数 Fac#，编写计算例9.1.1中 C_m^n 的主程序。m 和 n 为已知数，用键盘输入语句 INPUT 输入。程序输出中用符号 C（m，n）表示 m 中取 n 的组合。

按照上述调用函数的规则，写出主程序如下：

```
DEFINT M—N
INPUT "Enter integers m and n", m, n
PRINT "C ("; m; ","; n; ") ="
PRINT Fac#(m)/(Fac#(n) * Fac#(m—n))
END
```

当输入 m=10，n=2时，程序的输出为：

C (10, 2) =45

三、子程序过程的定义和调用

作为一种过程结构，和函数过程一样，在使用子程序过程之前，也必须先定义子程序过程，以确定子程序过程的名称、参数和功能（程序代码）。定义子程序过程语句的格式如下：

```
SUB name [ (parameterlist)] [STATIC]
    ⋮
[EXIT SUB]
    ⋮
END SUB
```

现对语句中各项内容说明如下：

SUB　子程序过程定义开始标志；

END SUB　子程序过程结束标志；

name　子程序过程的名字。它可以是最多为40个字符长的变量名；

parameterlist　用逗号隔开的参数表，用来向子程序过程传送自变量的值和子程序过程返回主程序的值。

EXIT SUB　退出子程序过程返回主程序语句。当需要执行到 END SUB 之前返回主程序时，使用此语句。

STATIC　子程序过程的属性，以后介绍。

例 9.1.4　定义一个计算 n!的子程序过程。

令子程序过程的名字为 Factorial。设置两个参数：整型数 n 代表所要计算的阶乘，长整型数 x 代表计算的结果。于是有如下定义的子程序过程：

```
SUB Factorial (n%, x&)
    x&=1
```

```
    FOR k%=1 TO n%
        x&=x&*k%
    NEXT k%
END SUB
```

在 Quick BASIC 环境建立 SUB 过程的操作与建立 FUNCTION 过程的操作基本一样，不同之处仅仅是调用 Edit 菜单 New SUB 命令，而不是调用 New FUNCTION 命令，这时，屏幕显示图9.1.3所示的对话框，输入 SUB 过程的名字，例如，Factorial，便进入过程的建立。屏幕显示：

```
SUB Factorial _
END SUB
```

为建立 SUB 过程作好了准备。

同样，调用 View 菜单的 SUBs 命令，可以在 SUB 过程与模块级代码之间进行转换。

现在来谈子程序过程的调用。与函数过程不同，子程序是用语句来调用的。调用子程序语句的格式为：

CALL name [(argumentlist)]

name　被调用的子程序名；

argumentlist　自变量表，表中用逗号隔开用以传送到子程序过程去的变量和常数，也包括子程序过程送回主程序计算结果的变量。和函数过程一样，子程序过程的调用，也要求参数和自变量保持相同的顺序和相同的类型，因为只有这样才能保证在子程序过程和主程序之间正确地传送数据。

例如，调用例9.1.4的子程序过程，计算 m 的阶乘，计算结果保存在变量 b 中。调用语句应写为：

CALL Factorial (m%, b&)

CALL　语句的另一种格式，是省略关键字 CALL，这时自变量表的圆括号也必须略去。例如，上面的 CALL 语句可写为

Factorial m%, b&

其效果是一样的。

例 9.1.5 利用在例9.1.4中定义的子程序过程，编写计算 m 中取 n 的组合数 C_m^n 的程序。对输入和输出的要求与例9.1.3相同。

主程序和 SUB 过程如下：

```
DEFLNG A－C
DEFINT M－N
INPUT "Enter integers m and n", m, n
CALL Factorial (m, b)
CALL Factorial (n, a)
CALL Factorial (m－n, c)
PRINT "C ("; m; ","; n; ") ="
PRINT b/ (a * c)
END
SUB Factorial (n%, x&)
    x&＝1
    FOR k%＝1 TO n%
        x&＝x& * k%
    NEXT k%
END SUB
```

最后还要指出，所有 BASIC 表达式和语句都允许出现在过程体内，即 FUNCTION…ENDFUNCTION 之间和 SUB…END SUB 之间，但下列语句除外：

```
COMMON
DECLARE
DIM SHARED
OPTION BASE
TYPE…END TYPE
FUNCTION…END FUNCTION
SUB…END SUB
```

第二节　传送自变量到过程

在上一节我们已经说过，传送自变量到过程，是按调用的自变量表与过程定义的参数表之间的对应关系进行的。在过程定义中，参数是自变量的位置标志符号。当调用过程时，第一个参数接受第一个自变量，第二个参数接受第二个自变量，等等。过程也可以通过这种对应关系，把数据送给自变量。本节详细介绍有关向过程传送数据的问题。

在 Quick BASIC 环境下编写程序，当程序存入磁盘时，在调用过程的模块级的前面，应为每个过程插入一条 DECLARE 语句。例如，对例9.1.3的程序，插入了语句：

DECLARE FUNCTION Fac# (n%)

对例9.1.5的程序，插入了语句：

DECLARE SUB Factorial (n%, x&)

每条语句 DECLARE 包含有关键字 FUNCTION 过程和 SUB 过程名和括号内的参数表。如果过程没有参数，则括号内是空的。这个参数表具有与过程定义的参数表一样的格式。DECLARE 语句的主要功能是说明对过程的引用和对自变量数目及类型的检查。由于在 Quick BASIC 环境下编程，参数类型检查和过程引用不必用户去考虑，能自动生成 DECLARE 语句，因此，仅当在 Quick BASIC 环境外编程时，才需要考虑上述问题。这里就不详细介绍了。

一、传送常数和表达式

常数，包括字符串常数和数值常数，都可以出现在调用过程的自变量表中。当然，字符串常数必须传送给字符串参数，数值常数必须传送给数值参数。

如果自变量表中的数值常数与过程的相应参数不是同一类型，那么常数被强制为与参数一样的类型。这可以通过后面的例

9.2.1的输出看出。

由常数和变量运算组合成的表达式，也可以传送给过程，并且与常数传送的情形一样，即与参数不是同类型的数字表达式时，可以被强制为参数的类型。

例9.1.3中的 Fac＃（m－n）和例9.1.5中的 CALL Factorial（m－n，c），都是将表达式（m－n）的值传送给相应的过程。

例 9.2.1 子程序过程 test 有两个整型参数 x％和 y％。其功能是将传送给它的参数的两个量打印（显示）出来。主程序用两个单精度数4.1和4.6调用这个子程序过程，还要用两个单精度变量 m 和 n 的和（m＋n）和差（m－n）调用这个子程序过程。试编写主程序和子程序过程，观察程序的运行结果。

程序如下：

```
DECLARE SUB test (x%, y%)
CALL test (4.6, 4.1)
m=5.5
n=2.3
CALL test (m+n, m-n)
END
SUB test (x%, y%)
    PRINT x%, y%
END SUB
```

程序给出如下的运行结果：

```
 5          4
 8          3
```

请读者来分析一下，为什么程序给出这样的结果。

二、传送变量

变量的传送有两种方式：通过引用传送和通过数值传送。

1. 通过引用传送变量

变量，无论是简单变量、数组或记录，都可通过引用传送给FUNCTION过程或SUB过程。

什么是通过引用传送变量?我们知道，任何一个程序变量，在内存中都有一个相对应的地址，在那里保存着该变量的值。通过引用传送变量，就是向过程传送变量的地址，这就是说，变量的地址和过程中相应参数的地址，是同一个地址。

本章第一节中的程序举例中，下列过程调用，都是通过引用传送自变量的：

Fac# (m), Fac# (n)

CALL Factorial (m, b)

CALL Factorial (n, a)

在通过引用传送自变量到过程的情况下，如果过程中修改了参数值，它同时也修改了传送给它的变量值。

例 9.2.2 用随机函数产生20个随机数并存入数组 Array 中。然后调用一排序子程序过程 Sort，将随机数按由小到大的顺序存储在同一数组中，由主程序将排序结果打印出来。

排序子程序过程 Sort 可以用上一章例8.2.3所介绍的方法实现。本例用其第二种算法，由该算法可知，作为子程序过程，它应有两个参数：一个单精度型数组和一个整型数。前者对应于存储随机数的数组，后者对应于随机数的个数，因此，Sort 子程序过程的参数表有如下的形式：

SUB SORT (A ( ), n%)

而调用语句的自变量表，应有如下的形式：

CALL SORT (Array ( ), 20%)

这里 Array ( ) 是通过参考传送给过程的。

注意，数组在自变量表和参数表中的表示方法是，数组名后跟左、右圆括号。另外，数组 A 是单精度型的，因此，在参数表中，它的类型说明可以省略。

下面是本例题的程序清单。

```
DECLARE SUB Sort (a! (  ), n%)
DIM Array (1 TO 20)
FOR i=1 TO 20
      Array (i) =RND
NEXT i
Sort Array (  ), 20
FOR i=1 TO 20
    PRINT Array (i)
NEXT i
END
SUB Sort (a (  ), n%)
    FOR i=1 TO n%-1
        FOR j=i+1 TO n%
            IF a(i)>a(j) THEN
                SWAP a(i),a(j)
            END IF
        NEXT j
    NEXT i
END SUB
```

2. 通过数值传送自变量

通过数值传送变量，是传送变量的值而不是地址。它是通过把变量复制到一个暂存单元，然后传送这个暂存单元的地址来实现数值传送的。由于过程访问的是暂存单元的地址，而不是原来变量的地址，因而过程也就不能改变原来变量的值。

按数值传送变量的表示方法是，将自变量表中的相应变量，用圆括号括起来。

例 9.2.3 编写一个求自然数 N 的质因子的子程序过程。

算法：和手算方法一样，首先用2去除N。若能除尽，则2就是一个因子。继续用2去除上次运算所得之商，若除不尽，将除数加1，即用3再去除上次运算所得之商，若除尽，这除数3是所求因子。再重复上述过程，一直到除数的平方大于被除数（上次运算的商），运算结束。

例如，求N＝126的质因子，其过程为

$$
\begin{array}{r|l}
2 & 126 \\
\hline
3 & 63 \\
\hline
3 & 21 \\
\hline
 & 7
\end{array}
$$

得126的质因子为2，3，3和7。

上述算法的流程图如图9.2.1所示。

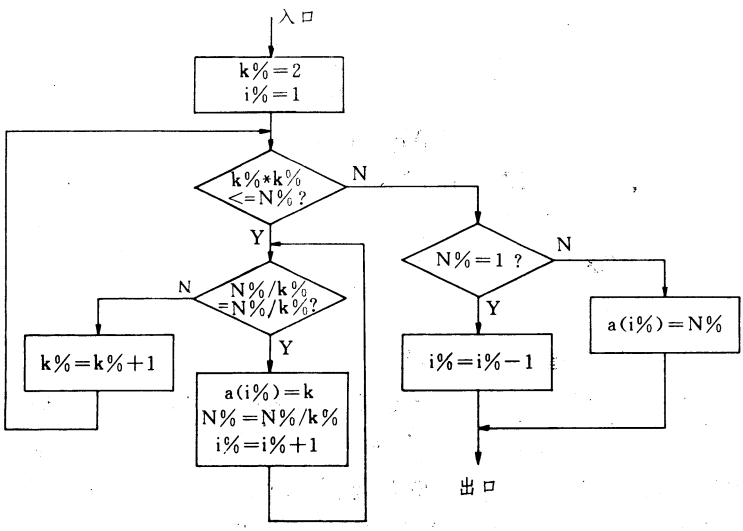


图 9.2.1　例9.2.3流程图

下面是主程序和子程序过程清单：

```
DECLARE SUB Factor (N%, a ( ) AS INTEGER, i%)
DIM Fac (1 TO 10) AS INTEGER
INPUT "N= ", N%
CALL Factor ( (N%), Fac ( ), i%)
PRINT "The factors of number"; N%; "are:"
FOR k%=1 TO i%
    PRINT Fac (k%);
NEXT k%
PRINT
END

SUB Factor (N%, a ( ) AS INTEGER, i%)
    k%=2
    i%=1
    DO WHILE k% * k%<=N%
        DO WHILE N% /k%=N%\k%
            a (i%) =k%
            N%=N%/k%
            i%=i%+1
        LOOP
        k%=k%+1
    LOOP
    IF N%=1 THEN
        i%=i%-1
    ELSE
        a (i%) =N%
    END IF
END SUB
```

由主程序输入一个整数N，调用过程Factor，计算整数N的

质因子，最后由主程序输出计算结果。

为子程序过程设置了三个参数：一个是整型数N，用于接受主程序送来的数据N，第二个是整型数组a，用以把计算结果送回主程序，第三个是整型数i，用以把求得的质因子个数传送给主程序。

在调用过程时，第一个参数N是按数值传送的，使得变量N的值在主程序中保持不变，虽然在过程中，这个变量的值已被改变。

当在主程序输入N＝126时，程序给出如下的输出：

The factors of number 126 are:

2 3 3 7

第三节 变量和常数的作用范围

无论何时出现变量，BASIC都有一定的规则来决定该变量在模块中所涉及的范围。变量若超出它的范围就是无效的，符号常数也是如此。

变量或符号常数只限于在模块的某一部分，如模块级代码部分或某一个过程内有效，称为局部变量或局部常数；变量或符号常数在整个模块的任何地方，包括模块级代码和过程中都有效时，称为全局变量或全局常数。

在Quick BASIC中，我们可以控制变量和符号常数的作用范围，以便设计出良好的程序。

本节讨论的中心内容便是如何控制变量或符号常数的作用范围。

一、局部变量和常数

本章第一节和第二节所涉及的程序中，都是局部变量。只有在模块级代码中，用CONST定义的符号常数才是全局符号常数。

以例9.2.3的程序为例，在模块级代码部分出现了变量N%和

k%，它们是在模块级代码中有效的局部变量。在SUB过程中也出现了变量N%和k%，但它们是在Factor过程内有效的局部变量。尽管名字相同，但它们是不同的变量。它们之间互不影响。

为了进一步说明局部变量的上述特点，我们考察下面的例题。

例 9.3.1 分析下面程序中的变量和符号常数：

```
DECLARE SUB test ( )
CONST P=3.1
i=2*P
CALL test
PRINT Pi, i, P
END

SUB test
    CONST pi=3.14159
    i=2*Pi
    PRINT Pi, i, P
END SUB
```

在模块级代码部分，有一符号常数P和变量i，根据前面讲的原则，P是全局常数，在没有说明的情况下，变量i只在模块级代码部分有效，因此，变量i是局部变量。

在过程代码部分，有符号常数P_i和变量i，它们只是在本过程代码部分有效，因此，它们是局部变量。本程序中，实际上有两个变量i，它们有各自的值。由于P是全局常数，所以在过程代码部分，也可以获得P的值。

通过上述分析，可以清楚地从程序的输出看出，P_i，i和P三个量在模块级代码部分的输出和在过程代码部分的输出如下：

	P_i	i	p
SUB	3.14159	6.28318	3.1
MainProg	0	6.2	3.1

二、全局变量

全局变量也称全程变量，这种变量在整个模块有效。

使变量成为全局变量的方法是，在模块级代码部分，用带有SHARED属性的COMMON语句，DIM语句或REDIM语句对变量加以说明。这些语句的格式如下：

COMMON SHARED variablelist

DIM SHARED variablelist

REDIM SHARED variablelist

其中SHARED是相应语句的属性，它表明变量在模块中的所有SUB和FUNCTION过程中都有效，也叫作共享。

variablelist是定义为全局变量的变量表。变量表有如下的格式：

variable[( )][AS type][,variable[( )][AS type]]…

其中variable是变量名，变量如果是数组，则变量名后跟圆括号。对于DIM和REDIM语句，括号内还应有维数的定义。变量类型type可以是INTEGER,LONG,SINGLE,DOUBLE或用户定义类型。

例如下面的语句定义整型数组A和长整型变量B为全局变量：

COMMON SHARED A ( ) AS INTEGER, B AS LONG

DIM SHARED A (1 TO 20) AS INTEGER, B AS LONG

REDIM SHARE A (1 TO 5) AS INTEGR

例9.32 下面程序含有两个SUB过程，分别用于计算一批数据的平均值和均方差。程序全部使用局部变量，数据由主程序输入，然后依次调用过程并输出计算结果。下面是完整的程序清单：

```
DIM NumValues AS INTEGER
DIM Mean AS DOUBLE, StdDev AS DOUBLE
DIM Values (1 TO 20) AS DOUBLE
```

```
INPUT "Enter the number of values ", NumValues

FOR i% = 1 TO NumValues
    PRINT "Values ("; i%; ") ="
    INPUT "  ", Values (i%)
NEXT i%

CALL MeanSub (Values (  ), NumValues, Mean)
PRINT "Mean="; Mean

CALL StdDevSub (Values (  ), NumValues, StdDev)
PRINT "Standard Deviation="; StdDev

END

SUB MeanSub (Values (  ) AS DOUBLE, NumValues
             AS INTEGER, Mean AS DOUBLE)
    FOR i% = 1 TO NumValues
        Sum# = Sum# + Values (i%)
    NEXT i%
    Mean = Sum# / NumValues
END SUB

SUB StdDevSub (Values (  ) AS DOUBLE, NumValues
               AS INTEGER, StdDev AS DOUBLE)
    FOR i% = 1 TO NumValues
        Sum# = Sum# + Values (i%)
        SumSq# = SumSq# + Values (i%) ^ 2
    NEXT i%
    StdDev = SQR ( SumSq# / NumValues
           - (Sum# / NumValues) ^ 2)
END SUB
```

程序运行结果举例如下：

Values (1) =1

Values (2) =2

Values (3) =3

Mean=2

Standard Deviation=.8164965809277263

现在让我们把程序中下列局部变量改为全局变量：

存放数据的变量 Values (1 TO 20)；

存放数据个数的变量 NumValues；

平均值变量 Mean；

标准差变量 StdDev。

这些变量通过 DIM SHARED 或 COMMON SHARED 语句说明，便成为全局变量。成为全局变量意味着模块级代码和过程都可以直接访问这些变量，或者说，这些变量在整个模块是共享的。

共享变量是没有必要设置在过程的参数表中进行传送的，可见，全局变量也是模块级代码和过程之间传送数据的一种方法。

这样，若要把上面给出的用局部变量编写的程序，改为采用全局变量编写，只须作两点变化：

(1) 补充具有 SHARED 属性的说明语句；

(2) 删去过程参数表中相关的参数。

修改后的程序如下：

```
DECLARE SUB MeanSub ( )
DECLARE SUB StdDevSub ( )

DIM SHARED NumValues AS INTEGER
DIM SHARED Mean AS DOUBLE, StdDev AS DOUBLE
DIM Values (1 TO 20) AS DOUBLE
COMMON SHARED Values ( ) AS DOUBLE

INPUT "Enter the number of values ", NumValues

FOR i%=1 TO NumValues
```

```
        PRINT "Values ("; i%; ") ="
        INPUT "  ", Values (i%)
NEXT i%

CALL MeanSub
PRINT "Mean"; Mean

CALL StdDevSub
PRINT "Standard Deviation="; StdDev

END

SUB MeanSub
    FOR i%=1 TO NumValues
        Sum#=Sum#+Values (i%)
    NEXT i%
    Mean=Sum#/NumValues
END SUB

SUB StdDevSub
    FOR i%=1 TO NumValues
        Sum#=Sum#+Values (i%)
        SumSq#=SumSq#+Values (i%) ^ 2
    NEXT i%
    StdDev =SQR (SumSq#/NumValues
          - (Sum#/NumValues) ^ 2)
END SUB
```

三、用 SHARED 语句定义共享变量

有时我们并不需要把某些变量定义为在整个模块共享的全局变量，而只需在模块的个别过程与模块级代码之间共享时，可以把它们定义为局部的共享变量。例如在例9.3.2中的变量 Mean 和 StdDev 便可定义成这种共享变量。

定义局部共享的共享变量的方法是，在共享该变量的过程内，

用 SHARED 语句说明该变量。SHARED 语句的格式是：

SHARED variable [AS type] [, variable [AS type]] …

例如，在例9.3.2的 MeanSub 过程中，用语句

SHARED Mean AS DOUBLE

可以使变量 Mean 在模块级代码和 MeanSub 过程之间成为共享变量。

要特别指出，SHARED 语句只能出现在 SUB 和 FUNCTION 过程中，它只能使变量在同一模块中的模块级代码和过程之间共享。

例 9.3.3 将例9.3.2中采用全局变量的程序，改 Mean 和 StdDev 为局部共享变量。

为此目的，在模块级代码中，不需把变量 Mean 和 StdDev 说明为全局变量。在过程 MeanSub 增加语句 SHARED Mean 和在 StdDevSub 过程增加语句 SHARED StdDev。

修改后的程序如下：

```
DECLARE SUB MeanSub (  )
DECLARE SUB StdDevSub (  )

DIM SHARED NumValues AS INTEGER
DIM Mean AS DOUBLE, StdDev AS DOUBLE
DIM SHARED Values (1 TO 20) AS DOUBLE

INPUT "Enter the number of values ", NumValues

FOR i%=1 TO NumValues
      PRINT "Values ("; i%; ") =";
      INPUT "  ", Values (i%)
NEXT i%

CALL MeanSub
PRINT "Mean="; Mean
```

```
CALL StdDevSub
PRINT "Standard Deviation="; StdDev
END

SUB MeanSub
    SHARED Mean AS DOUBLE
    FOR i%=1 TO NumValues
        Sum#=Sum#+Values (i%)
    NEXT i%
    Mean=Sum#/NumValues
END SUB

SUB StdDevSub
    SHARED StdDev AS DOUBLE
    FOR i%=1 TO NumValues
        Sum#=Sum#+Values (i%)
        SumSq#=SumSq#+Values (i%) ^ 2
    NEXT i%
    StdDev =SQR (SumSq#/NumValues
          - (Sum#/NumValues) ^ 2)
END SUB
```

第四节 STATIC 变量和 STATIC 语句

一、STATIC 变量

BASIC 过程中的局部变量，在没有特殊说明的情况下，每次调用 FUNCTION 或 SUB 时，都被初始化，即数值变量置成0，字符串变量置为空。

如果在过程的定义行，加上 STATIC 属性，例如，

FUNCTION A(x,y) STATIC

SUB B(x,y) STATIC

则过程中的局部变量都是 STATIC 变量。STATIC 变量在各次调用之间，保持其值不变。

过程应用 STATIC 属性，可以使程序的运行速度多少有些增加。

例 9.4.1 本例给出两个形式上相同的子程序过程 TestSub1和 TestSub2，但前者无 STATIC 属性，后者有 STATIC 属性。子程序的内容都是如下的两条语句：

```
A=A+1
PRINT A
```

其中 A 为过程的局部变量。

程序如下：

```
DECLARE SUB TestSub1 ( )
DECLARE SUB TestSub2 ( )

FOR i=1 TO 5
    CALL TestSub1
NEXT i

FOR i=1 TO 5
    CALL TestSub2
NEXT i

END


SUB TestSub1
    x=x+1
    PRINT x
END SUB

SUB TestSub2 STATIC
```

```
    y=y+1
    PRINT y
END SUB
```

主程序五次调用过程 TestSub1，每次输出的 x 值都是1，这是因为每次调用时，x 的值都初始化为0。

主程序五次调用过程 TestSub2，五次输出的 y 值依次是1，2，3，4和5。这是因为过程 TestSub2具有 STATIC 属性，使它的局部变量 y，在每次调用时，保持其原值不变。

二、STATIC 语句

我们已经知道，过程的 STATIC 属性，使它的局部变量成为 STATIC 变量。有时我们希望过程中的某些变量是 STATIC 变量，而另一些则不是，这时，过程的定义行，不能带有 STATIC，而是将 STATIC 变量，在过程中用 STATIC 语句加以说明。STATIC 语句的格式是：

```
STATIC variablelist
```

其中 variablelist 的格式，在前面提到过，不再重复。

被 STATIC 语句说明的变量，既或是在模块级代码中已被说明为共享变量，也要将其变为局部变量。语句 STATIC 只用在过程体内。

例 9.4.2 分析下面程序的输出：

```
DECLARE SUB TestSub (  )
FOR i=1 TO 5
    CALL TestSub
NEXT i
END

SUB TestSub
```

```
    STATIC y
    x=x+1
    y=y+1
    PRINT x, y
END SUB
```

不难看出，这个程序是将例9.4.1的两个过程 TestSub1和TestSub2合并为一个过程 TestSub。其中变量 y 仍保持为 STATIC 变量，因为过程体内有 STATIC y 语句。

由以上简单分析可以判定，在五次调用过程中，变量 x 和变量 y 的五个值，是和例9.4.1一样的。本程序输出如下：

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

第五节　多模块程序结构

到目前为止，我们讨论的都是由一个模块（也称主模块）组成的程序。对于完成多功能和比较复杂的程序，若采用多模块结构的程序，可给程序的编制、调试和修改带来很大的方便。

本节介绍两种多模块结构的程序。一种是组成程序的各模块同时装入内存的多模块程序；另一种是用 CHAIN 语句链接外存中模块的多模块程序。

一、多模块程序

1. 多模块程序的建立与保存

BASIC 把每一个文件都看作一个模块，因此，设计多模块程序的第一步，是建立每个单独模块文件。例如，把例9.1.2定义的

函数过程 Fac＃建立为一个独立文件，假设文件名为 FACP. BAS，还可以把例9. 1. 3中的主程序建立为一个独立文件，假设文件名为 FACM. BAS，于是我们有两个程序模块，它们是两个文件。

我们已经知道如何在 Quick BASIC 环境建立程序模块。现在来介绍一种更为方便的方法，这就是用 File 菜单的 Create File 命令。这个命令被调用后，出现一个如图9. 5. 1所示的对话框，利用这个对话框，可以建立三种不同的文件：模块文件（Module），包含文件（Include）和文本文件（Document）。现在，写入所要建立的文件名并选择 Module 项，便在内存中把模块文件建立起来了，屏幕回到编辑状态，可开始编辑程序。

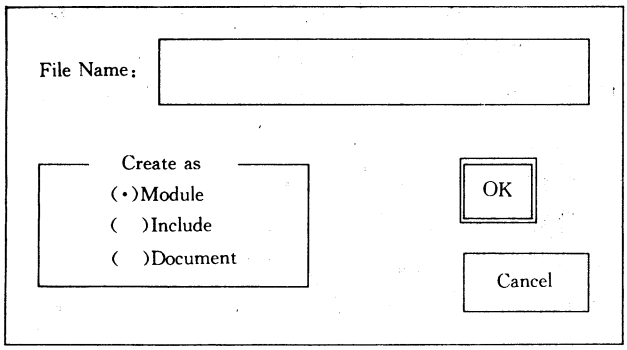


图 9. 5. 1　Create File 对话框

与以前建立程序模块的方法不同，用 Create File 命令建立文件时，不仅可以建立三种不同的文件，而且，更重要的是，它不影响已存于内存的其他文件，或者说，内存中可以同时存在多个模块。这非常有利于多模块程序的操作，如编辑和调试。

每调试完一个模块，就可以用 File 菜单的 Save 命令，将它存入磁盘。

在多模块程序的编辑调试过程中，要显示和编辑某个模块时，可通过 View 的 SUBs 命令来选择。操作图9. 4. 1的对话框，通过

选择 Split Window 命令，在屏幕上开辟两个窗口，所选择的模块将在新的窗口显示。

在组建多模块程序过程中，有时需要将某模块从程序中除去，使该模块不再是程序的一部分。其操作步骤如下：

(1) 选择 File 菜单的 Unload 命令，显示出含有当前装入的所有模块文件名的对话框；

(2) 在对话框上指定所要删除的模块名；

(3) 按回车键，该模块被除去。

但是，经上述操作，从程序中除去的模块，仍以磁盘文件的形式存在，只是本程序不再包括该模块而已。

如果多模块程序中的某些模块，已经事先建好并存储在磁盘中，应按下列步骤将它们装入内存，使其作为当前多模块程序中的一个模块。装入模块的步骤如下：

(1) 选择 File 菜单的 Load File 命令，这时显示出如图3.3.4所示的对话框；

(2) 选择所要装入的模块的文件名；

(3) 选择 Load AS Module 项；

(4) 按回车键，指定模块被装入。

使用 Load File 装入模块时，不影响已在内存中建立或装入的其他模块。

当所需要的模块均建立或装入后，建立多模块程序的第二步，是设置主模块。

在多模块程序中，含有第一条执行语句的模块，称为主模块(Main Module)。BASIC 系统可自动地把最先装入或建立的模块设置为主模块，但是，根据需要，可以设置任何一模块为程序的主模块。

设置主模块的操作方法：

(1) 选择 RUN 菜单的 Set Main Module 命令，于是出现一个含有装入的模块清单的对话框，如图9.5.2所示。

(2) 选择指定为主模块的那个模块，按回车键，完成选择。

应用 File 菜单的 Unload 命令，还可以从装入的模块中去掉某些不想要的模块。

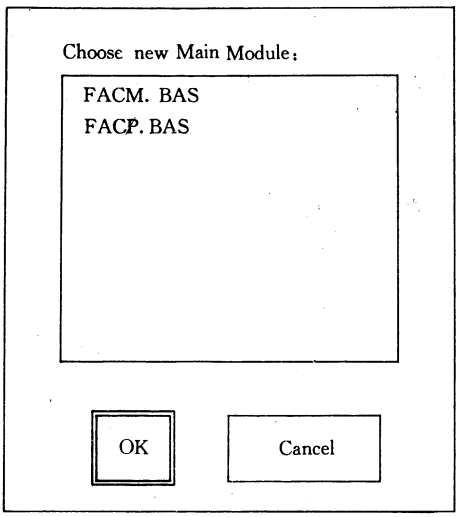


图 9.5.2 Set Main Module 对话框

当所需要的模块都装入内存，便可进行程序的调试和试运行。程序调试结束，应把多模块程序存入磁盘。

多模块程序的存盘，是用 File 菜单的 Save All 命令实现的。当调用这个命令时，主模块文件名就是当前的多模块程序文件名，修改后的各模块（包括主模块），分别存入各模块的磁盘文件中，同时产生一个以主模块文件名为文件名，以·MAK 为扩展名的 MAK 文件。这是一个 ASCII 码文件，文件中记录着构成此多模块程序的模块文件名。

当再次装入多模块程序时，BASIC 将按 MAK 文件，把各模块自动装入内存。装入多模块程序的操作方法，与装入单模块程序的操作方法一样。

2. **各模块之间数据的传送**

多模块程序设计的核心问题是解决好各模块之间数据的正确传送。解决此问题有以下三种方法：

（1）和单模块程序一样，在没有说明的情况下，各模块中的变量，都是局部变量。在这种情形下，一个模块调用另一个模块中的过程时，采用自变量向过程传送的方法，把数据传送给过程。这与单模块程序中采用的方法没有什么不同。下面通过例子来说明。

例 9.5.1 将例9.3.2中模块级代码、MeanSub 过程和 StdDevSub 过程分别构成独立模块，然后用上述三个模块组成一个多模块程序。

第一个模块（主模块）：

```
DECLARE SUB MeanSub (x! (  ), n!, y!)
DECLARE SUB StdDevSub (x! (  ), n!, y!)
DIM Values (1 TO 20)
INPUT "Enter the number of Values", NumValues
FOR i=1 TO NumValues
    PRINT "Values ("; i; ") =";
    INPUT "  ", Values (i)
NEXT i
CALL MeanSub (Values (  ), NumValues, Mean)
PRINT "Mean="; Mean
CALL StdDevSub (Values (  ), NumValues, StdDev)
PRINT "Standard Deviation="; StdDev
END
```

第二个模块：

```
SUB MeanSub (x (  ), n, y)
    FOR i=1 TO n
        Sum=Sum+x (i)
    NEXT i
```

```
    y=Sum/n
END SUB
```

第三个模块：

```
SUB StdDevSub (x (  ), n, y)
    FOR 1=1 TO n
        Sum=Sum+x (i)
        SumSq=SumSq+x (i) ^ 2
    NEXT i
    y=SQR (SumSq/n- (Sum/n) ^ 2)
END SUB
```

从上面这个例子可以看出，在采用自变量——参数的方式在模块之间传送数据的条件下，各模块的程序，与单模块时各部分的程序没有什么不同(我们有意将双精度变量改为单精度变量，整型变量也改为单精度变量)，差别仅在于由一个模块变为三个模块了。

(2) 采用模块之间共享变量的方法，也可以把数据从一个模块传送至另一个模块。

在不同模块中共享变量，需要在每个模块的模块级代码中，用语句 COMMON 或 COMMON SHARED 说明那些变量。

例如，在主模块中有语句：

```
COMMON A, B
```

在定义过程的模块中有语句：

```
COMMON SHARED X, Y
```

则第一个模块中的变量 A 和第二个模块中的变量 X 是同一个共享变量，它们虽然名字不同，但类型相同，并且是根据它们在 COMMON 语句中的位置，确定其一一对应关系。同样，第一个模块中的变量 B 和第二个模块中的变量 Y，是同一个共享变量。借助于语句 COMMON SHARED 语句，第二个模块的所有过程，都能访问共享变量。

例 9.5.2 用模块间共享变量的方法，重新编写例9.5.1的多模块程序。

在这个程序中，可定义 Values ( )，NumValues，Mean 和 StdDev 四个变量为共享变量。为此，在各模块的模块级代码，需增加 COMMON 和 COMMON SHARED 语句。由于上述四个变量已定义为共享的，因此，MeanSub 和 StdDevSub 两个过程，就不需带参数了。

于是，有如下三个模块：

第一个模块（主模块）：

```
DECLARE SUB MeanSub ( )
DECLARE SUB StdDevSub ( )
DIM Values (1 TO 20)
COMMON Values ( ), NumValues
COMMON Mean, StdDev
INPUT "Enter the number of values", NumValues
FOR i%=1 TO NumValues
    PRINT "Values (";i%;") =";
    INPUT " ", Values (i%)
NEXT i%
CALL MeanSub
PRINT "Mean=";Mean
CALL StdDevSub
PRINT "standard Deviation=";StdDev
END
```

第二个模块

```
DIM SHARED  x (1 TO 20)
COMMON SHARED  x ( ), n
COMMON SHARED  y, z
SUB MeanSub
```

```
    FOR i=1 TO n
        Sum=Sum+ x(i)
    NEXT i
    y=Sum/n
END SUB
第三个模块
DIM SHARED  x (1 TO 20)
COMMON SHARED  x(  ), n
COMMON SHARED  y, z
SUB stdDevSub
    FOR i=1 TO n
        Sum=Sum+ x(i)
        SumSq=SumSq+ x(i) ^ 2
    NEXT i
    z=SQR (SumSq/n- (Sum/n) ^ 2)
END SUB
```

不同模块的共享变量，是严格按照它们在各自模块中的COMMON 语句的顺序，决定其共享关系的。在共享变量较多的情况下，可把共享变量分组，并给以组名，将会给程序设计带来方便。带有组名的 COMMON 语句有如下的格式：

COMMON [SHARED] /组名/变量表

当共享变量定义在有组名的 COMMON 语句中时，不同模块间变量的共享关系，首先按组名对应，其次则按在该同名语句内的顺序对应。

在 COMMON 语句中采用组名，将给程序定义变量带来一定灵活性，这可从下面的例子看出。

例 9.5.3　给例9.5.2程序中的 COMMON 语句增加组名。

将共享变量 Values ( ) 和 NumValues 编为一组，命名为 a 组。这两个变量的特点是程序的所有模块，都需要访问它们。

变量 Mean 定义为 b 组，只有第二个模块涉及它。变量 StdDev 定义为 c 组，只有第三个模块涉及它。

由于第二个模块不需共享变量 StdDev，第三个模块不需共享变量 Mean，因此，在各模块的模块代码级，省去相应的COMMON SHARED 语句。

于是，我们有如下的程序：

第一个模块：

```
DECLARE SUB MeanSub ( )
DECLARE SUB StdDevSub ( )
DIM Values (1 TO 20)
COMMON/a/Values ( ), NumValues
COMMON/b/Mean
COMMON/c/StdDev
INPUT "Enter the number of values", NumValues
FOR i%=1 TO NumValues
    PRINT "Values ("; i%; ") ="
    INPUT "  ", Values (i%)
NEXT i%
CALL MeanSub
PRINT "Mean="; Mean
CALL StdDevSub
PRINT "standard Deviation="; StdDev
END
```

第二个模块：

```
DIM SHARED  x (1 TO 20)
COMMON SHARED/a/  x ( ), n
COMMON SHARED/b/y
SUB MeanSub
    FOR i=1 TO n
```

```
            Sum=Sum+ x(i)
        NEXT i
        y=Sum/n
    END SUB
```

第三个模块：

```
    DIM SHARED  x(1 TO 20)
    COMMON SHARED/a/  x(  ), n
    COMMON SHARED /c/y
    SUB StdDevSub
        FOR i=1 TO n
            Sum=Sum+ x(i)
            SumSq=SumSq+ x(i) ^ 2
        NEXT i
        y=SQR (SumSq/n-(Sum/n) ^ 2)
    END SUB
```

(3) 模块间共享变量的另一种方法是，在各模块的模块级代码部分，均用不带 SHARED 的 COMMON 语句列出相关的变量。至于哪个变量与哪个过程共享，分别由各过程自己用 SHARED 语句来定义。而 COMMON SHARED 定义的变量则在整个模块内各过程都是共享的，这是两者的重要区别。

例 9.5.4 应用不带 SHARED 属性的 COMMON 语句，重写例9.5.3的程序。

第一个模块（主模块）与例9.5.3同，这里不再重复。

第二个模块：

```
    DIM  x(1 TO 20)
    COMMON  /a/  x(  ), n
    COMMON  /b/y
    COMMON  /c/z
    SUB MeanSub
```

```
    SHARED  x(  ), n, y
    FOR i=1 TO n
        Sum=Sum+ x(i)
    NEXT i
    y=Sum/n
END SUB
```

第三个模块：

```
DIM  x(1 TO 20)
COMMON  /a/x(  ), n
COMMON  /c/z
COMMON  /b/y
SUB StdDevSub
    SHARED x(  ), n, z
    FOR i=1 TO n
        Sum=Sum+ x(i)
        SumSq=SumSq+ x(i) ^ 2
    NEXT i
    z=SQR(SumSq/n- (Sum/n) ^ 2)
END SUB
```

3. 包含文件（Include File）的应用

包含文件是一种文本文件（ASCII 码文件），用一条 $INCLUDE 元命令，可将它编入（或插入）程序的任何部分。包含文件通常以扩展名.BI 作为标志。

包含文件与任何其他文本文件不同之处是，在它的文本中不可含有 SUB…END SUB 或 FUNCTION…END FUNCTION。

包含文件的最主要用途是，用于编写多模块程序的说明语句。例9.5.4的程序中，第二个模块和第三个模块的说明语句，是完全一样的。如果三个模块的这些共享变量，取相同的变量名，例如 x( )，n，y 和 z 分别统一为 Values( )，NumValues，Mean 和

StdDev，那么，这三个模块的说明部分将是完全一样的，都是如下的四个语句：

```
DIM Values (1 TO 20)
COMMON  /a/  Values ( ), NumValues
COMMON  /b/  Mean
COMMON  /c/  StdDev
```

为了避免在每个模块都重复输入这些语句，我们用这段语句建立一个包含文件，例如 IncFil.BI，然后，在每个模块需要这些说明语句的地方，用元命令 $INCLUDE 把这个包含文件引入：

```
REM $INCLUDE:'IncFil.BI'
```

元命令的格式规定，元命令必须写在注释语句中。

BASIC 在编译上述元命令语句时，便把相应包含文件的内容，插入元命令处。

包含文件可用 File 菜单的 Create File 命令，并通过图9.5.1所示对话框的操作建立。要注意先选择 Create as Include 这一项。如果文件名中省略了扩展名，系统会自动赋给一个.BI 扩展名。

例 9.5.4 修改例9.5.3，使之应用前面给出的包含文件 IncFil.BI。

程序如下：

主模块：

```
DECLARE SUB MeanSub ( )
DECLARE SUB StdDevSub ( )
'$INCLUDE, 'IncFil.bi'
INPUT "Enter the number of values", NumValues
FOR i%=1 TO NumValues
    PRINT "Values ("; i%; ") ="
    INPUT "  ", Values (i%)
NEXT i%
CALL MeanSub
```

```
PRINT "Mean="; Mean
CALL StdDevSub
PRINT "Standard Deviation="; stdDev
END
```

第二个模块：

```
REM $INCLUDE: 'IncFil.bi'
SUB MeanSub
    SHARED Values ( ), NumValues, Mean
    FOR i=1 TO NumValues
        Sum=Sum+Values (i)
    NEXT i
    Mean=Sum/NumValues
END SUB
```

第三个模块：

```
REM $INCLUDE: 'IncFil.bi'
SUB StdDevSub
    SHARED Values ( ), NumValues, stdDev
    FOR i=1 TO NumValues
        Sum=Sum+values (i)
        SumSq=SumSq+Values (i) ^ 2
    NEXT i
    StdDev =SQR ( SumSq/NumValues
          - ( Sum/NumValues) ^ 2)
END SUB
```

二、链接方式的多模块程序

用 CHAIN 语句链接的多模块程序，与前面介绍的多模块程序不同，它的各模块不需要同时装入内存，只有当前运行的模块存在于内存中。它用 CHAIN 语句将另一模块调入内存并运行它，

同时，原来在内存中运行的模块被清除。因此，从一个模块链接到另一个模块时，总是按下列顺序进行：

（1）当前模块执行 CHAIN 语句；

（2）下一个模块装入内存，清除前一个模块；

（3）运行被装入的模块。

CHAIN 语句的格式如下：

CHAIN “filespec”

其中 filespec 是文件名。语句的功能是将控制权从当前的程序模块传送到名为 filespec 的程序模块。

模块之间用 COMMON 语句（不带模块名）传递变量，也可以用包含文件，专门记录各模块之间需要传送的变量，这时，每个模块的开头使用 $INCLUDE 命令。

用 CHAIN 语句链接的多模块程序的优点是，能把占用大量内存的程序，分隔成若干个小程序，从而缓解内存空间紧张的问题。

例 9.5.5 用链接方法实现例9.5.4的多模块程序，同时，作如下两点变化：

（1）除主模块外，再设置三个模块。第一个模块计算并输出平均值，第二个模块计算并输出标准差，第三个模块计算平均值和标准差并输出。

（2）主模块按照用户需要，选用上述三个模块中的任意一个。

模块间需要传送的变量有 Values（ ）和 NumValues。为此，建立一个名为 “commons.bi” 的包含文件。文件内容如下：

```
DIM Values (1 TO 20)
COMMON Values ( ), NumValues
```

下面给出各个模块的程序。

模块1：

```
REM $INCLUDE: 'commons.bi'
INPUT "Enter the number of data values", NumValues
```

```
FOR i=1 TO NumValues
    PRINT "Values ("; i; ") ="
    INPUT Values (i)
NEXT i
PRINT   "1 mean"
PRINT   "2 standard deviation"
PRINT   "3 mean and standard deviation"
INPUT   "Calculation number"; Choice
SELECT CASE Choice
   CASE 1
      CHAIN "Mean"
   CASE 2
      CHAIN "StdDev"
   CASE 3
      CHAIN "MeanSt"
END SELECT
END
```

模块2：

```
'$INCLUDE: 'commons.bi'
FOR i=1 TO NumValues
    Sum=Sum+Values (i)
NEXT i
Mean=Sum/NumValues
PRINT "The Mean is:"; Mean
CHAIN "main"
END
```

模块3：

```
'$INCLUDE:'commons.bi'
FOR i=1 TO NumValues
```

```
        Sum=Sum+Values (i)
        SumSq=SumSq+Values (i) ^ 2
    NEXT i
    StdDev =SQR (SumSq/NumValues
            - (Sum/NumValues) ^ 2)
    PRINT "The Standard Deviation is:"; StdDev
    CHAIN "main"
    END
```

模块4：

```
' $ INCLUDE: 'commons.bi'
    FOR i=1 TO NumValues
        Sum=Sum+Values (i)
        SumSq=SumSq+Values (i) ^ 2
    NEXT i
    Mean=Sum/NumValues
    StdDev =SQR (SumSq/NumValues
            - (Sum/NumValues) ^ 2)
    PRINT "The Mean is:"; Mean
    PRINT "The Standard Deviation is:"; StdDev
    CHAIN "Main"
    END
```

第六节 程序库的应用

程序库是一种含有一个模块或一个组模块全部过程的特殊文件，当把这个库装入内存时，其他程序可以调用库中的任何一个过程。这一节将说明怎样建立和使用程序库。

一、库的建立

Quick BASIC 能建立两种类型的库，这两种库可以根据文件

的扩展名加以区分。一种是以.QLB为扩展名的库，供在Quick BASIC环境中使用。这种库也叫“快速库”。另一种是以.LIB为扩展名的库，供生成独立的EXE文件时调用。这种库也叫“独立库”。

两种库既可以在Quick BASIC环境建立，也可以在DOS下建立。本节只介绍在Quick BASIC环境下建立库的方法。

在Quick BASIC环境下建立程序库，除了文件QB.EXE外，还需要下列文件：

BC.EXE

LINK.EXE

LIB.EXE

BQLB40.LIB

建立库的步骤如下：

(1) 用File菜单中Load File命令，将想要放入库中的模块，逐个装入内存；

(2) 选择Run菜单中的Make Library命令。这条命令给出如图9.6.1所示的对话框。

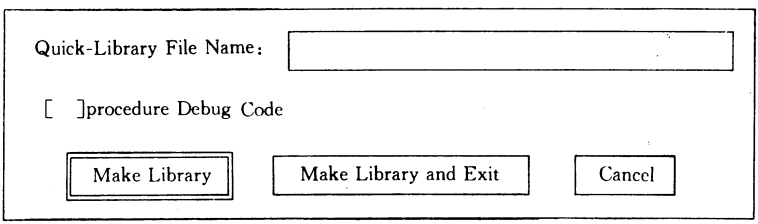


图 9.6.1 Make Library 对话框

(3) 在对话框的Quick Library File Name栏中输入所希望建立的快速库的文件名。

如果只输入基本名，BASIC自动给出一个扩展名.QLB。如果想要这个库没有扩展名，需把一个句号加到这个基本名之后，就

可以输入任意所需的，符合 DOS 命名规则的基本名和扩展名。

在对话框中，Produce Debug Code 栏是任选的。选择此栏时，将产生检测码，这对调试库是有用的，但它将使库增大，运行速度减慢，而且仅给出少量错误控制。一般不必选择此栏。

对话框的下部，有三个栏目，为用户提供三种选择。如果希望在库建立之后，仍然保持在原环境下，则可以选择 Make Library 栏。双线框表示已选中。如果希望在库建立之后，回到 DOS 命令状态，则选择 Make Library and Exit 栏。选择 Cancel 栏表示取消当前的命令。

(4) 按回车键，开始建库。在建库的过程中，需要上面提到的一些文件，当找不到所需文件时，会给出相应的提示。

库建成后，磁盘上将有.QLB 文件和.LIB 文件。它们都是机器语言文件。

二、使用快速库

快速库需要在启动 Quick BASIC 时，同 Quick BASIC 一起装入内存。命令格式是：

QB/L＜库文件名＞

快速库装入内存后就可以使用了。

调用库内的过程，与调用一般的过程没有任何区别。

快速库中的过程可以从立即窗口中直接执行，就像执行一个 BASIC 语句一样。

例如，我们可以将例9.5.1中的第二个模块和第三个模块建成一个快速库，名为 Meastd.QLB。将此库装入内存，再装入例9.5.1中的第一个模块(设文件名为 MAIN.BAS)，便可以运行这个程序了。

所要装入的模块和库，可用一条命令，与 Quick BASIC 同时装入。这个命令有如下的格式(以上述的模块和库为例)：

QB MAIN/L MEASTD

如果希望在装入库和必要的模块后立即运行，可用如下格式的命令：

QB/RUN＜文件名＞ /L＜库文件名＞

三、向快速库增添新模块

有时我们需要把一个调试好的新模块放进已存在的一个快速库中，供以后使用。

为了把模块加入到一个快速库中，首先应把那个库装入内存，然后，装入要放进库中的那些模块，最后，用 RUN 菜单的 Make Library 命令建立新库。这时，新库中包含了原来库中的内容和新增加进去的内容。

小　　结

本章介绍了 Quick BASIC 系统在结构化程序设计方面提供的手段，主要讲了三个方面的问题：

(1) 过程的概念和应用；

(2) 多模块程序的设计；

(3) 程序库的建立和应用。

关于过程，我们要掌握好以下几个方面的问题：

(1) 函数过程和子程序过程的不同；函数是怎样定义和调用的；子程序是怎样定义和调用的；

(2) 过程调用语句中的自变量与过程定义语句中的参数之间的数据传送；

(3)程序中变量和符号常数的作用范围问题。这里涉及到局部变量和全局变量的概念，变量共享及实现变量共享的方法，过程的 STATIC 属性和 STATIC 语句的应用等。

关于多模块程序的设计，要着重掌握的问题是：

(1) 多模块程序的建立方法；

(2) 模块之间数据的传送方法和所包含文件的应用；

(3) 多模块程序的链接方法。

将许多常用的过程组建成一个程序库，供其他程序调用，给程序设计带来一定的方便。本章最后简要地介绍了 Quick BASIC 提供的两种库：以.QLB 为扩展名的快速库和以.LIB 为扩展名的独立库。学过此内容后，我们要会建立程序库和使用程序库。

习　　题

9-1 说明函数过程定义语句的格式和调用函数过程的方法。

9-2 说明子程序过程定义语句的格式和调用子程序过程的方法。

9-3 什么是全局变量?什么是局部变量?怎样使变量成为全局变量?怎样使变量成为模块某些部分的共享变量?

9-4 通过引用传送自变量和通过数值传送自变量有何区别?

9-5 过程的 STATIC 属性有何作用?STATIC 语句的功能是什么?

9-6 利用阶乘的函数过程和子程序过程，编两个计算下式的程序：

$$(3!+4!+5!)/6!$$

9-7 将例9.2.2程序中的 Array (i) 改为全局变量，常数20改为符号常数，重新编写例9.2.2的程序。

9-8 编写两个子程序过程，其中一个过程计算并打印一个数的平方，另一个过程计算并打印一个数的立方，这两个过程组成一个独立的模块。再设计一个主模块，这个模块给定一个数，然后依次调用上述的两个过程。要求主模块给定的数据与过程块是共享变量。试写出上述两个模块的程序。

9-9 将上例中的两个过程建成一个库，然后用主模块调用库中的过程。

第十章　字符串

本章讲解如何对 ASCII 码字符串进行处理，如字符串的合并、比较、查找、更改、变换等。字符串处理在文本文件、报表生成以及事务管理等软件设计中，都是不可缺少的。

Quick BASIC 的字符串处理功能，大部分是用字符串函数实现的，因此，本章将介绍字符串处理中应用的各种函数。

第一节　字符串常数与字符串变量

一、字符串常数

一个字符串，就是一个字符序列。字符可以是大写或小写的英文字母、数字、标点符号、运算符号以及其他特殊符号等。字符串常数可以用下述两种方法进行描述。

(1) 用双引号括起来的一组字符，就是一个字符串，例如打印一个字符串常数：

```
PRINT"Please wait…"
```

但是，DATA 语句中的常数字符串，如果在其中不含有逗号和冒号，或者其首或其尾不含有空格，则允许省略双引号。例如：

```
DATA abcd,1234,"x:=?"
```

在这个 DATA 语句中，共有三个常量，其中第一和第三个，显然是字符串常数。第二个量是字符串常数还是数字常数，要由 READ 语句中的相应变量的类型来决定。如果把第二个量用双引号括起来，它就是字符串常数。

(2) 用 CONST 语句定义字符串常数，例如：

CONST MESSAGE＝“Waiting”

这里符号 MESSAGE 代表字符串常数“Waiting”。

二、字符串变量

字符串变量有三种描述方法，现分述如下：

(1) 变量名后带有字符串变量说明符(后缀)$。例如，字符串赋值语句：

a$＝“1204kb”

这里变量 a$ 是字符串变量。

(2) 使用 DEFSTR 语句定义字符串变量。例如：

DEFSTR B,M-O

经过这样定义后，以字母 B 或以字母 M 到 O 开头的变量，都是字符串变量。例如，可以有如下的赋值语句：

nstr＝“nnnnn”

需要注意的是，类型说明符，包括 $，%，&，! 或 #，总是比 DEFtype(type 可以是 INT，SNG，DBL，LNG 或 STR)优先起作用，这就是说，尽管定义了以 B 字母开头的变量是字符串变量，整型变量 B% 仍是合法的。

(3) 使用带有 AS STRING 的语句，如用 DIM 语句来定义字符串变量 astr：

DIM astr AS STRING

除了 DIM 语句之外，COMMON，REDIM，SHARED 语句都可以带有 AS STRING，因而也可以用来定义字符串变量。

以上各例中，介绍的都是简单变量。类似数值变量，也有字符串数组变量，它们同样可用上述方法加以定义。

三、可变长度和固定长度字符串

Quick BASIC 提供可变长度和固定长度两种字符串。

字符串含有的字符数是字符串的长度。可变长度字符串的长

度，随着它所存储字符的多少而变化。

例如，有字符串赋值语句

```
Temp$ ="1234567"
```

则字符串 Temp$ 的长度是 7。如果对这个变量重新进行赋值：

```
Temp$ ="1234"
```

则字符串 Temp$ 的长度变为 4 了。

固定长度字符串的长度由相应语句定义，长度一旦被定义就不能再改变。固定长度字符串所存储的字符数不能超过其规定的长度，否则，多余的字符会从右边丢失。

固定长度字符串一般是用作公共性的记录单元。固定长度字符串可用 COMMON，DIM，REDIM，SHARED 来定义。例如，用 DIM 语句定义一个长度为 10 个字符的字符串变量 Buffer，语句格式如下：

```
DIM Buffer AS STRING * 10
```

固定长度字符串，还可以直接在用户定义数据类型的定义中加以定义。例如：

```
TYPE Rectype
Buffer AS STRING * 10
...
END TYPE
```

当给固定长度字符串变量赋的值的字符数小于它的长度时，字符串后面用空格补足。当给固定长度字符串变量赋的值的字符数大于它的长度时，则字符串后面多余的字符自动丢失。

例 10.1.1 在这个例子中，用 DIM 语句定义了两个固定长度的字符串。在给它们赋值时，一个字符串超过了定义的长度，一个小于给定的长度，让我们来观察程序的输出格式。

```
DIM LastName AS STRING * 12
DIM FirstName AS STRING * 10
LastName ="Huntington-Ashley"
```

```
FirstName="Claire"
PRINT"12345678901234567890123"
PRINT FirstName;LastName
END
```

程序的输出有如下的形式：

```
12345678901234567890123
Claire          Huntington-A
```

如果将本例中的两个字符串变量改为可变长度的，则本程序的输出形式变为：

```
12345678901234567890123
ClaireHuntington-Ashley
```

第二节　字符串的合并与比较

字符串的合并与比较，是两种最基本的运算。关于字符串的其他许多运算，是通过字符串函数实现的。下一节将专门介绍字符串函数。

两个或几个字符串，可用"+"号进行合并操作，使之成为一个字符串，而且在两个字符串的合并处，没有空格。

例 10.2.1　有三个字符串，分别为 IBM，Personal 和Computer，现将它们合并为一个字符串，要求字符串之间有空格。

程序举例如下：

```
company $ ="IBM"
type $ ="Personal"
FullName $ =company $ +" "+type $ +" "+"Computer"
PRINT FullName $
END
```

合并的结果，有如下的形式：

```
IBM Personal Computer
```

由于固定长度字符串是从左边取，在后边往往剩有空格，所以，当合并时，在字符串之间，会有多出的空格。

例 10.2.2 固定长度字符串的合并。

```
TYPE NameType
        First AS STRING * 10
        Last AS STRING * 12
END TYPE
DIM NameRecord AS NameType
NameRecord. First="Ed"
NameRecord. Last="Feldstein"
PRINT"12345678901234567890"
wholeName $ =NameRecord. First+NameRecord. Last
PRINT WholeName $
END
```

程序的输出如下：

```
12345678901234567890
Ed          Feldstein
```

在 Quick BASIC 中有 DATE $ 和 TIME $ 两个语句，用来设置系统的当前日期和时间(24 小时制)。日期和时间是以字符串的形式给 DATE $ 和 TIME $ 赋值的。这两个语句的格式是：

DATE $ =字符串表达式 1

TIME $ =字符串表达式 2

字符串表达式 1 的格式可以有下列形式：

"mm-dd-YY"

"mm-dd-YYYY"

"mm/dd/YY"

"mm/dd/YYYY"

这里 mm 是月，dd 是日，YY 和 YYYY 是两位或四位数字的年。

字符串表达式 2 可以有如下的格式：

“hh”设置小时，分和秒缺省为零；

“hh：mm”设置小时和分，秒缺省为零；

“hh：mm：ss”设置小时、分和秒。

DATE$函数和TIME$函数，可以以字符串的形式返回当前日期和时间。

例 10.2.3　下面的程序由用户输入当前日期和时间，通过DATE$和TIME$两个语句将用户输入的日期和时间赋给系统。

程序如下：

```
PRINT“Enter the date below.”
INPUT“    Month: ”,   Month$
INPUT“    Day:   ”,   Day$
INPUT“    Year:  ”,   Year$
DATE$=Month$+“/”+Day$+“/”+Year$
PRINT“Enter the time below.”
INPUT“    Hour;  ”,   Hour$
INPUT“    Minute;”,   Minute$
INPUT“    Second;”,   Second$
TIME$=Hour$+“:”+Minute$+“:”+Second$
END
```

例如，运行上述程序时，输入了如下数据：

```
Enter the date below.
    Month: 9
    Day:   15
    Year:  1991
Enter the time below.
    Hour:  23
    Minute: 59
    Second: 50
```

系统的日期和时间被设置后，用DATE$函数和TIME$函

数，随时可以得到当前的日期和时间。例如，用下列 PRINT 语句，

```
PRINT DATE$
PRINT TIME$
```

可以得到如下的结果：

```
09-16-1991
00:08:24
```

字符串的比较是通过下列关系符实现的：

<>，=，<，>，<=，>=

一个字符的大小，由它的 ASCII 码值的大小决定，如英文大写字母 A—Z 的 ASCII 码是 65—90，英文小写字母 a—z 的 ASCII 码是 97—122，由此可知，“B”>“A”，“A”<“a”都是正确的。

例 10.2.4 将 6 个小写英文字母 x，f，e，g，s，a 按 a—z 的顺序排列并打印出来。

程序和输出结果如下：

```
DATA 6
DATA x,f,e,g,s,a
READ n%
FOR i%=1 TO n%
    READ x$(i%)
NEXT i%
FOR i%=1 TO n%-1
    FOR j%=i%+1 TO n%
        IF x$(i%)>x$(j%)THEN
           SWAP x$(i%),x$(j%)
        END IF
    NEXT j%
NEXT i%
FOR i%=1 TO n%
    PRINT x$(i%) +"  ";
```

```
NEXT i%
END
```

a e f g s x

在比较两个字符串时，BASIC 要看字符串中每个字符的 ASCII 码的值，从两个字符串的第一个字符比起，根据比较结果，确定字符串的大小。若第一个字符相等，则继续比较第二个字符，依此类推，来判定字符串的大小。显然，下列关系式是正确的：

"abc"="abc"

"aaaa">"aaa"

"aab"<"aac"

"abcfb">"abcfab"

第三节　字符串函数与语句

对于字符串的比较复杂的处理，主要是靠字符串函数实现。本节将介绍 Quick BASIC 提供的一些常用的字符串函数。Quick BASIC 中还有为数不多的，用于字符串处理的语句，将随字符串函数一并介绍。

一、字符串的搜索

在许多情况下，需要在一个字符串中查找是否含有另一个给定的字符串。字符串函数 INSTR 能够进行这种查找，其格式如下：

INSTR([start,]string 1,string 2)

其中自变量 start 用于指定起始查找的位置。它必须在 1－32767 的范围内。如果省略自变量 start，则从 string 1 的第一个字符开始查找。

string1 是被查找的字符串。

string2 是要查找的字符串。

string1 和 string2 可以是字符串变量、字符串表达式或者是字

符串常数。

当 INSTR 函数找到所要查找的字符串时，它返回到 string1 中第一次出现 string2 的第一个字符的位置。

例 10.3.1 给定 string1 $ 和 string2 $，用 INSTR 函数找 string2 $ 在 string1 $ 中的位置。

程序如下：

```
String1 $ ="A line of text with 37 letters in it."
String2 $ ="letters"
PRINT"          1         2         3         4"
PRINT"12345678901234567890123456789012345678901234567890"
PRINT String1 $
PRINT String2 $
PRINT INSTR(String1 $ ,String2 $ )
END
```

这个程序给出如下的结果：

```
          1         2         3         4
12345678901234567890123456789012345678901234567890
A line of text with 37 letters in it.
letters
24
```

例10.3.2 在一个字符串中查找给定字符串出现的次数和在字符串中的位置。例如，在字符串"What part of China are you from?"中查找有多少个字符串"a"，并给出 a 在字符串中的位置。

这可以通过循环调用函数 INSTR 来实现。

图10.3.1是完成本例题的程序框图。

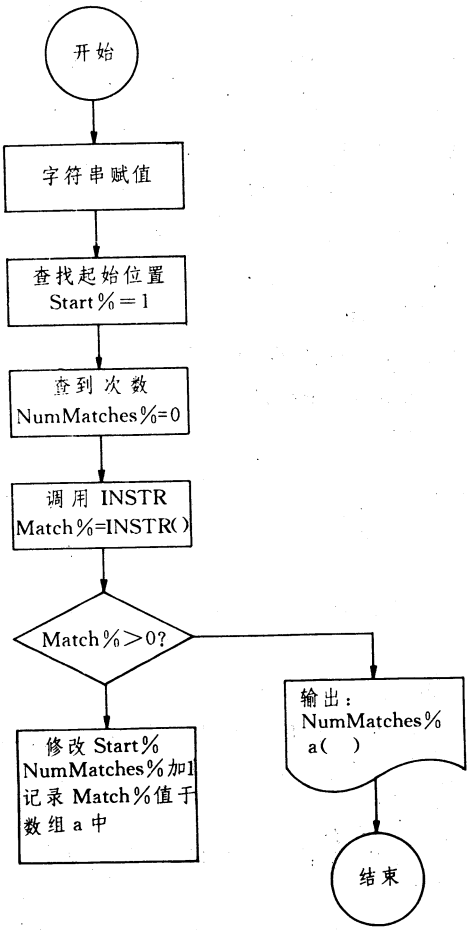


图10.3.1 例10.3.2程序流程图

根据框图写出程序如下：

```
DIM a(0 TO 20)AS INTEGER
```

```
String1 $ ="What part of China are you from?"
Start% =1
NumMatches% =0
DO
    Match% =INSTR(Start% ,String1 $ ,"a")
    IF Match% >0 THEN
       Start% =Match% +1
       NumMatches% =NumMatches% +1
       a(NumMatches% ) =Match%
    END IF
LOOP WHILE Match% >0
PRINT NumMatches% ;"Matches:"
FOR i=1 TO NumMatches%
    PRINT a(i);
NEXT i
PRINT
END
```

程序给出如下的结果：

```
4  Matches:
3  7  18  20
```

如果我们只需了解字符串的长度(字符数)，则可用函数 LEN。这个函数的格式是：

```
LEN(String)
```

自变量 String 可以是字符串变量、字符串表达式或字符串常数。

函数的功能是返回字符串的长度。另外函数 LEN 还能给出变量所需要的内存字节数，这对确定文件记录的大小是很有用的。LEN 函数用于确定变量字节数时的格式为：

```
LEN(variable)
```

自变量 variable 为变量名，它可以是各种 BASIC 类型的变量。

例10.3.3 应用 LEN 函数，确定一个自定义类型变量字节数，一个双精度变量的字节数和一个字符串的长度。

程序举例如下：

```
TYPE EmpRec
    EmpName AS STRING * 2
    Num       AS INTEGER
END TYPE
DIM a AS EmpRec
DIM b AS DOUBLE,c AS STRING
c="A string."
PRINT "EmpRec:";LEN(a)
PRINT "Double:";LEN(b)
PRINT "String c:";LEN(c)
END
```

程序输出：

```
EmpRec:4
Double:8
String c:9
```

二、部分字符串的截取

在字符串的处理中，常需要截取字符串的某一部分，下面介绍这方面的字符串函数。

函数 LTRIM$ 能够将字符串的前导空格移去，只取其空格后的字符串。这时，字符串一定以非空格字符开始。而函数 RTRIM$ 则能取字符串尾端(右侧)的空格移去后的字符串。这样处理后的字符串，一定以非空格字符结束。

上述两个函数的格式是：

LTRIM$ (Stringexpression)

RTRIM $ (Stringexpression)

其中自变量(Stringexpression)可以是任何字符串表达式，包括字符串变量和常数。

函数 LEFT $ 可从字符串左边起取其 n 个字符，而函数 RIGHT $ 则从字符串右边起取其 n 个字符。这两个函数有如下的格式：

LEFT $ (Stringexpression,n)

RIGHT $ (Stringexpression,n)

其中自变量 Stringexpression 可以是任何字符串变量，包括字符串变量和常数。

自变量 n 是一个在0－32767之间的数值表达式。它表示要从由 Stringexpression 给定的字符串中截取的字符数。如果 n 大于字符串长度，取字符串的全部字符。如果 n＝0，则给一个长度为零的空字符串。

例10.3.4 有字符串“ Computer Personal ”，现需将该字符串首、尾部分的空格去掉，然后将字符串中的两个字前后顺序颠倒过来输出。

首先用 LTRIM $ 和 RTRIM $ 两个函数移去其头、尾部分的空格。

其次，用 INSTR 函数，找到两个字之间空格的位置，用 LEN 函数求出字符串的长度。

然后，用 LEFT $ 函数和 RIGHT $ 两个函数，把字符串分离成两个字符串：“Computer”和“Personal”。

最后，按题目要求的顺序输出这两个字符串，同时，在两个字符串中间，加入一个空格。程序既输出了改变前的字符串，也输出了改变后的字符串。

程序和输出如下：

```
LonStr $ =“ Computer Personal”
PRINT LonStr $ :PRINT
```

```
LonStr $ =LTRIM $ (LonStr $ )
LonStr $ =RTRIM $ (LonStr $ )
Poss% =INSTR(LonStr $ ," ")
Length% =LEN(LonStr $ )
Fir $ =LEFT $ (LonStr $ ,Poss% -1)
Sec $ =RIGHT $ (Lonstr $ ,Length% -Poss% )
PRINT Sec $ ;" ";Fir $
END
```

```
        Computer Personal
Personal Computer
```

在字符串处理中，另一个重要的字符串函数是：

MID $ (stringexpression,start[,length])

这个函数的功能是：从 stringexpression 给定的字符串的第 start 个字符开始，截取长度为 length 个字符。自变量 start 和 length 必须在1到32767范围之内。如果略去自变量 length，则取从 start 往右的全部字符。如果从 start 往右的字符数小于 length，也是取从 start 往右的全部字符。如果自变量 start 大于 stringexpression 的字符数，则函数取一空字符串。

例如函数

MID $ ("QuickBASIC",6)

给出的字符串是 BASIC。

前面说过，字符串的比较，是逐个字符进行比较的。怎样才能从字符串中逐个取出字符呢？显然，MID $ 函数可以帮助我们解决这个问题。

例10.3.5 将一系列字符串按由小到大的顺序排列起来。

应用前面学习过的排序算法，对字符串的字符逐个比较，便可完成排序。但是，每个字符串的长度不同，因此，在进行字符串比较之前，还应先求出字符串的长度，根据字符串的长度进行比较。

程序框图如图10.3.2所示。

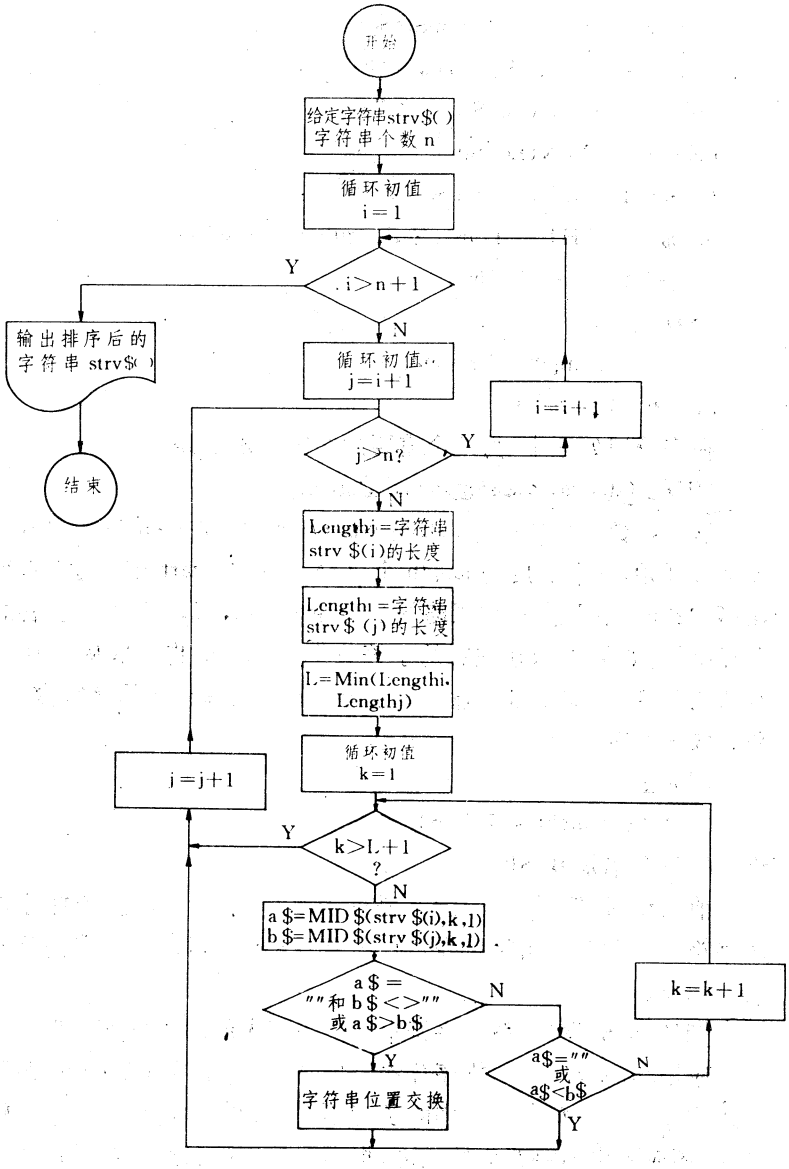


图 10.3.2　例10.3.5程序框图

下面是按照框图写出的程序：

```
DATA 7
DATA abcdefg,abcfb,abcfab,xa,abcd,abc,abcde
READ n
FOR i=1 TO n
    READ strv$(i)
NEXT i
FOR i=1 TO n-1
    FOR j=i+1 TO n
        (Lengthi=LEN(strv$(i))
        (Lengthj=LEN(strv$(j))
        IF Lengthi>Lengthj THEN
           L=Lengthj
        ELSE
           L=Lengthi
        END IF
        FOR k=1 TO L+1
            a$=MID$(strv$(i),k,1)
            b$=MID$(strv$(j),k,1)
            IF b$=" "AND a$<>" "OR a$>b$
               THEN
               SWAP strv$(i),strv$(j)
               EXIT FOR
            ELSEIF a$=" "OR a$<b$ THEN
               EXIT FOR
            END IF
        NEXT k
    NEXT j
NEXT i
```

```
FOR i=1 TO n
    PRINT strv$(i)
NEXT i
END
```

程序的输出如下：

```
abc
abcd
abcde
abcdefg
abcfab
abcfb
xa
```

三、字符串的修改

下面介绍两个函数和三个语句，它们能对字符串进行某种改变。

函数 LCASE$ 能把大写英文字母改变为小写英文字母，而函数 UCASE$ 能把小写英文字母改变为大写英文字母。这两个函数的格式为：

LCASE$ (string expression)

UCASE$ (string expression)

自变量 string expression 可以是字符串常数、字符串变量或字符串表达式。

语句 MID$ 可以用一个字符串来取代另一个字符串中的某一部分字符。语句 MID$ 的格式为：

MID$ (stringvariable, start[, length]) = stringexpression

其中 stringvariable 是被修改的字符串变量。start 是数值表达式，它给出被修改变量中被取代的起始位置。length 是数值表达式，它表示被取代的字符数。stringexpression 是用于取代的字符串

表达式，字符串变量或字符串常数。

length 是任选项，当它被省略时，用全部 stringexpression 进行取代。无论是否给出 length，取代的字符决不会超出被修改字符串变量的长度。

对于固定长度字符串，语句 RSET 和 LSET 可在赋值时，将字符串向左边对齐和向右边对齐。语句的格式为：

LSET Stringvariable＝stringexpression

RSET Stringvariable＝stringexpression

其中 stringvariable 是被赋值的字符串变量，stringexpression 是所赋的值的字符串表达式。

例10.3.6 将字符串“China，Beijing”修改为“CHINA，SHANGHAI”，并使它的最后一个字符位于第20个字符的位置上。

下边的程序，先按一般情况打印出原始的字符串，然后，用 MID $ 语句把 Beijing 改为 Shanghai，用 UCASE $ 实现由小写到大写的变换，最后，用 LSET 语句使字符串向左边对齐。为使字符和左边第20个字符位置对齐，把字符串定义为长度为20的定长字符串。

程序和输出如下：

```
DIM test AS STRING
DIM test1 AS STRING * 20
test＝“China，Beijing.”
PRINT“12345678901234567890”
PRINT test
a $ ＝“Shanghai”
MID $ (test，6，8)＝a $
test $ ＝UCASE $ (test)
RSET test1＝test
PRINT test1
END
```

```
12345678901234567890
China ,Beijing.
      CHINA,SHANGHAI
```

四、字符串、数与 ASCII 码的转换

BASIC 不允许把字符串赋给数值变量，也不允许把数值赋给字符串变量。然而，我们可以通过函数 STR $，把数转换为字符串，通过函数 VAL，把由数字符组成的字符串转换为数。

这两个函数的格式是

STR $ (numeric-expression)

VAL(stringexpression)

numericexpression 是数值表达式。STR $ 函数的功能是把数值表达式的值作为字符串，如果数值表达式是正数，则字符串含有引导空格，即在字符串之前，加一空格，作为字符串的一部分。

stringexpression 是数字字符串表达式。VAL 函数的功能是取数字字符串的数字值，使其为数字量，如果数字字符串的第一个字符不是数字，则 VAL 函数不能进行字符串—数字的转换。如果数字字符串的开头是空格，则忽略这些空格，然后取字符串的数字部分，直到遇到非数字字符。

例10.3.7 将二进制正整数转换为十进制数。

首先，用函数 STR $ 将二进制数转换为字符串 BinaryNum $，并用函数 LTRIM $ 消去引导空格，然后，用 MID $ 函数取出字符串中的每一位二进制数符 Digit $，用 VAL 函数将其转换为数字，以便通过运算，计算出相应的十进制数 Decimal。

程序如下：

```
INPUT"Binary number=",BinaryNum
BinaryNum $ =LTRIM $ (STR $ (BinaryNum))
Length=LEN(BinaryNum $ )
FOR k=1 TO Length
    digit $ =MID $ (BinaryNum $ ,k,1)
    Decimal=2 * Decimal+VAL(digit $ )
```

```
NEXT k
PRINT"Decimal Number=";Decimal
END
```

下面是程序的一次运行举例：

```
Binary number=101101
Decimal Number=45
```

另一类字符串与数字之间的转换，是字符与其相应的 ASCII 码之间的转换，这种转换是通过下面的两个函数实现的：

ASC(stringexpression)

CHR $ (numericalexpression)

函数 ASC 的功能是返回字符串表达式 stringexpression 中的第一个字符的 ASCII 码值，如遇到空串，在程序运行时，会产生错误信息(Illegal function call)。

函数 CHR $ 的功能是返回数字表达式 numericalexpression 所代表的 ASCII 码的字符，自变量的值必须在0—255的范围内。

我们知道，字符串常数要用双引号括起来。这就产生一个问题，如何使字符串本身含有双引号呢?CHR $ 函数可以解决这个问题。例如：

```
Msg $ =CHR $ (34)+"Quoted string"+CHR $ (34)
```

这里34是双引号的 ASCII 码。字符串 Msg $ 的首尾都有一个双引号。

例 10.3.8 编码程序

这里所说的编码，是指将任意一字符串，按给定的规律，变换成一个新字符串。

编码规律如下：

按字符串中字符由左往右的顺序，逐个将每一个字符的 ASCII 码值加＋5，然后以这个新的 ASCII 码所对应的字符，作为编码后的相应字符。如果新的 ASCII 码大于255，则取其与255之差作为新的 ASCII 码。

程序流程图如图10.3.3所示。由流程图可以看出，我们先用 ASC 函数获得字符的 ASCII 码并将其值改变，然后，再用 CHR $ 函数找到对应新 ASCII 码值的字符，最后，用合并运算，把新字符连

接成字符串，这个新的字符串，就是编码后的字符串。

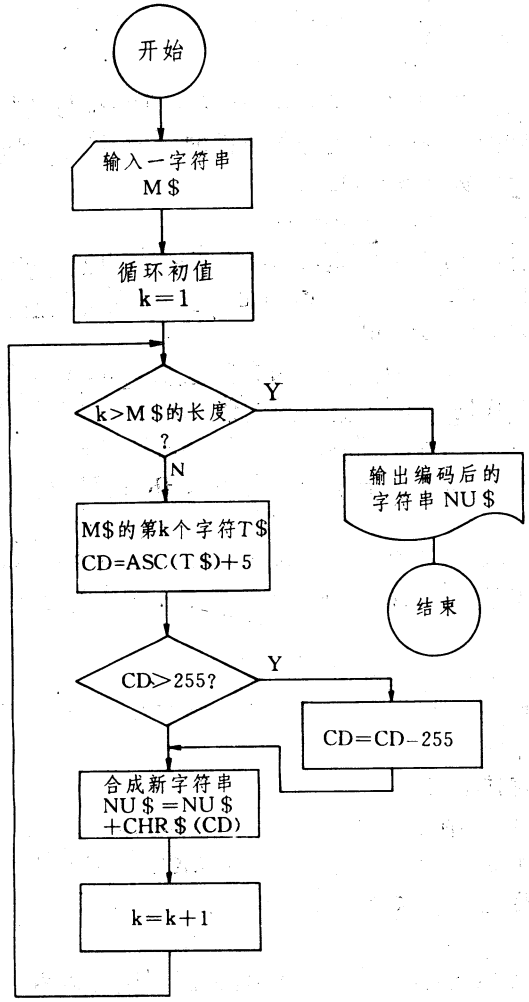


图 10.3.3　例10.3.8程序框图

程序如下：

```
INPUT"Enter message:",m$
FOR k=1 TO LEN(m$)
    t$=MID$(m$,k,1)
    cd=ASC(t$)+5
    IF cd>255 THEN cd=cd-255
    nu$=nu$+CHR$(cd)
NEXT k
PRINT "The coded mesage is:";nu$
END
```

当给程序输入字符串"computer"时，得到编码后的字符串是"htruzyjw"。

小　结

本章讨论了字符串的处理问题，首先介绍了字符串常数和字符串变量的概念，然后介绍了可变长度和固定长度两种类型的字符串。

字符串的处理是通过字符串加法(合并)运算、比较运算及字符串函数实现的。因此，学习字符串的处理，主要是掌握好字符串的两种运算和字符串函数的应用。

习　题

10-1　什么是可变长度字符串，什么是固定长度字符串？它们是怎样定义的？

10-2　下面程序的输出是什么？

```
A=12:B=13
C=(A+B)/100
A$=STR$(C)
```

```
A$=A$+"%"
PRINT A$
END
```

10-3　编写一程序，其功能是将一字符串中的字符，以完全相反的次序组成一个新字符串，保持原来的字符串不受破坏。

10-4　在 DATA 语句中有下列数据：

ABC,SCD,AXY,PQR,AFF

试编写一程序，将以 A 开头的字符串打印出来。

10-5　例10.3.8是将一个字符串按给定规则编码。试编写一程序，完成相反的过程，即将被编码的字符串，还原成编码前的字符串(称为解码)。

1、填空(10-20)

2、选择1

3、

4、

5、

第十一章　数据文件

文件是软件中的一个重要概念，也是程序设计的一个重点。按照文件的内容，有程序文件和数据文件。本章讲述数据文件的使用。

当程序要处理的数据很多时(如在事务管理中)，使用数据文件，几乎是唯一可取的方法。

本章首先阐述文件的概念并说明为什么要使用文件。接着分别介绍 Quick BASIC 中两种主要数据文件——顺序文件和随机文件的建立和读取方法。

除了上述两种文件外，在本章的最后，还将介绍二进制文件和设备文件。

学完本章应掌握：

(1) 生成新数据文件；

(2) 打开已存在的文件并读取其内容；

(3) 改变已存在数据文件的内容；

(4) 增加新的数据到已存在的数据文件中。

第一节　文件概述

“文件”一词已多次使用，例如，我们把存储在磁盘中的 BASIC 源程序称为文件。一般地说，文件是指存放在外存储器(如磁盘、磁带等)上的一些信息的集合。文件通常分为程序文件和数据文件。前者的信息是程序，后者的信息是数据。本章讨论的是数据文件。一些计算机的外设，如键盘、显示器、打印机等，也看作是文件，因而有时称它们为设备文件。

数据文件有两个重要特性：第一，组成一个文件的每个可被直接访问的单元，称为记录（注意，这不同于前面讲过的记录类型数据），一个文件的所有记录，必须是相同的数据类型；第二，数据文件中的记录是按一定顺序排列的，我们必须根据记录在文件中的位置访问这些记录。上述两个特性，进一步描述了数据文件的概念。

为什么要使用文件呢？或者说，在什么情况下使用文件呢？首先，操作系统以及 BASIC 的编译系统等，要对使用它的任何用户的任何数据进行加工处理，产生相应的结果，如程序的存、取、编译等，所有这些，只能是以文件的形式进行。其次，一个程序可能要用到数量很大的原始数据，这在事务管理，如工资管理，人事管理，财务管理等程序中是常见的事，把这些数量可观的数据以文件的形式保存在磁盘或其他外存中，可给数据的使用和管理带来很大的方便。如果一些数据是供程序使用的，那么就更应该把这些数据作成独立的文件。最后，一个程序可能要输出大量的数据，这些数据可以输出至屏幕上显示或输出至打印机打印出来。如果需要保存这些数据并供今后多次使用，那么就应以文件的形式存储这些数据。

从以上的说明可以看出，文件的使用是非常重要的，在某些情况下，不用文件就难以解决所面临的问题。

我们说过，数据文件是由一些“记录”组成的，每个记录又可分为若干个“域”，每个域中存放着记录的最小信息单元。对数据文件的存取，是以记录为单位进行的。

对数据文件的操作，主要是：

(1) 建立新数据文件；

(2) 读取已存在文件的内容；

(3) 增加新的信息到已存在的文件中；

(4) 改变已存在文件的内容。

上述四项操作，也就是文件的生成、使用和更新。

文件根据存储和读取(访问)磁盘数据文件方法的不同,分为顺序文件和随机文件。

顺序文件中的每个记录是以其输入的先后次序存放的,从磁盘读出时,也只能按记录号从头往后顺序地读,不能从顺序文件中直接存或取某一个指定的记录。

随机文件的存或取,可以直接根据给定的记录号进行。

顺序文件就像磁带,为了在磁带上找到某一首歌,必须从头开始走带,直到找到想找的歌为止。而随机文件象唱片,如果想要听唱片上的某首指定的乐曲,可以把唱臂上的唱针直接放在所要听的乐曲上。

第二节　顺序文件

一、顺序文件的打开与关闭

在建立、读取、修改或增加数据到文件之前,必须先把文件打开(OPEN)。打开文件的目的是:第一,确定所要处理的文件名;第二,指出进行何种操作,是建立新文件、读取文件还是增加数据等;第三,在文件与内存之间建立数据传送通道。为此,打开文件时在内存中设置一个与该文件相对应的数据缓冲区,并分配给它一个号码(1 到 225),称为文件号或缓冲区号。

可以同时打开多个文件,每个文件通过自己的数据缓冲区,在内存与文件之间传送数据。可见,打开一个文件,就使这个文件与对应的数据缓冲区通过文件号联系起来了,也就是说,可以对文件进行读写了。

打开文件是用 OPEN 语句实现的。OPEN 语句的格式如下:

OPEN file FOR model AS #filenum [LEN=reclen]

语句中 file 是文件名,可包含磁盘驱动器名和路径名。

model 是文件存取方式,可有以下三种选择:

(1) OUTPUT　指定为顺序输出方式(写入磁盘)；

(2) INPUT　指定为顺序输入方式(读磁盘)；

(3) APPEND　指定为顺序输出方式，可向文件尾增加新记录。

filenum 是文件号，它是 1 到 225 的一个整数表达式。

reclen 是记录长度，也是一个整数表达式，当省略此项时，其缺省记录长度为 512 个字节。reclen 的值不能超过 32767 个字节。

OPEN 语句举例：

OPEN "Price. Dat" FOR OUTPUT AS #1

如果磁盘上不存在 Price. Dat 文件，则这个语句的功能是建立并打开文件，使记录可以顺序地(从第一个记录开始)被写入到文件中。

如果磁盘上已存在文件 Price. Dat，则这个语句的功能是打开这个文件，新记录可以写入文件，但原有的记录将全部丢失。

OPEN "Price. Dat" FOR APPEND AS #1

这个语句打开已存在的文件 Price. Dat，使其中的记录可以被读出。

OPEN "Price. Dat" FOR APPEND AS #1

这个语句打开已存在的文件 Price. Dat，使新的记录可以增加到文件的末尾处，而以前的记录继续保留在文件中。如果给定的文件名 Price. Dat 在当前目录中不存在，则建立这个文件，如同在 OUTPUT 模式下建立新文件一样。

当文件使用完毕后，要关闭文件。关闭文件有两个重要作用：第一，如果缓冲区内有待存入文件的数据，在关闭文件时，缓冲区的内容将写入文件，从而防止了数据的丢失；第二，关闭文件的结果，释放了打开文件时与该文件相关的文件号和缓冲区，这意味着对该文件的读或写已不可能，释放出来的文件号，可以重新用在其他 OPEN 语句中。

关闭文件是由 CLOSE 语句实现的。该语句有如下的格式：

CLOSE [一组文件号]

CLOSE 后面要写上所要关闭的文件号，文件号之间用逗号分开。如果省略了文件号，则关闭所有打开的文件。

除了 CLOSE 语句，执行语句 END，CLEAR，RESET 等，也将关闭打开的文件。

最后，我们还要指出，OPEN 语句的另一种格式，也可被用户选用，其格式如下：

OPEN mode2，#filenum，file [，reclen]

其中 mode2 可以是下列符号之一：

I——指定为顺序输出方式（第一种格式的 INPUT）；

O——指定为顺序输出方式（第一种格式的 OUTPUT）；

A——指定为顺序输出方式，可在文件尾增加新记录（第一种格式的 APPEND）。

语句格式中的其他各项，均与第一种格式的含义相同。

二、顺序文件中的记录格式

顺序文件是 ASCII 码（文本）文件，因此，可用任何文字处理系统查看和修改它。

顺序文件中的记录是以回车—换行结束的一个文本行。文件中的各个记录，可以有不同的长度。

每个记录分成若干个域，所谓域，就是各个记录中按同样数据类型次序重复出现的数据，例如，每个记录含有 3 个域。但是，不同记录的相应域，可有不同的长度。

存储在记录域内的数据类型，决定了域的起始和结束位置（即域的界限）。对于字符串变量，下列情况标志着域与域之间的界限或记录的结束：

（1）以双引号开始遇到下一个双引号；

（2）不以双引号开始但遇到逗号；

（3）遇到回车—换行符号。

对于数值变量，下面的情况则标志一个域的结束：

(1) 逗号；

(2) 一个或多个空格；

(3) 回车换行。

图 11.2.1 是一个顺序文件的几个记录的示意图，在这个图中，表明了顺序文件中记录和域的划分。

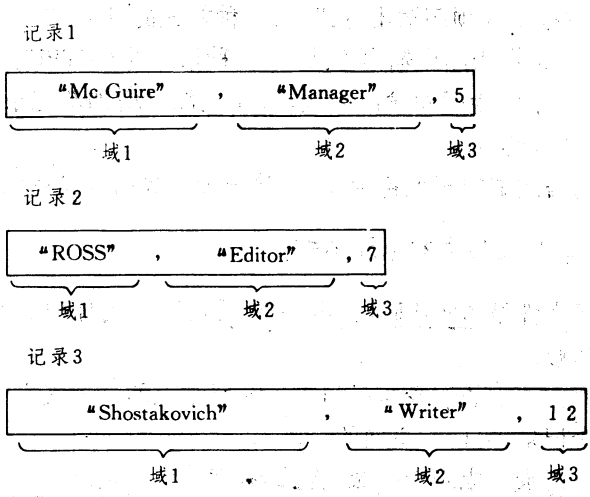


图 11.2.1 顺序文件的记录和域

三、顺序文件的建立和向文件增加数据

建立一个顺序文件，包含以下几个步骤：

(1) 用 OUTPUT 或 APPEND 模式的 OPEN 语句打开文件；

(2) 用 WRITE #语句把记录写入文件；

(3) 用 CLOSE 语句关闭文件。

用于把数据(记录)写到顺序文件上的 WRITE 语句的格式为：

WRITE #filenum，expressionlist

filenum 是相应于 OPEN 语句中分配的文件号，expressionlist 是一组表达式，表达式之间用逗号分隔。这些表达式是写到顺序文件的数据内容。如果缺省了 expressionlist，则把一个空行写到文件中。

用 WRITE 语句写入顺序文件的记录格式如图 11.2.1 所示。图 11.2.1 表明，记录的各域是用逗号分隔的，字符串数据是用双引号括起来的，因此，各域的界限是分明的。

例 11.2.1 将表 11.2.1 的英制——公制换算表建成一个顺序文件，供换算时使用，试编写建立此磁盘文件的程序。

表 11.2.1

英制单位	公制当量
1 英寸（IN）	2.54001 厘米（CM）
1 英里（MI）	1.60935 千米（KM）
1 英亩（ACRE）	4046.86 平方米（SQ.M）
1 英立方寸（CU.IN）	0.016387 升（LTR）
1（美）加仑（GAL）	3.785 升（LTR）
1（液体）夸脱（LIQ.QT）	0.9463 升（LTR）
1 磅（LB）	0.45359 千克（kG）

令文件名为 Data，则可用模式为 OUTPUT 的 OPEN 语句建立此文件，如

OPEN "b：Data" FOR OUTPUT AS #1

也可以用 APPEND 模式的 OPEN 语句建立，因为在当前的 B 盘上没有建立过 Data 文件。

表 11.2.1 的内容用 INPUT 语句输入。输入的内容为单位变换，用变量 Unit$ 表示，例如，对表的第一行，Unit$ 的值可表示为字符串"IN—>CM"。输入的第二项为当量，用变量 Factor 表示。这两个数据就构成一个记录。整个文件共七个记录。

每个记录的 Unit$ 和 Factor 值，用 Write 语句写入 Data 文件。用一个 DO-LOOP 循环语句可以写入任意个记录。

程序如下：

OPEN "b：Data" FOR APPEND AS #1

```
DO
    INPUT "Unit and Factor:", Unit$, Factor
    IF Unit$ = "q" THEN EXIT DO
    WRITE #1, Unit$, Factor
LOOP
CLOSE
END
```

由于这个程序使用了 APPEND 模式，因此，还可以用它向文件的末尾增添记录，或者将表 11.2.1 的内容分成几次输入。

四、读取顺序文件中的数据

顺序文件中的数据可以读出，供查阅或处理(如计算、修改、删除等)之用。

读顺序文件中的数据，首先必须用 INPUT 模式的 OPEN 语句，将文件打开。然后，用语句 INPUT # 读文件，该语句每次读一个记录，并把记录中每个域的值分别赋给相应的变量。

INPUT # 语句的格式如下：

```
INPUT #filenum, variablelist
```

其中 filenum 是打开文件时分配的文件号，variablelist 是被赋值的变量名，变量的类型必须与记录中各个域数据的类型一致。

因为是顺序文件，所以读取记录是按存入时的记录号，从头到尾顺序地读，每读出一个记录，记录的指针便自动指向下一个记录。

文件结束函数 EOF 检测是否已读完最后一个记录，当未读完最后一个记录时，该函数的值为 0(假)；读完最后一个记录时，该函数值为 −1(真)。所以，这个函数可以帮助我们了解是否已读完全部记录。这个函数的格式是：

```
EOF (filenum)
```

其中 filenum 是在 OPEN 语句中赋予的文件号。

例 11.2.2 读取在例 11.2.1 中建立的顺序文件 Data 并将其显示在屏幕上。

我们可以用一循环程序将记录逐个读入并显示。由于可能不知道文件中有多少个记录，因此，用 EOF 函数作循环条件。程序如下：

```
OPEN "b:Data" FOR INPUT AS #1
DO UNTIL EOF (1)
    INPUT #1, Name$, Number
    PRINT Name$, Number
LOOP
CLOSE #1
END
```

五、顺序文件的更新

数据文件中的数据，由于情况的变化，需要进行某种改变，称为文件的更新。文件的更新主要包括以下内容：

(1) 添加或插入新数据；

(2) 修改旧数据；

(3) 删除部分旧数据等。

根据顺序文件的特点，处理任何一个记录，都要依次把记录读出来，再进行修改、删除、插入等工作，变化后的数据，依次写入一个临时文件中。当所有记录都被处理并写入临时文件后，这个临时文件就是更新的文件了。这时，我们可以用 KILL 语句从磁盘上删除旧文件，然后，用 NAME 语句，把临时文件的文件名改为原来文件的名字。至此，文件的更新就完成了。

语句 KILL 和 NAME 的格式为：

```
KILL  文件名
NAME  旧文件名  AS  新文件名
```

在使用这两个语句时，相关的文件必须是关闭着的。如果文件处于打开状态，程序运行时会产生错误信息：

File already open。

下面通过一个简单的例子说明更新数据文件的方法。

例 11.2.3 试编写一个程序，其功能是修改文件中的记录。

设已有例 11.2.1 建立的顺序文件 Data.

令 Data 为文件＃1，临时文件 temp 为文件＃2。程序首先要以 INPUT 模式打开文件 Data，以 OUTPUT 模式打开（建立）文件 temp，然后才能进行相应的读或写操作。程序框图如图11.2.2所示。

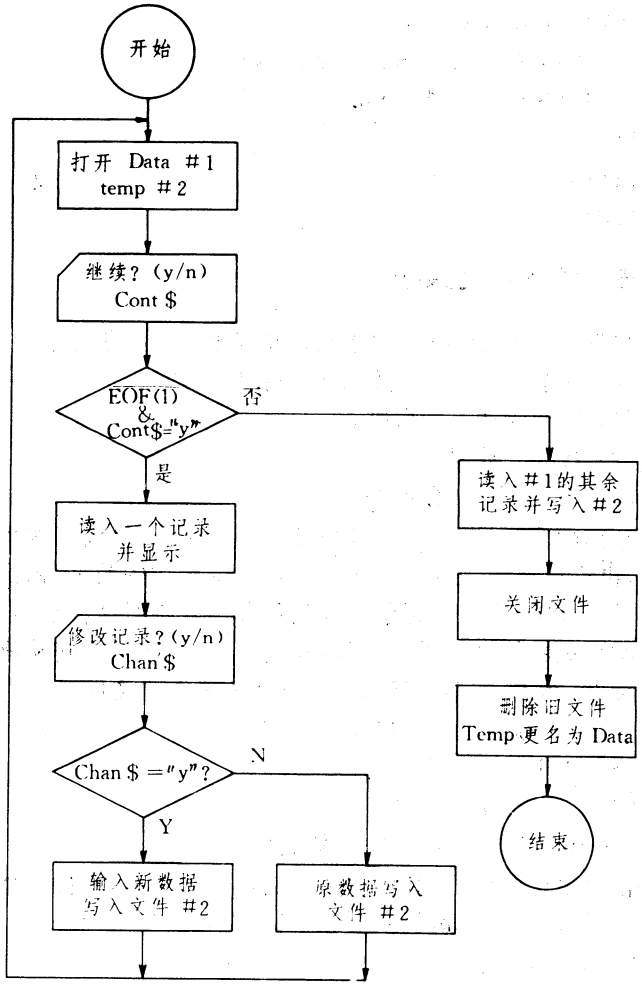


图 11.2.2　例 11.2.3 程序框图

下面是按照框图编写的程序：

```
OPEN "b:Data" FOR INPUT AS #1
OPEN "b:temp" FOR OUTPUT AS #2
DO
    INPUT "Continue? (y/n)", Cont$
    IF EOF(1) OR Cont$ ="n" THEN
       EXIT DO
    ELSE
        INPUT #1, Unit$, Factor
        PRINT Unit$, Factor
        INPUT "Change?(y/n)", Chan$
        SELECT CASE Chan$
            CASE "y"
                INPUT "New Data:", Unit$, Factor
                WRITE #2, Unit$, Factor
            CASE "n"
                WRITE #2, Unit$, Factor
        END SELECT
    END IF
LOOP
DO UNTIL EOF(1)
    INPUT #1, Unit$, Factor
    WRITE #2, Unit$, Factor
LOOP
CLOSE
KILL "b:Data"
NAME "b:temp" AS "b:Data"
END
```

按照例11.2.3的程序，读者不难写出具有更多功能的文件更

新程序。

数据文件的基本用途是为程序提供数据，或者存储程序产生的数据，例如，有了例11.2.1建立的数据文件，我们就可以编制一个程序，应用该数据文件，实现英制与公制之间的数量转换。这个任务留给读者去作。

第三节　随机文件

一、随机文件的打开与关闭

和顺序文件一样，随机文件在建立或读、写时，需先用 OPEN 语句打开文件；文件使用完毕，要用 CLOSE 语句将其关闭。

对于随机文件，OPEN 语句的格式为：

OPEN file [FOR RANDOM] AS # filenum [LEN=reclen]

其中 file 是要打开的文件名；FOR RANDOM 表示打开的是随机文件，在这种模式下，既可读文件中的记录，也可以往文件写入数据，此项可以省略；filenum 是文件号；reclen 是记录的长度（字节数），此项也可缺省，其缺省值为128个字节，reclen 的值不能超过32767个字节。

打开随机文件语句的另一种格式是：

OPEN R, # filenum, file [,reclen]

CLOSE 语句对顺序文件和随机文件完全一样，这里就不重复了。

二、随机文件记录的格式

随机文件中的数据也是以记录为单元进行读、写，每个记录又分成一定数目的域。与顺序文件不同，随机文件的记录是固定长度的，记录中各个域的长度也是固定的。

随机文件中存储的数字，使用的是压缩二进制格式，在这种格式下，一个整型数占2个字节，长整型数和单精度数占4个字节，双精度数占8个字节。由此可知，随机文件与顺序文件不同，它不是

ASCII 码文件，字符串是用 ASCII 码存储的，一个字符占一个字节。

由于随机文件的记录和域是固定长度的，因此，就不需要分隔各个域和记录的分隔符号(如逗号、双引号、回车——换行等)。

既然记录域的长度是一定的，因此，在程序之首，先要定义记录。定义记录可用定义记录类型数据的 TYPE/END TYPE 语句来实现。

例如，我们要建立某个班级学生的数学和英语成绩的文件，显然，每个记录应分为三个域：学生姓名，数学分数和英语分数。设姓名域是长度为20个字符的定长字符串，数学分数为整型数，英语分数为整型数，则记录可定义如下：

```
TYPE StudentRec
    StudName AS STRING * 20
    Math     AS INTEGER
    English  AS INTEGER
END TYPE
```

由上述记录定义不难知道，记录长度为24个字节，其中第一域长20字节，第二域长2字节，第三域长2字节。这个记录结构如图11.3.1所示。

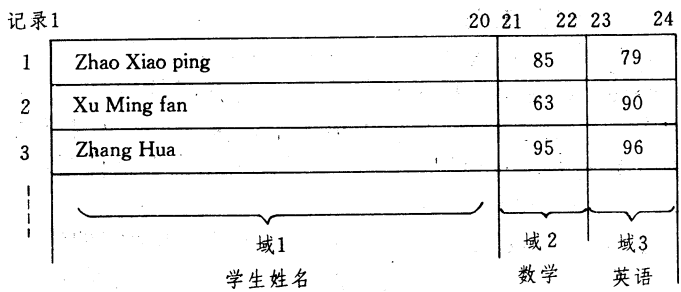


图11.3.1 随机文件记录的结构

用上述方法定义的记录结构，其长度还可以用函数 LEN 得到。这个函数的格式是：

LEN(字符串表达式)

LEN(变量)

LEN 函数的功能是给出字符串的字符数或变量所需的字节数。

对于上面定义的记录,其长度可表示为(设有 DIM Student AS StudentRec):

LEN(Student)

用函数 LOF 还可得到整个文件的长度(字节数)。函数的格式是

LOF(文件号)

已知文件的总长度和记录的长度,不难计算出文件的记录数:

Record Number=LOF(文件号)/LEN(记录变量)

三、随机文件的建立

建立随机文件一般要经过以下四个步骤:

(1) 用 TYPE/END TYPE 定义文件记录的结构;

(2) 用 OPEN 语句打开文件;

(3) 将要存入文件的数据,以记录为单位,用 PUT 语句写入指定的记录号;重复本条所述操作,直至写入全部数据;

(4) 关闭文件。

PUT 语句的格式是:

PUT # filenumber[,[recordnumber][,variable]]

filenumber 是在 OPEN 语句中文件打开时的编号。recordnumber 是写入记录的编号,文件中第一个记录是1号,如果省略记录号,那么就写入下一个记录(下一个记录号由内部指针指示),允许的最大记录号是$2^{31}-1$,即2147483647。

例 11.3.1 建立一个关于学生数学和英语成绩的随机文件,每个记录包含三个内容:学生姓名,数学分数(整数)和英语分数(整数)。

按照上述建立随机文件的四个步骤,可写出如下的程序:

```
TYPE StudentData
    StudName AS STRING * 20
```

```
    Math        AS INTEGER
    English     AS INTEGER
END TYPE
DIM StudentRec AS StudentData
OPEN "b:Student" FOR RANDOM AS #1 LEN=LEN(StudentRec)
PRINT "Enter <end> to end."
DO
   INPUT "Name:", StudentRec.StudName
   IF StudentRec.StudName<>"end" THEN
      INPUT "Math:", StudentRec.Math
      INPUT "English:", StudentRec.English
      PUT #1,, StudentRec
   ELSE
      EXIT DO
   END IF
LOOP
CLOSE
END
```

四、随机文件的读取

与顺序文件不同，随机文件可以根据给定的记录号，读取指定的记录。读取随机文件记录由 GET 语句来实现。GET 语句的格式如下：

GET #filenumber [,[recordnumber][,variable]]

语句中各项的意义与 PUT 语句相同。

读取随机文件的步骤与建立随机文件类似，也有四个步骤：

(1) 定义记录结构；

(2) 打开文件；

(3) 读取记录；

(4) 关闭文件。

对于例11.3.1建立的随机文件，如若读取记录2，其相应的

GET 语句应为

```
GET #1,2, StudentRec
```

如要读取文件的全部记录，可用一循环程序来实现，但不能像例11.2.2那样，使用 DO UNTIL EOF(1)-LOOP 循环，因为，对于随机文件，只有当 GET 语句不能读出一个完整的记录时，函数 EOF 才返回一个“真”值，因此，如果采用上述循环，必定多读一次。在这种情况下，可用 FOR-NEXT 语句循环读出文件记录，循环次数就等于文件中的记录数。在11.3.2节中，我们也曾给出计算文件中记录数的方法。

例11.3.2 编写一个程序，使之读出并显示例11.3.1中建立的文件中的全部记录。

程序如下：

```
TYPE StudentData
    StudName AS STRING * 20
    Math     AS INTEGER
    English  AS INTEGER
END TYPE
DIM StudentRec AS StudentData
OPEN "b:Student" FOR RANDOM AS #1 LEN=LEN(StudentRec)
NumberOfRecord=LOF(1)/LEN(StudentRec)
FOR i=1 TO NumberOfRecord
    GET #1,, StudentRec
    PRINT StudentRec.StudName,
    PRINT StudentRec.Math,
    PRINT StudentRec.English
NEXT i
CLOSE
END
```

随机文件的修改比较简单，因为可以指定任何记录来修改，只要把新的数据写入指定记录，就完成了该记录数据的更新。

我们再来看一个应用文件的例子。

例11.3.3　设已建立好例11.3.1的数据文件，编一个计算数学和英语平均分数的程序。

首先给出框图，如图11.3.2所示。

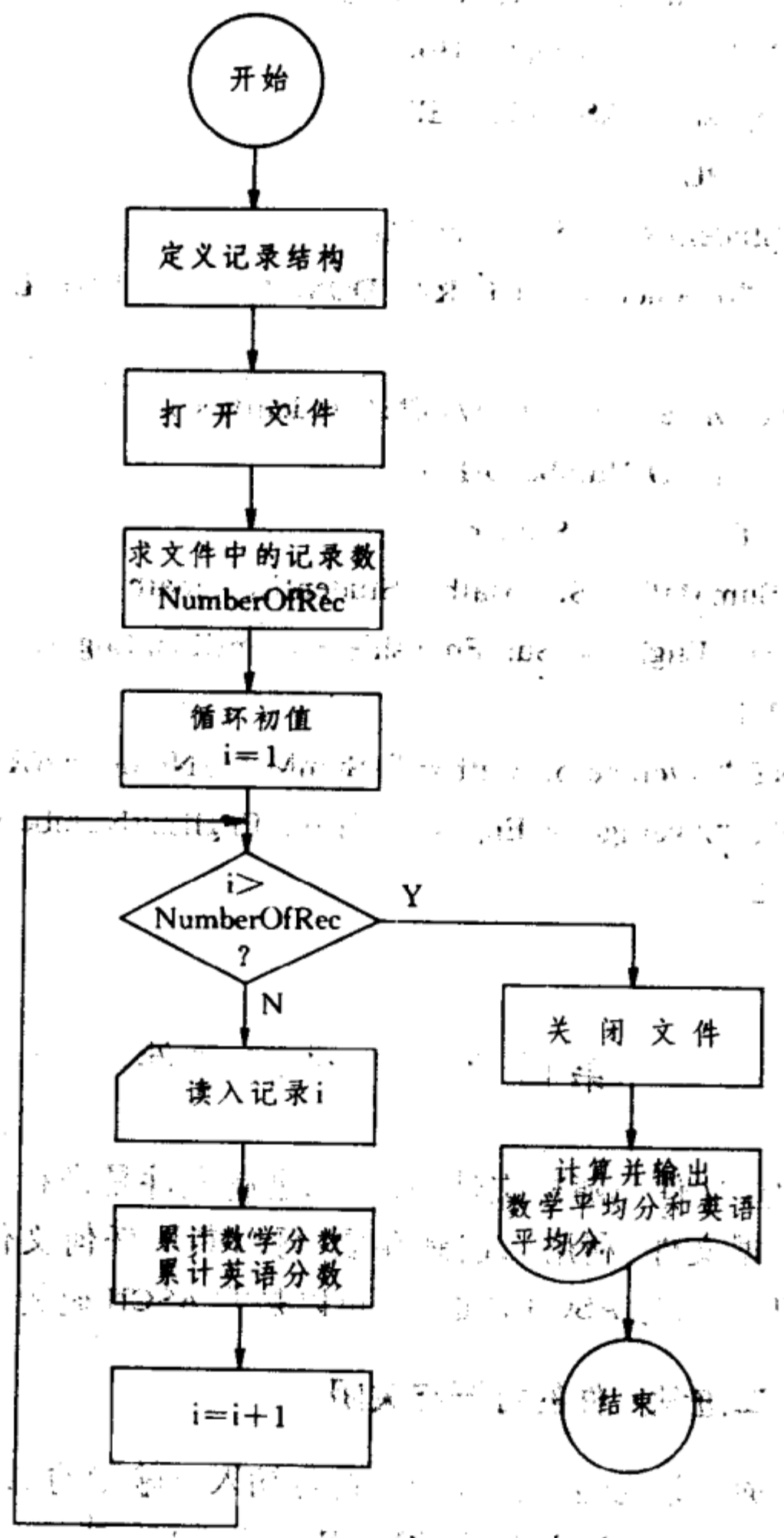


图11.3.2　例11.3.3程序框图

程序如下：

```
TYPE StudentData
    StudName AS STRING * 20
    Math     AS INTEGER
    English  AS INTEGER
END TYPE
DIM StudentRec AS StudentData
OPEN "b:Student" FOR RANDOM AS #1 LEN=LEN(StudentRec)
NumberOfRec=LOF(1)/LEN(StudentRec)
FOR i=1 TO NumberOfRec
    GET #1,i, StudentRec
    SumMath=SumMath+StudentRec. math
    SumEnglish=SumEnglish+StudentRec. English
NEXT i
PRINT "Average of Math=";SumMath/NumberOfRec
PRINT "Average of English=";SumEnglish/NumberOfRec
CLOSE
END
```

第四节　二进制文件

与顺序文件和随机文件不同，二进制文件是进行二进制存取的一种数据文件。利用二进制存取，可以获取任何文件的任意字节，文件既可以是 ASCII 码的，也可以是非 ASCII 码的。

一、二进制文件的打开与关闭

用下面的语句可以打开一个进行输入与输出的二进制文件：

OPEN 文件名 FOR BINARY AS #文件号

以二进制存取打开的文件，是作为字节序列来处理的，文件本身不必涉及记录，但可以把每个字节看作是一个记录来进行二进制存取，也可以把用户定义的各种类型变量作为记录，以二进制存取方式，读或写到文件中。

二进制文件的关闭语句与前面介绍的两种文件的关闭语句是一样的。

二、二进制存取

二进制文件的存取操作，与随机文件的存取操作是类似的，这主要表现在下述三个方面：

首先，用 OPEN 语句打开文件后，对文件既可以读，也可以写，不需在读和写之间进行切换。

其次，可以在打开的文件中，任意向前或向后移动文件的指针。

最后，可以用读、写随机文件的语句，读、写二进制存取文件。这两个语句是：

GET ＃文件号，[位置]，变量

PUT ＃文件号，[位置]，变量

语句中的“变量”可以是任何类型的，包括变长字符串和用户定义类型。“位置”指示下一条 GET 或 PUT 语句操作在文件的什么地方。文件的开头位置为1，即第一个字节的位置是1，第二个字节的位置是2，等等。

GET 语句从文件中读出的字节数，等于语句中“变量”的类型长度；同样，PUT 语句向文件写入的字节数，也是等于该语句中“变量”类型的长度。如整型变量为2个字节，单精度型变量为4个字节。由此可知，每执行一次 GET 或 PUT 语句，文件指针移动的距离，等于相应变量的字节数。

例11.4.1 将用随机数函数产生的五个随机数连同它们的序号存入名为“DataFile”的二进制文件中。

程序如下：

```
OPEN "DataFile" FOR BINARY AS #1
FOR i%=1 TO 5
    a=RND
    PUT #1,,i%
    PUT #1,,a
NEXT i%
CLOSE
END
```

PUT 语句是向以二进制方式打开的文件写入数据的唯一语句，因此，上面的程序是用 PUT 语句将数据写入"DataFile"文件的。当然，写操作前，先要用 OPEN 语句以二进制存取方式打开文件。为存入序号和随机数，在一个 PUT 语句中使用了一个整型变量，在另一个 PUT 语句中使用了单精度型变量。

例11.4.2 将例13.1.1中建立的二进制文件"DataFile"读出并显示在屏幕上。

程序如下：

```
OPEN "DataFile" FOR BINARY AS #1
number%=LOF(1)/6
FOR i%=1 TO number%
    GET #1,,a%
    GET #1,,b
    PRINT a%,b
NEXT i%
CLOSE
END
```

在这个程序中，采用循环语句显示文件的内容，每循环一次显示两个数据。循环次数是由下面的语句确定的：

```
number%=LOF(1)/6
```

程序的输出如下：

```
1            .7107346
2            .99058
3            .8523988
4            .3503776
5            4.363585E-02
```

二进制存取是其他 BASIC 及某些高级程序设计语言所不具备的功能，它为修改二进制文件提供了方便。

例11.4.3 编写程序，读取并显示二进制文件 Datafile 的各个字节，采用十六进制数的显示方式。

程序如下：

```
CLS
DIM a AS STRING * 1
OPEN "datafile" FOR BINARY AS #1
text$ =" "
FOR i% =1 TO 40
    GET #1,i%,a
    PRINT USING "\\";HEX$(ASC(a));
    IF (ASC(a)<32 OR ASC(a)>126) THEN a="."
    text$ =text$ +a
    IF i% MOD 8=0 THEN
       PRINT" ";text$
       text$ =" "
    END IF
NEXT i%
CLOSE #1
END
```

这个程序的输出形式是：每行显示8个字节，接着显示与这8个字节相对应的 ASCII 码(<126)的可打印字符。如果不能显示为这

样的字符，就用字符“.”代替。

程序给出如下的输出结果：

1	0	B4	F2	35	3F	2	0	....5?..
A7	96	7D	3F	3	0	CE	36	..}?...6
5A	3F	4	0	B2	64	B3	3E	Z?...d.〉
5	0	80	BB	32	3D	0	0	....2=..
0	0	0	0	0	0	0	0	.....

三、文件指针

在二进制文件中，可以把文件指针移到文件中任意的字节位置，从而实现对任何位置的文件读、写操作。文件指针的定位是通过下面的语句实现的：

SEEK ＃文件号，位置

“文件号”的含义同前，前面的符号“＃”可以省略。“位置”是一个数值表达式，它指示下一个要读、写的字节位置。“位置”的取值在1—($2^{31}-1$)范围内。

用 SEEK()函数可以得到文件指针的当前位置。这个函数的格式如下：

SEEK(文件号)

例11.4.4 编写程序，读取二进制文件“DataFile”的第三组数据(第5和第6个数据)。

我们知道，第1,3,5,…个数据是整型数，每个数占两个字节的长度；第2,4,6,…个数据是单精度型数，每个数占4个字节，所以，为读取文件中的第5个数据，应将文件指针的位置设置为13。读出一个数据后，指针会自动后移，后移的距离等于被读出数据的字节数。

程序如下：

```
OPEN “DataFile” FOR BINARY AS ＃1
SEEK ＃1,13
```

```
GET #1,,a%
PRINT a%,
GET #1,,b:
PRINT b
CLOSE
END
```

第五节　设备文件

Quick BASIC 支持设备的输入输出，也就是说，外部设备也可以作为文件打开，进行输入输出操作。Quick BASIC 所支持的设备文件如表11.5.1所示。

表11.5.1

设　备	名　称	I/O 方式
第一串行接口	COM1:	输入/输出
第二串行接口	COM2:	输入/输出
屏幕	CONS:	输出
键盘	KYBD:	输入
第一打印机	LPT1:	输出
第二打印机	LPT2:	输出
第三打印机	LPT3:	输出
屏幕	SCRN	输出

设备文件键盘(KYBD:)、屏幕(SCRN:)、打印机(LPTn:)和串行口(COMn:)通常叫作字符设备。字符设备文件可以像磁盘文件那样打开和使用，但是，字符设备文件的记录长度为1。

例11.5.1　编写程序，将指定磁盘文件输出至设备文件LPT1:和设备文件SCRN:。磁盘文件名是输出至LPT1:还是SCRN:，都通过INPUT语句的输入给定。

程序如下：

```
CLS
INPUT "File Name to output:",FilNam $
```

```
OPEN FilNam$ FOR INPUT AS #1
DO
    PRINT "Output to screen(s) or to printer(P):";
    choice$ =UCASE$ (INPUT$ (1))
LOOP WHILE choice$ <>"S" AND choice$ <>"P"
SELECT CASE choice$
    CASE "P"
        OPEN "LPT1:" FOR OUTPUT AS #2
    CASE "S"
        CLS
        OPEN "SCRN:" FOR OUTPUT AS #2
    END SELECT
WIDTH #2,80
DO UNTIL EOF(1)
    LINE INPUT #1,Dataline$
    PRINT #2,Dataline$
LOOP
CLOSE
END
```

在上面的程序中，首先要求用户输入需要读取的文件名，然后将该文件打开，接着让用户选择输出设备。输入字母"S"时，打开屏幕文件 SCRN:；输入字母"P"时，打开打印机文件 LPT1:。最后，程序逐行地读文件内容，并将这些内容送至打开的设备文件输出。

读取文件的内容使用 LINE INPUT 语句，输出使用 PRINT 语句。程序中语句

```
WIDTH #2,80
```

用来指定打开的#2设备文件输出的行宽为80个字符。

小　结

本章主要介绍文件的概念及数据文件的建立与应用。

首先讲述了什么是文件，为什么要使用文件。并讲述了程序文件和数据文件的区别及数据文件的应用。

按着存取方式的不同，Quick BASIC 的数据文件分为顺序文件、随机文件和二进制文件三大类。针对这三类文件，分别介绍了文件打开与关闭，文件的建立，分件的读取以及分件的更新等内容。

结合文件的学习，本章还介绍了下列8个语句：

OPEN　　CLOSE　　INPUT #

WRITE #　　PUT #　　GET #

KILL　　NAME

和三个函数

EOF　　LEN　　LOF

通过本章的学习，要求掌握文件的概念；有关语句和函数；能够建立和应用顺序文件和随机文件。

习　题

11-1　什么是文件？为什么要使用文件？

11-2　什么是顺序文件？什么是随机文件？

11-3　怎样建立和读取顺序文件？怎样建立和读取随机文件？

11-4　编写一个 BASIC 程序，要求：

(1) 将100个随机数存入顺序文件；

(2) 对数据进行求和及求平均值的计算；

(3) 将计算结果存入数据文件。

11-5　用表11.2.1的数据建立一个随机文件，然后再编写一个程序，其功能是根据用户给定的英制数据，计算出相应的公制数

据。

11-6　用顺序文件实现例11.3.1。

11-7　用顺序文件实现例11.3.2。

11-8　用顺序文件实现例11.3.3。

第十二章　图　　形

Quick BASIC 具有很强的图形支持能力。本章学习实现图形功能的语句和函数，掌握编写绘制各种不同颜色的图形、曲线程序的方法。在这一章，将学到如何画线、画圆、画椭圆和画框等，并学习屏幕坐标定位、图形填充和颜色的使用，还将简单地介绍动画技术的基本知识。

在运行本章有关图形的例子时，要求计算机配有图形适配器，如单色显示与打印机适配器 MDPA，彩色图形适配器 CGA，增强性图形适配器 EGA，视频图形卡 VGA 和 IBM 多色图形阵列 MCGA 等，同时还要有一个图形显示器，它可以是单色的，也可以是彩色的。

本章的全部内容都是针对 IBMPC 及其兼容机讨论的。

第一节　屏幕显示

屏幕显示有两种方式：文本方式和图形方式，到目前为止，我们使用的全是文本方式。

在文本方式下，屏幕显示位置是用字符坐标系来描述的。这种坐标系以屏幕的左上角为坐标原点，水平方向为 x 轴，垂直方向为 y 轴，原点坐标为(1,1)。文本方式通常分为40列文本和80列文本，前者每行可显示40个字符，后者每行可显示80个字符。每屏共显示25行。

屏幕显示的列数可以是30,43,50或60，这取决于使用的显示适配器和显示模式。屏幕显示的行、列数可用 WIDTH 语句设定。该语句的格式为：

WIDTH 列,行

为了实现图形功能，屏幕应工作在图形方式下。屏幕显示模式用 SCREEN 语句设置，这个语句的用法在下一节介绍。

在图形方式中，屏幕上的每一个可以控制的单元不是字符而是“点”，通常称之为像元或像素，它是图形的基本元素。

屏幕上像素的个数叫作分辨率。例如，屏幕的水平方向有320个像素，垂直方向有200个像素，则该屏幕的分辨率为320×200。

图形方式中，像素在屏幕上的显示位置，用点坐标系来描述。坐标系的原点通常设在屏幕的左上角，与文本方式不同的是，它的原点坐标是(0,0)。BASIC 所以能在屏幕上画图，就在于它能控制图像在指定坐标处亮或灭，或改变其颜色。

在文本方式中，要显示的字符的位置，是用下面的语句指定：

LOCATE 行号，列号

而在图形方式中是不能使用这个语句的。

在图形方式下，仍可以显示文本字符，例如对于 CGA，中分辨率图形方式(320×200)对应40列文本，高分辨率图形方式(640×200)对应80列文本。

图12.1.1和图12.1.2分别给出80列文本方式显示屏幕和分辨率(640×200)的图形方式显示屏幕。

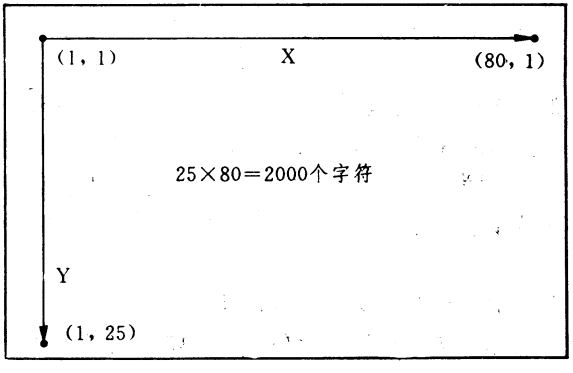


图12.1.1　文本方式屏幕

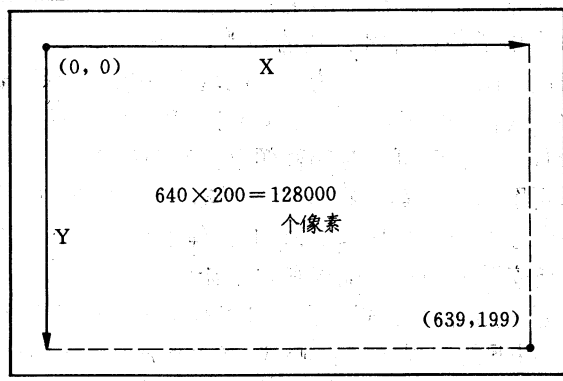


图12.1.2　图形方式屏幕

第二节　屏幕显示方式设置

为了在屏幕上画出图形，必须先设置屏幕显示方式，这个操作是用 SCREEN 语句实现的。

SCREEN 语句的格式如下：

SCREEN[方式][,[颜色]][,[当前页]][,[可见页]]

现在来说明语句中各参量的含义。

方式　是一个整型常量或表达式，它指明屏幕方式。后面将介绍各种适配器的屏幕方式。

颜色　是一个取值在0－255范围内的数值表达式。当该值为真(非0)时，禁止彩色，只显示黑白图像；当该值为假(0)时，显示彩色图像。当工作在“方式0”时，上述“颜色开关”的意义相反。在“方式”为2或2以上时，忽略“颜色开关”。

当前页　是一个数值表达式，表示输出显示信息时被写入的屏幕页号。页号的有效值与适配器有关。

可见页　是一个数值表达式，表示正在显示的屏幕页号。

SCREEN 语句的作用就在于选择屏幕方式，以适合显示器与

适配器的特定组合。SCREEN 共有10种方式，编号为0－2和7－13。对于不同的显示器与适配器的组合，屏幕方式的功能是不一样的。表12.2.1列出了各种屏幕方式的性能，表中的“中”、“高”、“增”和“极”分别代表图形的分辨率为中、高、增强和极高四种；表中的 C，E，V 和 M 分别代表 CGA、EGA、VGA 和 MCGA；“颜色/属性”栏中的数值，如16/4，表示可以对16种颜色指定四种属性。

考虑到图形程序都应在一种图形输出的模式支持下运行，在图形程序中应有一条 SCREEN 语句。具体使用哪种屏幕方式，要根据计算机所安装的图形适配器和显示器而定。

假设要使用屏幕方式1，但不知道所用硬件是否具备彩色/图形能力，可用下面的程序进行测试。这个程序使用了 Quick BASIC 的陷阱技术。这种程序的设计原理，将在下一章详细介绍。

表12.2.1

方式	文本格式	字符点阵	图形格式	颜色/属性	适配器
0	40×25 40×43 40×50 80×25 80×43 80×50	8×8 8×14 9×16		16/2 16/6	
1	40×25	8×8	320×200，中	16/4	C,E,V,M
2	80×25	8×8	640×200，高	16/2	C,E,V,M
7	40×25	8×8	320×200，中 多屏幕页	16/16	E,V
8	80×25	8×8	640×200，高 多屏幕页	16/16	E,V
9	80×25 80×43	8×14 8×8	640×350，增 多屏幕页	64/16 16/4	E,V
10	80×25 80×43	8×14 8×8	640×350，增 多屏幕页	9/4	E,V
11	80×30 80×60	8×16 8×8	640×480，极	256/2	V,M
12	80×30 80×60	8×16 8×8	640×480，极	256/16	V
13	40×25	8×8	320×200，中	256/256	V,M

程序如下：

```
CONST false=0:true=NOT false
ON ERROR GOTO Message
SCREEN 1
ON ERROR GOTO 0
IF NoGraphics THEN END
PRINT "Graphics OK!"
END
Message:
    PRINT "This program requires graphic capability."
    NoGraphics=true
    RESUME NEXT
```

如果用户只有单色显示器，没有图形适配器，则 SCREEN 语句产生错误，于是转到错误处理子程序 Message。

第三节　基本几何图形的绘制

在这一节我们介绍画点、直线、矩形框、圆、椭圆、弧和楔形图的方法。

一、画点

Quick BASIC 的画点功能是由 PSET 和 PRESET 两个语句实现的。语句格式为：

PSET [STEP](x,y)[,颜色号]

PRESET [STEP](x,y)[,颜色号]

语句的功能是在坐标为(x,y)的位置上画一点。

当不使用选择项 STEP 时，x 和 y 为屏幕的绝对坐标，即屏幕的左上角坐标为(0,0)。当使用选择项 STEP 时，则 x 和 y 为相对坐标，相对坐标原点为程序中上一次使用的坐标点。

选择项"颜色号"是所要画点的颜色属性，它是由后面将要介

绍的 COLOR 语句定义的当前配色器中的颜色。

PSET 和 PRESET 语句的功能基本相同，唯一的区别是，当参量“颜色号”省略时，PRESET 自动把该参量设置为0，即用背景色画点.这就相当于擦除了在(x,y)位置上已显示的点。在这种情况下，PSET(x,y)和 PRESET(x,y)的作用正好相反。

例12.3.1 编写程序，在 y=100的水平位置，从 x=100开始，间隔50，画9个点，并使这些点不断闪烁，当用户按任一键时，程序运行结束。

对一个指定的点，重复使用 PSET 和 PRESET 语句，就可产生点闪烁的效果，闪烁的速度可以用 PSET 和 PRESET 之间的延迟时间加以控制。

程序如下：

```
SCREEN 2      ' 640 * 200 resolution
PRINT "Press any key to end"
DO
   FOR x=100 TO 500 STEP 50
      PSET (x,100)
   NEXT x
   FOR x=0 TO 5000: NEXT x
   FOR x=100 TO 500 STEP 50
      PRESET (x,100)
   NEXT x
   FOR x=0 TO 5000: NEXT x
LOOP UNTIL INKEYY <> " "
END
```

下面再举一个例子，说明 PSET 和 PRESET 语句中选择项“STEP”的使用。

例12.3.2 编写程序，显示如图12.3.1所示的图形，要求图形显示后，稍停片刻即消失。

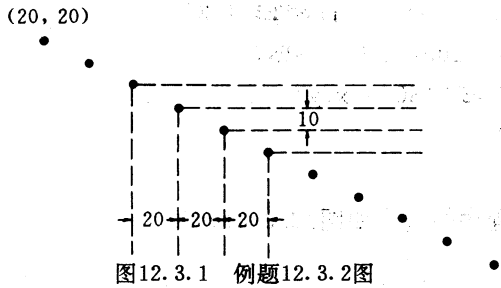


图12.3.1　例题12.3.2图

由图可知，相邻点坐标的 x 值增20，y 值增10。这样的点非常容易用相对坐标来表示，这时，各点(第一个点除外)的相对坐标均为(20,10)。

程序如下：

```
SCREEN 2      ' 640 * 200 resolution
PSET (20,20)
FOR x=1 TO 10
    PSET STEP (20,10)
NEXT x
FOR x=0 TO 15000:NEXT x
PRESET (20,20)
FOR x=1 TO 10
    PRESET STEP(20,10)
NEXT x
END
```

画点的重要应用之一，就是用来画曲线。

例12.3.3　编写程序，显示一条正弦曲线。

为了得到尽可能圆滑的曲线，采用屏幕方式11，这时的屏幕分辨率为640×480。

程序如下：

```
pi=3.14159
SCREEN 11
FOR x=0 TO 2 * pi STEP 1/300
    y=200+200 * SIN(x)
    PSET (80 * x,y)
NEXT x
END
```

这个程序的输出如图12.3.2所示。

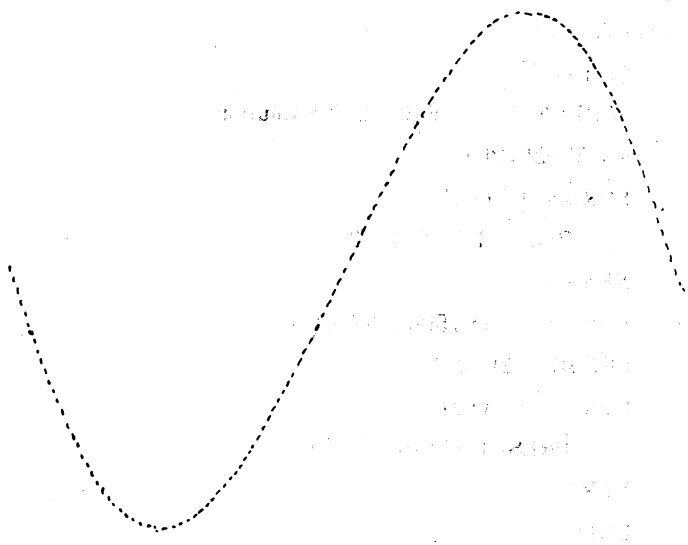


图12.3.2 例12.3.3程序的输出

二、画直线和矩形框

用上面介绍的画点方法，能画出曲线，也能画直线，如下面的

程序段，能画出一条水平直线：

```
FOR x=0 TO 200
    PSET (x,100)
NEXT
```

用类似的方法，也可以画矩形框。但是，Quick BASIC 提供了专门用于画直线和矩形框的语句。这个语句的格式为：

LINE [[STEP](x1,y1)]-[STEP](x2,y2)[,[颜色号][,B[F]][,类型]]

语句的功能是在屏幕上画直线和矩形框，其中(x1,y1)和(x2,y2)是起点和终点坐标，它们的先后次序无关紧要。如果语句中选用 STEP，则上述坐标为相对坐标。当第一个坐标点省略时，则直线从最后的参考点画至(x2,y2)点。下面是一些 LINE 语句的例子：

LINE(10,10)-(50,50)　从(10,10)到(50,50)画线

设最后参考点为(10,10)，则

LINE-(40,40)　　从(10,10)到(40,40)画线

LINE-STEP(40,40)　从(10,10)到(50,50)画线

LINE(20,20)-STEP(40,40) 从(20,20)到(60,60)画线

语句中"颜色号"的含义同 PSET 语句。

语句中选用参量 B 时，表示 LINE 语句画一矩形框，其对角点坐标为(x1,y1)和(x2,y2)。如果选用 BF，则画出矩形框后，还要用指定的颜色填充该矩形。

语句中的"类型"是一个16比特的整数，其取值范围为0-&HFFFF。当选择该项时，LINE 从左至右读出每个比特，如果一个比特为0，则不画点；如果一个比特为1，则画出一个点。因此，利用带有此选项的语句，可以画出特殊类型的线，如虚线、点划线等。

下面是几个应用 LINE 语句的例子。

例12.3.4　应用 LINE 语句和相对坐标，在屏幕方式2的图形模式下，画出如图12.3.3所示的图形。

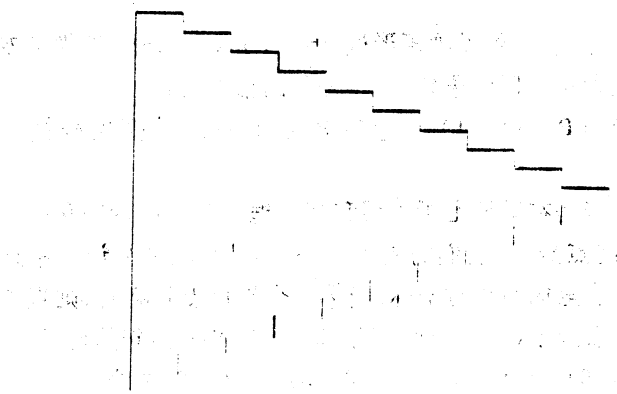


图12.3.3　例12.3.4的图形

程序如下：

```
SCREEN 2      ' 640 * 200 resolution
LINE STEP(-200,50)-STEP(0,-100)
FOR i%=1 TO 10
    LINE -STEP(30,0)
    LINE -STEP(0,5)
NEXT i%
FOR i%=1 TO 9
    LINE STEP(-30,0)-STEP(0,5)
NEXT i%
END
```

例12.3.5　编写程序，画一个正方形框和它的两条对角线。令正方形的边长为100个像素。

程序如下：

```
SCREEN 1
```

```
LINE (50,50)-(150,150),,B
LINE -STEP(-100,-100)
LINE (150,50)-(50,150)
END
```

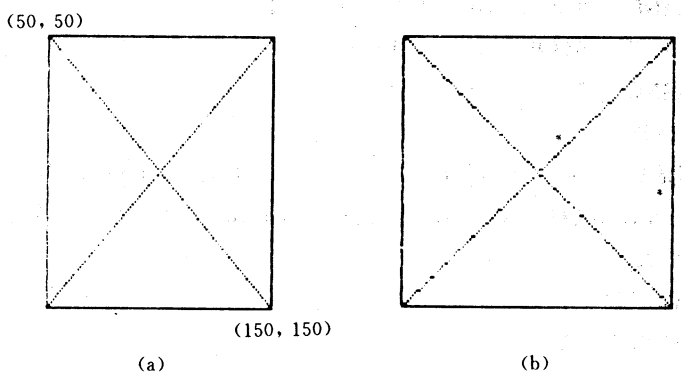


图12.3.4 例12.3.5的图形

程序的输出如图12.3.4(a)所示，显然，它不是一个正方形，其原因是，在单位屏幕长度内，水平方向的像素要比垂直方向的像素多。例如，在屏幕方式1的情况下，像素为320(水平)×200(垂直)。而屏幕的尺寸通常是4×3。所以，为了得到成比例的图形，如在本例中，要得到真正的正方形，必须考虑长宽比的关系。

如果显示器的长宽比是4:3，则应按下式计算图形的长宽比例：

$$\frac{4}{3}\times\frac{\text{y 轴像素数}}{\text{x 轴像素数}}$$

在 SCREEN 1方式下画图时，则有

$$\frac{4}{3}\times\frac{200}{320}=\frac{100}{120}=\frac{5}{6}$$

这表明，对应于垂直方向上的100个像素，相当于水平方向上的120个像素。这时两长度才相等。

根据以上的讨论，将上面的程序，作如下修改后，便可得到如图12.3.4(b)所示真正的正方形。

修改后的程序如下：

```
SCREEN 1
LINE (50,50)-STEP(120,100),,B
LINE -STEP(-120,-100)
LINE (170,50)-(50,150)
END
```

例12.3.6 利用LINE语句中“类型”参量值&hcccc，&hee00,,&hf0f0,&hf000,&h7788和&haa00产生6种不同的虚线和点划线。

程序如下：

```
SCREEN 2
DATA &hcccc,&hee00,&hf0f0
DATA &hf000,&h7788,&haa00
Row%=4:Column%=4
XLeft%=80:XRight%=500
y%=28
FOR i%=1 TO 6
    READ Style%
    LOCATE Row%,Column%
    PRINT HEX$(Style%)
    LINE (XLeft%,y%)-(XRight%,y%),,,Style%
    Row%=Row%+3
    y%=y%+24
NEXT i%
END
```

程序的输出如图12.3.5所示

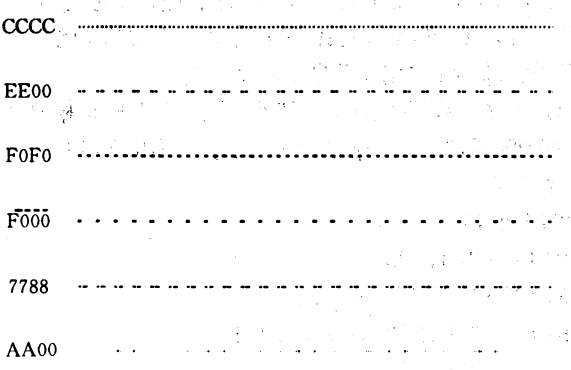


图12.3.5　例12.3.6的图形

三、画圆、椭圆和弧

Quick BASIC 提供了一个具有画圆、椭圆和弧功能的语句CIRCLE。该语句的格式为：

CIRCLE [STEP](x,y),半径[,[颜色号][,[起始角][,[终止角][,比率]]]]

语句格式中各参量的含义如下：

(1) (x,y)是圆、椭圆或弧的圆心坐标。

(2) "STEP"选项使(x,y)变为相对坐标。

(3) "半径"为当前坐标系下圆、椭圆(主轴方向)或弧的半径。

(4) 选项"颜色号"为所需颜色的属性。

(5) 选项"起始角"和"终止角"是以弧度为单位，用来确定一个圆弧的两个角度。角的弧度是按逆时针计算的。角度前冠以负号时，表示弧的端点与圆心相连。如果只给出终止角，则弧从2π画起；如果只给出起始角，则从起始角画到0。

(6) 选项"比率"是用来确定椭圆的y轴半径与x轴半径的比。缺省时，则画一圆。比率的缺省值为

$$\frac{4}{3}\times\frac{\text{屏幕 y 轴像素数}}{\text{屏幕 x 轴像素数}}=k$$

当画椭圆时，如果纵横比小于1，则以 x 轴半径作主轴半径；如果纵横比大于1，则以 y 轴半径作为主轴半径。这时语句一定含有“比率”选项。如果要想得到成比例的图形，还要在“比率”项中考虑到 k 值的问题，即应将“比率”的值扩大 k 倍。

例12.3.7 编写程序，令其画两个水平方向相切的、半径相等的圆；两个椭圆，其 x，y 轴的半径比分别为0.3和3。

程序如下：

```
SCREEN 11
CIRCLE (220,100),75
CIRCLE STEP(150,0),75
CIRCLE (220,250),60,,,,3/10
CIRCLE (370,250),60,,,,3
END
```

程序的输出如图12.3.6所示。

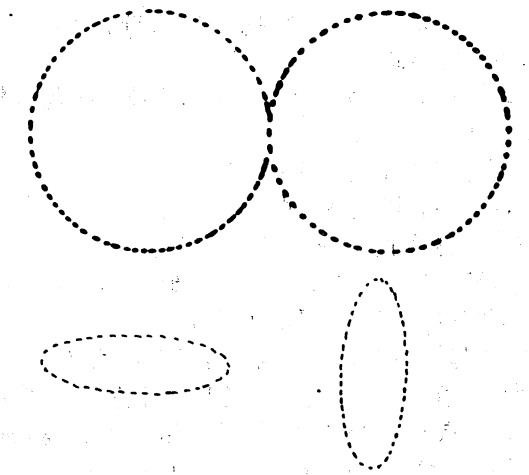


图12.3.6 例12.3.7程序的输出

例12.3.8 下面给出一个画圆弧的程序例子。

程序如下：

```
SCREEN 2
CLS
CONST pi=3.1415927#
StartAngle=0
PSET (320,60)
FOR radius%=100 TO 220 STEP 20
    EndAngle=StartAngle+(pi/2)
    CIRCLE (320,60),radius%,,StartAngle,EndAngle
    StartAngle=StartAngle+(pi/4)
NEXT radius%
END
```

由程序循环语句可以看出，这个程序画7个圆弧，这些圆弧都以一个共同的圆心画出，圆心坐标为(320,60)，各圆弧的角度都是π/2，各圆弧的起始位置相差π/4。第一个圆弧从0度开始。

程序的输出如图12.3.7所示。

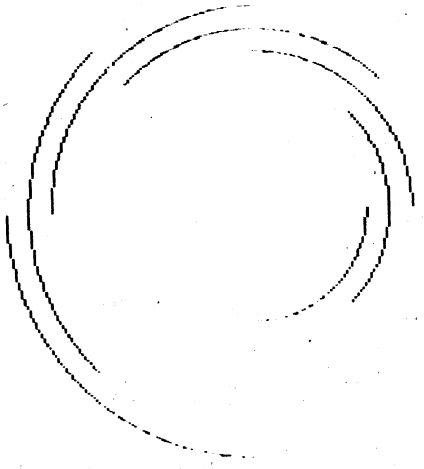


图12.3.7　例12.3.8程序的输出

例12.3.9 下面是画“楔形图”图形的程序：

```
SCREEN 2
CONST PI=3.141592653589#
CONST RADIUS=150
StartAngle=2.5
EndAngle=PI
CIRCLE (220,75),RADIUS,,-StartAngle,-EndAngle
SWAP StartAngle, EndAngle
CIRCLE STEP(70,10),RADIUS,,-StartAngle,-EndAngle
END
```

这个程序所画出的图形如图12.3.8所示。

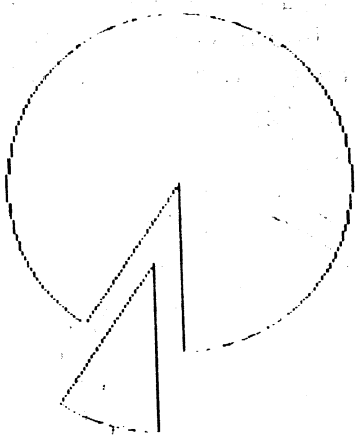


图12.3.8 例12.3.9程序的图形

第四节 连续画线

Quick BASIC 提供了一个图形宏语言，它定义了一套画图命令，这些命令由字符串组成。语句格式为：

DRAW 字符串

用字符串表示的命令包括：

1. 光标移动命令

用光标移动命令可以把光标向各方移动，同时画出直线。光标一次移动的距离单位，缺省单位为一个点(像素)。移动多少个单位，由命令中的自变量 n 给定。单位的大小，也可以自己定义。如果命令中没有给出 n 值，光标就移动一个单位的长度。每个命令都是从当前位置开始移动。程序开始运行时，当前作图位置缺省值为屏幕中心。光标移动命令有：

U[n] 向上移动 n 个单位

D[n] 向下移动 n 个单位

L[n] 向左移动 n 个单位

R[n] 向右移动 n 个单位

E[n] 向右上方移动 n 个单位

F[n] 向右下方移动 n 个单位

G[n] 向左上方移动 n 个光标

H[n] 向左下方移动 n 个光标

M x,y 绝对移动或相对移动。 x,y 为坐标。如果 x 或 y 前有“+”号或“−”号，为相对移动，x 和 y 的值被加到坐标上而成为坐标点，并把该点与当前位置连成一直线。如果在 x 或 y 前没有符号，则为绝对移动，即从当前坐标位置到坐标(x,y)之间画一直线。

2. 角度、颜色和比例因子命令

这些命令用于改变一个图形的显示输出，包括旋转角度，变化颜色和改变实际移动的距离：

An 设置旋转角 n。n 的取值为0，1，2和3，分别对应0°，90°，180°和270°。在标准的长宽比为4/3的显示屏幕上，将图形旋转90°或270°后，和旋转0°或180°所显示的图形，其尺寸是一样的。

TAn 旋转角度 n。n 可以在−360—360°之间取值。如果 n 是正数，则逆时针旋转；如果 n 是负数，则顺时针旋转。

Cn　把颜色号设置为 n。

Sn　设比例因子 n。n 的取值范围为1－255。用此比例因子乘以由光标移动命令给出的 n 值，就是实际移动的距离。

P 涂色，边界色　“涂色”是图形内部所涂的颜色，“边界色”是图形边界的颜色。

V　用逻辑坐标而不是物理坐标(用户定义的坐标系)执行后续的所有 DRAW 绘图命令。

3. 字符串子命令

X 字符串　执行子命令。这里的“字符串”一般是在 DRAW 语句之前赋值的字符串变量。设有字符串变量 A $(由绘图命令构成)，X 子命令有以下两种格式：

```
DRAW"XA $"
DRAW"X"+VARPTR $(A $)
```

两个语句的功能是一样的，但第一个语句只能在解释程序中运行，第二个语句既可以在解释程序中运行，也可以编译后运行。

下面通过一些例子来说明 DRAW 语句的具体使用方法。

例12.4.1　下面是画一个正方向和一个三角形的程序：

```
SCREEN 1
DRAW "C2"
DRAW "F60 L120 E60"
DRAW "BD30"
DRAW "P1,2"
PSET (135,80)
a $ ="R60 U50 L60 D50"
DRAW "X"+VARPTRY(A $)
END
```

第一个 DRAW 语句设置粉红色；第二个 DRAW 语句画三角形，起始坐标为 SCREEN 确定的屏幕中心；第三个 DRAW 语句将坐标点移至三角形内，做好给三角形涂色的准备；第四个 DRAW

语句将三角形涂为青色带粉红色的轮廓；最后一个 DRAW 语句，用子命令画一个正方形。

程序的输出如图12.4.1所示。

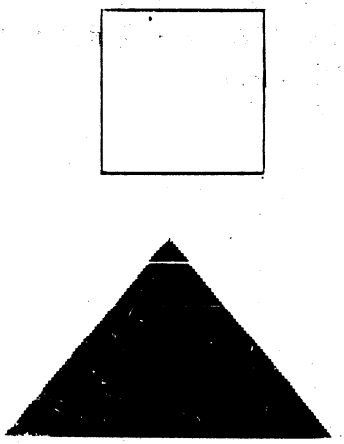


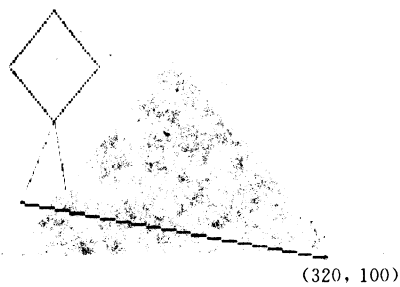
图12.4.1　例12.4.1程序的输出

例12.4.2　下面的程序用于说明 DRAW 语句的移动命令 M 和设置旋转角度命令 A 的作用：

```
SCREEN 2
DRAW "M 50,80"
DRAW "M 80,50"
'DRAW "A2"
DRAW "M+40,-20"
DRAW "M-40,-20"
DRAW "M-40,+20"
DRAW "M+40,+20"
'DRAW "A3"
```

```
x$="+10":y$="+25"
DRAW "M"+x$+","+y$
END
```

程序中第一段的两个 DRAW 语句画两条直线；第二段的四个 DRAW 语句画一个菱形；第三段的一个 DRAW 语句又画一条直线。程序中用于旋转图形的两个语句：DRAW"A2"和 DRAW"A3"不用时，程序的输出如图12.4.2(a)所示。当使用这两个语句时，程序的显示图形如图12.4.2(b)所示。



(a)



(b)

图12.4.2　例12.4.2程序的屏幕显示

例 12.4.3 下面的程序是使用 TA 命令画一个从中心点向各方向画辅射线的图形：

```
SCREEN 1
FOR i=1 TO 360 STER 10
    DRAW "TA="+VARPTR$(i)+"nu50"
NEXT i
END
```

程序的屏幕显示图形如图 12.4.3 所示。

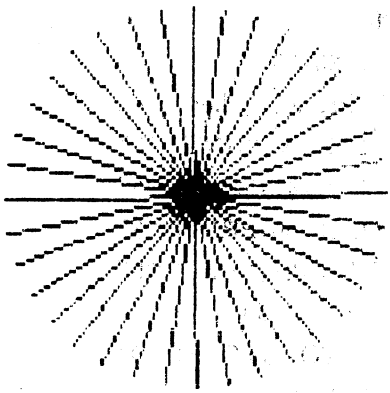


图 12.4.3 例 12.4.3 程序的屏幕显示

例 14.4.4 本例给出一个在屏幕上显示时钟的程序。程序中用一个子程序 Face 来画时钟，主程序每分钟调用子程序一次，以便改变时钟的指针。主程序还显示一个数字式时钟。时钟的时间取自计算机的时钟系统，这是用函数 TIME$ 实现的。程序如下：

```
DECLARE SUB Face (Min$)
SCREEN 2
```

```
DO
    CLS
    Min$ =MID$ (TIME$ ,4,2)
    Face Min$
    DO
        LOCATE 2,37
        PRINT TIME$
        Test$ =INKEY$
    LOOP WHILE Min$ = MID$ (TIME$ , 4, 2) AND
    Test$ =" "
LOOP WHILE Test$ =" "
END
SUB  Face(Min$ )
        LOCATE 23,30
        PRINT "Preaa any key to end."
        LOCATE 5,40:PRINT "12"
        LOCATE 21,41:PRINT "6"
        LOCATE 13,60:PRINT "3"
        LOCATE 13,21:PRINT "9"
        CIRCLE (320,100),175
        FOR i=0 TO 360 STEP 30
            DRAW "TA="+VARPTR$ (i)
            DRAW "bu69 u4"
            PSET(320,100)
        NEXT i
        Hr=VAL(TIME$ )
        Min=VAL(Min$ )
        Little=360-(30 * Hr+Min/2)
        Big=360-(6 * Min)
```

```
        DRAW "TA="+VARPTR$(Little)+"NU40"
        DRAW "TA="+VARPTR$(Big)+"NU70"
    END SUB
```

第五节　图形颜色的处理

图形可以使用的颜色，既和系统的图形适配器及显示器有关，也与屏幕的显示模式有关。对于图形适配器 CGA 来说，在 SCREEN 1 方式下，分辨率为 320×200，前景有四种颜色，背景有十六种颜色。而在 SCREEN 2 方式下，分辨率为 640×200，前景和背景各有一种颜色。

我们看到，颜色和清晰度在两种屏幕方式下是互相折衷的。为了得到图形颜色，就不得不放弃一定的清晰度。

在前几节介绍的一些语句，如 PSET，PRESET，LINE，CIRCLE，DRAW 等，都含有用于设置颜色的参量。在 SCREEN 1 方式下，这些参量的值可为 0，1，2 或 3，每一个值都称为“属性”，表示不同的颜色。

除此以外，Quick BASIC 还提供了用于控制前景和背景颜色、填充图案等手段，为设计彩色图形程序提供了方便。

一、改变前景和背景颜色

1. 屏幕颜色设置语句 COLOR

COLOR 语句的功能是设置屏幕的前景、背景及边框的颜色。语句格式为：

(1)COLOR[前景色][,[背景色][,边框色]]

(2)COLOR[背景色][,调色板]

(3)COLOR[前景色][,背景色]

(4)COLOR[前景色]

格式(1)只能在 SCREEN 0，即文本方式下使用。用它可以改

变前景、背景及边框颜色。

前景色的取值范围为0—31，表示字符的颜色号。前景色的编号如表12.5.1所示。

表12.5.1

编号	颜色	编号	颜色
0	黑	8	灰(或黑)
1	蓝	9	亮蓝
2	绿	10	亮绿
3	青	11	亮青
4	红	12	亮红
5	洋红	13	亮洋红
6	棕(或暗黄)	14	亮黄
7	白(或灰)	15	亮白(或亮灰)

将上述颜色编号加16，则给出上述相应颜色的闪烁字符。

背景色是一个数值表达式，取值范围为0—7，表示背景的颜色号。背景颜色不能闪烁。

边框色是一个数值表达式，取值范围为0—15，表示边框的颜色号。EGA、VGA和MCGA不支持这个“边框色”参量。

在COLOR语句中，允许“前景色”和“背景色”取相同值，此时字符在屏幕上不可见。

COLOR语句中的各参量全是任选的，如全部省略，则使用原来参量的值。

格式(2)只能在SCREEN 1，即中分辨图形方式下使用。

参量“背景色”是数值表达式，取值范围为0—15，表示背景颜色的编号。编号的意义与文本方式（即SCREEN 0）相同。

参量“调色板”是数值表达式，取值范围为0—255。当取值为偶数时，选择0号调色板；当取值为奇数时，选择1号调色板。在两个调色板中，各有4个不同的前景颜色，其编号如表12.5.2所示。

当进入SCREEN1时，自动选用1号调色板，背景颜色为黑

色。

表 12.5.2

调色板 / 前景色	0 号调色板	1 号调色板
0	当前背景色	当前背景色
1	绿	青
2	红	洋红
3	棕（黄）	白

格式（3）用于屏幕方式 7—10。在这些方式中，不能指定边框颜色。图形“背景色”的颜色号，必须在当前屏幕方式下有效颜色范围内。“前景色”是一个属性号，而“背景色”是颜色号。

格式（4）用于屏幕方式 SCREEN 12 和 13。在这种方式下，不能指定背景色。“前景色”是前景颜色的属性，该属性必须是在相应屏幕方式规定的正确范围内。

在 SCREEN 11 和 SCREEN 2 两种方式下，不能使用 COLOR 语句，否则，会产生错误信息“Illegal function call”。

例 14.5.1 下面的程序通过改变背景颜色和三个正方形的三种不同颜色，演示了所有 16 种背景颜色和两个调色板的组合所给出的屏幕颜色。程序如下：

```
SCREEN 1
BSC$ =CHR$ (27)
FOR ColorVal=1 TO 3
    LINE (x, y) -STEP (60, 50), ColorVal, BF
    x=x+61
    y=y+51
NEXT ColorVal
LOCATE 21, 1
PRINT "Press ESC to end"
```

```
PRINT "Press any other key to continue"
VIEW PRINT 23 TO 24
DO
    PaletteVal=1
    DO
        PaletteVal=1-PaletteVal
        COLOR BackGroundVal, PaletteVal
        PRINT "BackGroundVal="; BackGroundVal;
        PRINT "PaletteVal="; PaletteVal
        Pause$=INPUT$(1)
        LOOP UNTIL PaletteVal=1 OR Pause$=Esc$
        BackGroundVal=BackGroundVal+1
LOOP UNTIL BackGroundVal > 15 OR Pause$=Esc$
SCREEN 0
WIDTH 80
END
```

二、改变调色板属性

我们知道，图形语句 CIRCLE，DRAW，LINE 等，都是使用颜色属性，而不是使用颜色值。在上一节，我们利用 COLOR 语句的属性来改变颜色，同时指定调色板。每种调色板只有几种固定的颜色。这样就使图形颜色的使用，受到很大的限制。

这一节将要介绍一种可以分配显示颜色到颜色属性的方法，即改变调色板属性，从而做到分配任意的属性和任意颜色来组成所需要的调色板。这一功能是用 PALETTE 和 PALETTE USING 语句实现的。语句的格式为：

PALETTE [属性，颜色]

PALETTE USING 数组名 [（下标）]

语句中的“属性”为要改变的调色板属性。

“颜色”为给属性赋值的显示颜色号。对于 VGA 和 MCGA，“颜色”必须是长整型数表达式；对于 EGA 可以使用整型数或长整型数表达式。

“下标”是用来设置调色板的第一个数组元素的下标。

PALETTE 语句只能在带有 EGA，VGA 和 MCGA 适配器的系统中工作。

PALETTE 和 PALETTE USING 语句的功能，就是用来改变调色板中一种或多种颜色。没有参量的 PALETTE 语句，把调色板设置回缺省颜色值。

例 12.5.1 在屏幕 8 方式下，通过改变一正方形面积的颜色，使其显示出除背景颜色外的其余 15 种颜色。

程序如下：

```
SCREEN 8
ColorVal=1
DO
    LINE (50, 50) -STEP (120, 50), ColorVal, BF
    pause$ =INPUT$ (1)
    PALETTE ColorVal, ColorVal+1
    ColorVal=ColorVal+1
LOOP UNTIL pause$ = "q" OR ColorVal = 15
END
```

例 12.5.2 在屏幕方式 SCREEN 8 的情况下，首先用缺省的 15 种颜色前景画出 15 个同心的椭圆，然后，利用 PALETTE USING 语句改变这 15 个椭圆的颜色。按任何一个键，便改变一次颜色，但按“q”键后，终止程序的运行。

程序如下：

```
DECLARE SUB InitPalette ( )
DECLARE SUB ChangePalette ( )
DECLARE SUB DrawEllipses ( )
DEFINT A-Z
```

```
DIM SHARED PaletteArray (15)
SCREEN 8                ' 640×200 resolution; 16 colors
InitPalette
DrawEllipses
DO
    Paus $ =INPUT $ (1)
    ChangePalette
LOOP WHILE Paus $ <>"q"   'Shift palette until"q"pressed
END
DEFINT A-Z
'
SUB ChangePalette STATIC
    FOR I=1 TO 15
        PaletteArray(I)=(PaletteArray(I)MOD15)+1
    NEXT I
    'Shift the color displayed by each of the attributes from
    'one to fifteen:
    PALETTE USING PaletteArray(0)
END SUB
DEFINT A-Z
'
SUB  DrawEllipses STATIC
    CONST ASPECT=1/3
    FOR ColorVal=15 TO 1 STEP-1
        Radius=15 * ColorVal
        CIRCLE (320,100),Radius,ColorVal,,,ASPECT
    NEXT
END SUB
DEFINT A-Z
'
```

```
SUB InitPalette STATIC
    FOR I=0 TO 15
        PaletteArray(I)=I
    NEXT I
END SUB
```

程序由三个子程序和一个主程序组成。

子程序 InitPalette 的功能是对数组 PaletteArray 初始化，初始化值为 PaletteArray(0)=0，PaletteArray(1)=1，PaletteArray(2)=2，… PaletteArray(15)=15。

子程序 DrawEllipses 的功能是用15种颜色画出15个同心的椭圆。

子程序 ChangePalette 的功能是改变数组 PaletteArray 的15个元素（0元素除外）的值。其改变方式可用图12.5.1来说明，其中图（a）是数组被变化后的各元素的值，子程序被调用一次，各元素的值就“循环右移”一次；图（b）是第一次调用后，数组各元素值的情况。然后通过语句

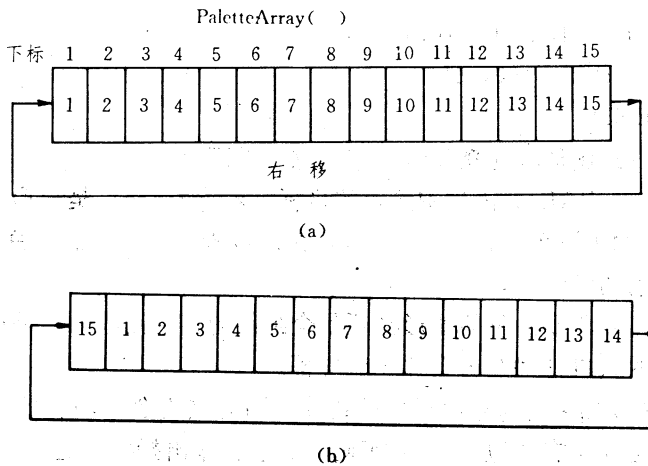


图 12.5.1 变化调色板示意图

PALETTE USING PaletteArray(0)

实现调色板颜色的改变，从而改变了15个椭圆图形的颜色。

主程序的基本任务就是反复调用 Change Palette 子程序。

三、填充图形

在用 LINE 语句画矩形框时，可以用语句的参量 BF 和颜色号为图形填色。实际上，利用 Quick BASIC 的 PAINT 语句可为任何封闭的图形着色，其颜色也是可以选择的。PAINT 语句还能用自己设计的图案来填充封闭图像。

1. 填充颜色

PAINT 语句可用来为封闭图形着色。该语句的格式如下：

PAINT [STEP] (x, y) [, [涂色] [, 边框色]]

语句中的 (x, y) 是所涂图形内一点的坐标。

例 12.5.3 用青色画一个圆，然后将圆的内部涂成洋红色，试编写程序。

程序如下：

```
SCREEN 1
CIRCLE (160, 100), 50, 1
PAINT (160, 100), 2, 1
END
```

在给图形填充颜色时，应注意以下三点：

(1) PAINT 语句中给定的坐标 (x, y)，必须是图形内的一个点。如果这个点在图之外，则对图形以外的部分充满颜色，而不是当前的背景色。

(2) 所要涂色的图形必须是全封闭的，否则涂上的颜色将从图形的缺口处“漏出”并充满屏幕。

(3) 如果用与被涂对象的边线不同的颜色涂色，则 PAINT 语句中必须含有“边框色”，否则，涂色将超出指定的范围。

例 12.5.4 下面的程序是画一条带有青色尾巴、粉红身的鱼的图形，如图12.5.2所示。

程序如下：

```
CONST PI=3.14159
CLS
SCREEN 1
CIRCLE (190, 100), 100, 1,,, .3
CIRCLE (265, 92), 5, 1,,, .7
PAINT (190, 100), 2, 1
LINE (40, 120) -STEP (0, -40), 2
LINE-STEP (60, 20), 2
LINE-STEP (-60, 20), 2
PAINT (50, 100), 1, 2
CIRCLE (250, 100), 30, 0, PI*3/4, PI*5/4, 1.5
FOR y=90 TO 110 STEP 4
   LINE (40, y) - (52, y), 0
NEXT y
END
```



图 12.5.2　例12.5.4图形的屏幕显示

2. 填充图案

利用 PAINT 语句，可以把用户自己设计的图案填充到一个封闭图形中，通常把这种技术叫作“贴砖”(Tilling)。

当进行图案填充时，PAINT 语句中的“涂色”应是一个字符

串表达式，而不是数值表达式，其一般格式为：

CHR $（自变量1）＋CHR $（自变量2）＋…＋CHR $（自变量 n）

这里的“自变量”是8位整型量（0—255），最多可有64个CHR $元素。

图案的每一块“砖”是一个矩形，矩形的宽为8位，长最多可达64个字节。每行中8位所能代表的像素数，随屏幕方式下颜色属性数的增加而降低。例如，在SCREEN 2方式下，每个像素占1位，在SCREEN 2方式中，则每个像素占2位。一般来说，对任一给定的屏幕方式，每个像素所占的位数，可用下式计算。

$$每个像素位数=LOG_2（属性数）$$

首先，我们看如何在屏幕方式2生成一个单色图案。下面以产生图案“M”为例来说明。

画如图12.5.3（a）所示图案的底图，根据底图确定哪些像素是“亮点”。图中每个方格就是一个像素，令带有“*”符号的像素为亮点。并用1表示亮，用0表示不亮，则可得到如图12.5.3（b）所示的用二进制数表示的图案。

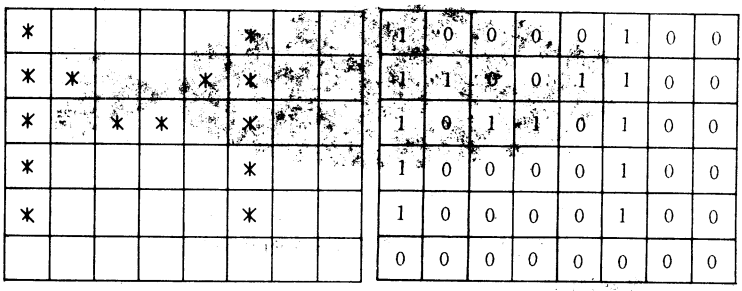


图 12.5.3 图案的底图及其二进制表示

把图中每一行二进制数表示为一个十六进制数，于是有：

1 0 0 0 0 1 0 0 =&H84

1 1 0 0 1 1 0 0 =&HCC

1 0 1 1 0 1 0 0 =&HB4

1 0 0 0 0 1 0 0 =&H84

1 0 0 0 0 1 0 0 =&H84

0 0 0 0 0 0 0 0 =&H00

把上述数据作为ASCII码，用函数CHR$求其相应字符，并把它们连成一个字符串，如

```
Tile$ =CHR$ (&H84)+CHR$ (&HCC)+CHR$ (&HB4)
       + CHR $ (&H84) + CHR $ (&H84) + CHR
       $ (&H00)
```

将字符串Tile$作为语句PAINT的参量，就实现了图案“M”的填充。

例 12.5.5 编写程序，使其画一个圆，然后用上面定义的图案“M”填充于该圆内。

程序如下：

```
SCREEN 2
CLS
Tile$ =CHR$ (&H84)+CHR$ (&HCC)+CHR$ (&H84)
Tile$ =Tile$ +CHR$ (&H84)+CHR$ (&H84)
       +CHR$ (&H0)
CIRCLE STEP(0,0), 150
PAINT STEP(0,0), Tile$
END
```

程序的输出如图12.5.4所示。

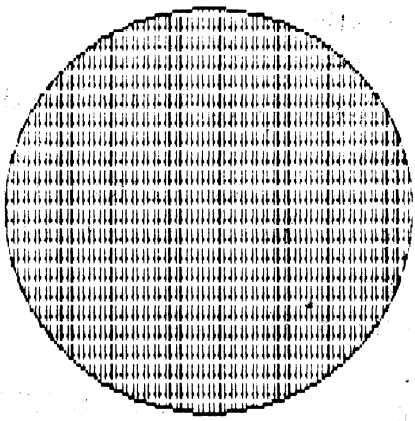


图 12.5.4　例15.5.5程序的输出图形

再看怎样在屏幕方式1下生成多色图案。

在 SCREEN 1方式下，每个像素的位数为：

$LOG_2 4=2$

因此，图案砖每行只有4列，即在8位长度内，每两位表示一个图素，一行有4个图素。行数最多可达64。

图12.5.5(a)为青和洋红色构成的斜线砖底图，图12.5.5(b)将颜色号换成相应的二进制数。

青	洋红	洋红	洋红
洋红	青	洋红	洋红
洋红	洋红	青	洋红
洋红	洋红	洋红	青

(a)

01	10	10	10
10	01	10	10
10	10	01	10
10	10	10	01

(b)

图 12.5.5 图案底图及其二进制表示

图中4行的二进制数可写成如下的十六进制数：

0 1 1 0 1 0 1 0 =&H6A

1 0 0 1 1 0 1 0 =&H9A

1 0 1 0 0 1 1 0 =&HA6

1 0 1 0 1 0 0 1 =&HA9

如前所述，将这些十六进制数作为 ASCII 码转换为字符串，即

Tile $ = CHR $ (&H6A) + CHR $ (&H9A) + CHR $ (&HA6)+CHR $ (&HA9)

便可用语句

PANT (x, y) Tile $

实现图案的填充。

例 12.5.6 用上面介绍的图案填充一个三角形

上述图案的颜色，正是屏幕方式 SCREEN 1的缺省颜色，因此，进入此屏幕方式就可得到所需颜色。程序如下：

```
SCREEN 1
Tile $ =CHR $ (&H6A)+CHR $ (&H9A)
Tile $ =Tile $ +CHR $ (&HA6)+CHR $ (&HA9)
LINE(10,25)-(310,25)
LINE-(160,175)
LINE-(10,25)
PAINT (160,100),Tile $
END
```

要注意的是，当被填充图形的边框颜色也包含在图内时，则应在 PAINT 语句中给出“边框色”参量，否则，图案会从图形的边缘溢出。例如，上边的程序，可作如下的变化：

```
SCREEN 1
Tile $ = CHR $ (&H6A) + CHR $ (&H9A) + CHR
        $ (&HA6)+CHR $ (&HA9)
LINE(10,25)-(310,25),2
LINE-(160,175),2
LINE-(10,25),2
PAINT (160,100),Tile $ ,2
END
```

第六节 图形窗口

前面所讲的图形操作，都是在整个屏幕上进行的，而且屏幕的左上角为绝对坐标的原点 (0, 0)。但是，也可以在屏幕上定义一个矩形区域，在其内部进行图形操作。这个区域叫作图形窗口。

这一节，我们还要学习怎样重新定义图形的坐标。

一、定义图形窗口

利用 VIEW 语句可以在屏幕上定义一个图形窗口。窗口一旦被定义，所有的图形操作都将在其内部实现，超过窗口边界的部分将被裁剪掉。窗口内部的图形可以用 CLS 1语句清除，而对窗口外的屏幕没有影响。VIEW 语句的格式如下：

VIEW [[SCREEN] (x1, y1) — (x2, y2) [, [颜色号] [, 边框]]]

各参量的含义说明如下：

SCREEN：当选用这个参量时，所有像素的坐标保持为绝对坐标；当省略它时，所有像素的坐标都相对于子窗口，而不是相对于整个屏幕的，即所定义图形窗口的左上角的坐标为原点（0，0）。

(x1，y1)—(x2，y2)：x1，y1，x2，y2为数值表达式，分别为所定义的图形窗口的左上角和右下角的坐标。

颜色号：图形窗口的颜色。如省略，则图形窗口为背景颜色。

边框：数值表达式。选择该项时，显示图形窗口的边框；如果省略该参量，则不显示窗口的边框。

如果 VIEW 语句不带任何参量，则把整个屏幕定义为一个图形窗口。

例 12.6.1 下面的两小段程序，显示出参量 SCREEN 在 VIEW 语句所产生的影响。

```
SCREEN 1
VIEW (100, 50) — (300, 150),, 1
CIRCLE (100, 50), 50
END
SCREEN 1
VIEW SCREEN (100, 50) — (300, 150),, 1
```

```
CIRCLE (100, 50), 50
END
```

这两个程序都是画一个大小相同的圆。在第一个程序中，VIEW 语句不带 SCREEN 参量，在所定义的图形窗口内画出一个完整的圆。第二个程序的 VIEW 语句有参量 SCREEN，这时圆心的坐标正好是所定义的图形窗口的左上角坐标，因此，这个程序只能画出1/4个圆，其余部分超出定义区域的边界，不能显示。

图12.6.1给出了这两个程序的输出

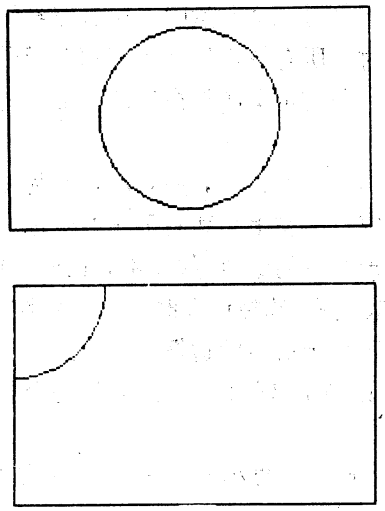


图 12.6.1　例12.6.1程序的输出

在一个程序中可以多次使用 VIEW 语句，建立多个图形窗口。

二、建立文本窗口

可以在屏幕的指定区域，建立用于显示字符的文本窗口。设置

屏幕文本窗口的语句为：

VIEW PRINT［顶行 TO 底行］

语句格式中的“顶行”是窗口的上界，“底行”是下界。屏幕的字符输出，限制在“顶行”和“底行”之间的范围内。

利用文本窗口，可以控制显示于该窗口内文本的滚动。而不使用文本窗口，则输出到底行时，屏幕就开始上滚，使屏幕顶端的文本或图形消失。在使用了 VIEW PRINT 语句后，滚动只发生在文本窗口的顶行与底行之间。

利用 CLS 2 语句可清除文本窗口内的内容，而屏幕上其他地方的内容则保持不变。

例 12.6.2 下面的程序演示 VIEW PRINT 语句的作用：

```
SCREEN 2
VIEW PRINT 20 TO 24
FOR i =1 TO 1000
    PRINT RND,
NEXT i
END
```

三、重定义图形窗口的坐标

前面所使用的坐标，都是相对于窗口左上角的距离，称为物理坐标，原点的坐标为 (0，0)。这种坐标系统，与笛卡尔坐标系统不同，可能很不习惯。

使用 WINDOW 语句，可以改变图形窗口的物理坐标值，也可以改变 x 轴和 y 轴的方向。经 WINDOW 语句确定的图形窗口坐标系统，称为逻辑坐标。

WINDOW 语句的格式如下：

WINDOW［［SCREEN］(x1，y1) — (x2，y2)］

语句中任选项 SCREEN 被选用时，则 (x1，y1) 为图形窗口

左上角的逻辑坐标；(x2，y2) 为右下角的逻辑坐标。x 轴的方向为从左向右，y 轴的方向从上向下。不用 SCREEN 选项时，则 (x1，y1) 为图形窗口左下角的逻辑坐标，(x2，y2) 为右上角的逻辑坐标。x 轴的方向为从左向右，y 轴的方向为从下向上。图12.6.2清楚地表示出 WINDOW 语句的功能。

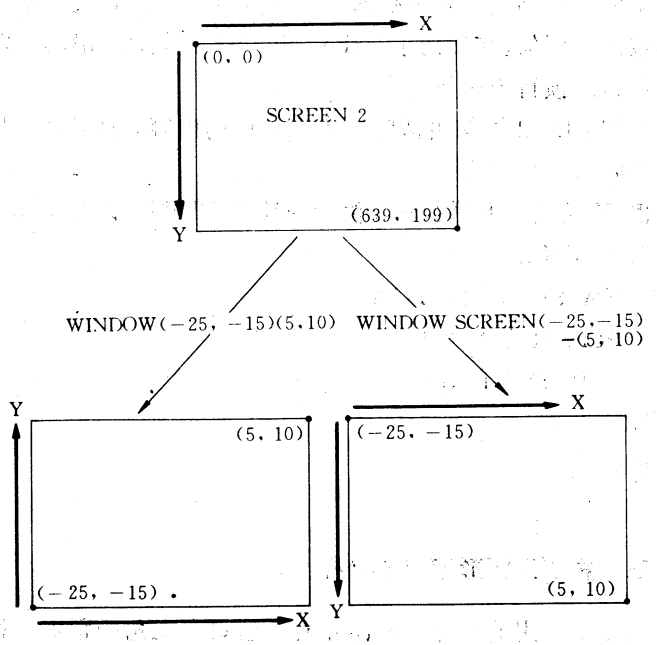


图 12.6.2　WINDOW 语句功能示例

执行不带任何参量的 WINDOW 语句时，恢复原来的物理坐标系。

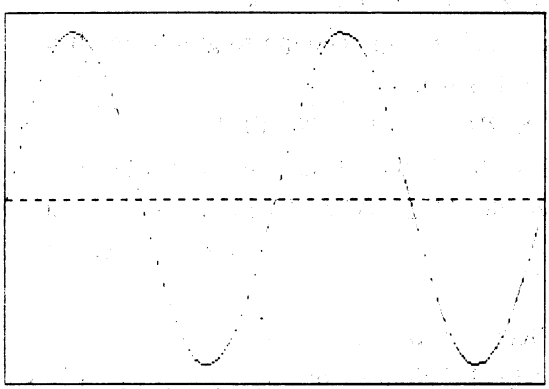
例12.6.3　利用 VIEW 和 WINDOW 两个语句生成正弦曲线。正弦曲线的周期数由用户给定。

程序如下：

```
SCREEN 2
PI＝3.141592653589#
```

```
'Viewport sized to proper scale for graph
VIEW(20,2)-(620,172),,1
'Make window large enouph to graph sine from
'0 to 2PI radians:
WINDOW(0,-1.1)-(2*PI,1.1)
Style%=&HFF00  'Use to make dashed line.
VIEW PRINT 23 TO 24 'Scroll printed output in
                    'rows 23 and 24.
DO
   PRINT TAB (20);
   INPUT "Number of cycles (0 to end):", Cycles
   CLS
   LINE (2*PI,0)-(0,0),,,Style% 'Draw the x axis.
   IF Cycles > 0 THEN
      FOR x = 0 TO 2 * PI STEP .1
         y=SIN(Cycles*x) 'Calculate the y coordinate.
         LINE-(x,y)   'Draw a line from the last
                      'point to the new point.
      NEXT x
   END IF
LOOP WHILE Cycles > 0
END
```

程序的输出如图12.6.3所示。



Number of cycles (0 to end) 2
Number of cycles (0 to end)

图 12.6.3 例12.6.3程序的输出

第七节 简单动画技术

Quick BASIC 的动画功能很强，包括字符动画，图形方式动画，用 GET 和 PUT 语句作动画和屏幕分页动画等。本节将简要地介绍这些动画技术的实现方法。

一、字符动画

用普通字符或图形字符作为动画对象，可以使其在屏幕上作上下左右地移动，从而产生动画效果。每次移动的距离以字符的行和列为单位进行计算。字符动画在文本方式下进行。

利用 ASCII 219—223的图形字符，可以组成简单的图形，把图形加以变换，并在屏幕的不同位置上交替显示，就可以得到动画效果。

例 12.7.1 令英文字母“O”在屏幕上作水平（由左往右）、垂直（由上往下）和斜向滚动。试编写程序。

字母“O”出现一次，要用以下两个语句实现：

PRINT “O”: PRINT “ ”

在屏幕不同位置上执行上述语句，便形成了字母“O”的移动。程序如下：

```
SCREEN 0
CLS
FOR x = 20 TO 60
    LOCATE 10, x
    PRINT "0"
    FOR i = 1 TO 500: NEXT i
    LOCATE 10, x
    PRINT " "
NEXT x
FOR y = 1 TO 20

    LOCATE y, 40
    PRINT "O"
    FOR i = 1 TO 1000: NEXT i
    LOCATE y, 40
    PRINT " "
NEXT y
y=0
FOR x = 1 TO 40 STEP 2
    y=y+1
    LOCATE y, x
    PRINT "O"
    FOR i = 1 TO 500: NEXT i
    LOCATE y, x
    PRINT " "
```

```
NEXT x
x = 60
FOR y = 1 TO 23
    x = x-1
    LOCATE y, x
    PRINT "0"
    FOR i = 1 TO 500: NEXT i
    LOCATE y, x
    PRINT " "
NEXT y
END
```

二、图形方式动画

字符动画是在文本方式下产生的，动画对象只能是字符或由字符组成的简单图形。字符动画的移动距离只能是以字符的行和列为单位，因此，动画效果不可能很理想。

图形方式动画是在图形方式下产生的，动画对象由像素构成，对图像的运动可以进行精细的控制。图像移动的原理与字符动画没有什么区别。

例12.7.2 下面的程序是利用 PSET 和 PRESET 两个语句实现点在一个方框中的运动。

```
SCREEN 1
CLS
xleft=50: xright=250
ytop=50: ybottom=150
LINE(xleft,ytop)-(xright,ybottom),2,B
dx=1: dy=1
x=xleft+INT((xright-xleft)/2)
y=ytop+INT((ybottom-ytop)/2)
```

```
DO
    PSET(x,y)
    IF x-1=xleft OR x+1=xright THEN dx=-dx
    IF y-1=ytop OR y+1=ybottom THEN dy=-dy
    PRESET(x,y)
      x=x+dx:y=y+dy
LOOP UNTIL INKEYY <> " "
END
```

三、用 GET 和 PUT 语句作动画

无论是字符动画还是图形方式动画，动画的原理都是一样的，即要不断地擦除图形和重画图形，这个过程要化费较多的时间。利用 Quick BASIC 提供的图形语句 GET，将图形存放到数组中，需要时，再用图形语句 PUT 将该图形显示到所要求的位置上，就可以方便地实现图形的移动。这样，避免了一条一条线地擦除和一条一条线地重画。

1. GET 语句

GET 语句格式为：

GET [STEP](x1,y1)-[STEP](x2,y2),数组名[(下标)]

这个语句的功能是把屏幕上指定区域的图像存放到“数组”中。

语句中 x1，y1，x2，y2为屏幕上矩形区域的对角顶点坐标。有选择项 STEP 时，上述坐标为相对坐标。数组名是用于存储图像的数组名，它通常是整型数组。数组的大小取决于以下三个因素：

(1) 包含着图像的矩形的高和宽；

(2) 屏幕方式；

(3) 数组类型。

计算存储图形所需要的字节数的公式如下：

4+矩形高 * 位平面数 * INT ((矩形宽 * 每个像素的比特数

/平面数＋7）/8）

其中：

矩形高＝ABS（y2－y1）＋1

矩形宽＝ABS（x2－x1）＋1

每个像素的比特数＝LOG_2（属性数）

位平面数与屏幕方式有关，如表12.7.1所示。

表12.7.1

屏幕方式	位平面数
1，2，11，3	2
9（视频内存64K），10	1
7，8，9（视频内存764K），12	4

用上面公式给出的字节数除以每个数组元素所占的字节数，就得到所需的数组元素数。

2. PUT 语句

PUT 语句的格式为：

PUT［STEP］（x，y），数组名［（下标）］［，操作］

语句的功能是，把由 GET 语句存储的图像映像，在屏幕的指定区域重新显示出来。

语句中的（x，y）是重新显示的矩形区域的左上角坐标，数组名的含义与 GET 语句中的数组名的含义是一样的，下标指明从哪个数组元素开始显示图像映像。

语句中的“操作”是用来指定显示方式的。有以下五种操作方式：

（1）PSET：按存入数组的像素信息原样显示；

（2）PRESET：把存入数组的像素信息取反后显示；

（3）OR：数组中的像素信息与屏幕上的像素信息进行“或”运算后显示；

(4) AND：数组中的像素信息与屏幕上的像素信息进行“与”运算后显示；

(5) XOR：数组中的像素信息与屏幕上的像素信息进行“按位加”运算后显示。

例 12.7.3 PUT 语句中五种显示操作的演示程序。

程序首先画一个椭圆并涂色，然后存入数组中。用图填充一个矩形，作为以后重新显示椭圆的屏幕背景。最后，用不同的显示操作，把椭圆重新显示在矩形背景区域上。

程序如下：

```
SCREEN 2
DIM circleimage(1 TO 485) AS INTEGER
'Draw and paint an ellipse then store its image with GET:
CIRCLE(22,100),80,,,,4
PAINT(22,100)
GET(0,22)-(44,180)'circleimage
CLS
'Draw a box and fill it with a pattern:
LINE(40,40)-(600,160),,B
Pattern$ =CHR$(126)+CHR$(0)+CHR$(126)
          +CHR$(126)
PAINT (50,50),Pattern$
'Put the images of the illipse over the box using
'the different action options:
PUT(100,20),circleimage,PSET
PUT(200,20),circleimage,PRESET
PUT(300,20),circleimage,AND
PUT(400,20),circleimage,OR
PUT(500,20),circleimage,XOR
END
```

程序的输出，如图12.7.1所示。

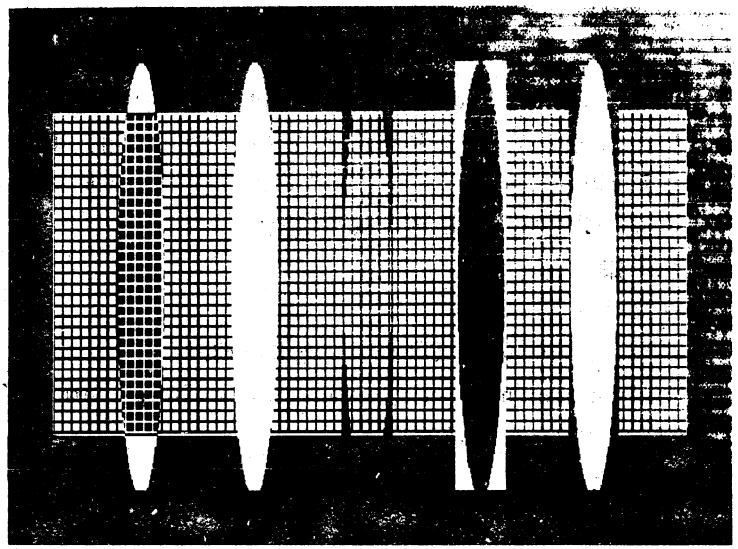


图 12.7.1　例12.7.3程序的输出

3. 用 GET 和 PUT 语句实现动画

用 GET 和 PUT 语句实现动画时，使用最多的显示方式是 XOR 和 PSET。但是，使用 PSET 时，会擦除背景；而使用 XOR 时，能保留背景。以用 XOR 为例，实现动画的步骤如下：

(1)用 GET 语句把动画对象存入数组；

(2)用 XOR 操作把动画对象 PUT 到屏幕上；

(3)计算动画对象的新位置；

(4) 再一次用 XOR 操作，把动画对象 PUT 到原来的位置上，以清除旧的对象；

(5)重复步骤(2)—(4)。

例 12.7.4　将屏幕分为三块不同颜色的矩形带，令一涂色的椭圆在屏幕上随机地游动。

程序如下：

```
SCREEN 1
DIM I(1 TO 800) AS INTEGER
CIRCLE(160,100),30,,,,,.45
PAINT(160,100),2,3
GET(130,70)-(190,130),I
LINE(0,0)-(100,150),1,BF
LINE(101,0)-(200,150),2,BF
LINE(201,0)-(300,150),3,BF
LOCATE 23,2:PRINT"Press any key to exxit."
DO
    x=240*RND:y=100*RND
    PUT (x,y),I,XOR
    FOR k=1 TO 200:NEXT k
    PUT (x,y),I,XOR
LOOP UNTIL INKEY$<>""
END
```

四、屏幕分页动画

屏幕分页动画，是用屏幕的缓冲区分页显示技术来实现动画的。例如，CGA 的屏幕缓冲区有16K，在文本方式下，可分为若干页。在40列文本时，分为8页，编号为0—7；在80列文本时，分为4页，编号为0到3。

屏幕的缓冲区，也称为视频内存。当观看屏幕上的一个页面时，可以同时在其他页面上输出图形。在视频内存中，屏幕上可见的区域叫作“可见页”，而被放置输出图形的区域叫作“活动页”。用 SCREEN 语句可以选择可见页和活动页。语句的格式为：

SCREEN,, 活动页，可见页

活动页和可见页只在文本方式下才是有效的。实现动画的方

法是，将需要显示的信息分别写入各活动页，然后通过选择可见页使其显示在屏幕上，由页面的变化，产生动画效果。

例 12.7.5 下面的程序是用 DRAW 语句画一立方体，并用 TA 微（子）命令改变立方体的角度。在屏幕方式1下，通过活动页和可见页的交换实现立方体的360°旋转。

程序如下：

```
'The macro string to draw the cube and paint its sides:
Plot$ = "BR30 BU25 C1 R54 U45 L54 D45 BE20 P1, 1
        G20 C2 G20"
Plot$ = PlotY + "R54 E20 L54 BD5 P2, 2 U5 C4 G20 U45
        E20 D45 BL5 P4,4"
APage% = 1   'Initialize values for the active and visual
VPage% = 0   'pages, as well as the angle of rotation.
Angle% = 0
DO
    'Draw to the active page while showing
    'the visual page:
    SCREEN 7,, APage%, VPage%
    CLS 1
    'Rotate the cube "Angle%" degrees:
    DRAW "TA" + STR$ (Angle%) + Plot$
    'Angle% is some multiple of 15 degrees:
    Angle% = (Angle% + 15) MOD 360
    'Switch the active and visual pages:
    SWAP APage%, VPage%
LOOP WHILE INKEY$ = ""   'A key press ends the
                         program.
END
```

小　　结

图形在软件中占有很重要的地位。本章的目的是为编制图形程序打下基础。由于 Quick BASIC 有较强的图形功能，因此，编写图形程序不是一件困难的事。

本章介绍了下列内容：

(1) 如何选择屏幕方式；

(2) 怎样画基本图形；

(3) 怎样使用颜色；

(4) 如何使用图形窗口；

(5) 基本动画技术。

本章涉及的 BASIC 语句主要有：

(1) 用于设置屏幕方式的语句 SCREEN；

(2) 在屏幕上绘制简单图形的语句 LINE，CIRCLE 和 DRAW；

(3) 使用颜色的语句 COLOR 和 PALETTE；

(4) 定义窗口坐标的语句 VIEW 和 VIEW PRINT；

(5) 填充图形的 PAINT 语句；

(6) 用于制作动画的语句 GET 和 PUT。

习　　题

1-1　编写程序，画函数 $y=e^{-x}$ 的曲线。

1-2　先用 VIEW 语句和 WINDOW 语句设定窗口和坐标系，然后在其中画出 $y=e^{-x}$ 曲线。

1-3　根据例12.4.1的程序，指出图12.4.1图形各顶点的坐标。

1-4　在例12.4.2中，如果只执行 DRAW“a2”语句，而不执行 DRAW“a3”语句，程序输出的图形是什么样子，绘图说明。

第十三章　陷阱技术

在运行程序时，可能会出现一些意外情况，如果不能对这些情况进行适当的处理，则可能影响程序的正确执行或无法得到预期的结果。用于捕获随机发生的这种事件的方法，称为陷阱技术。

有两类事件：错误事件（简称错误）和程序运行中发生非错误性质的事件（简称事件）。Quick BASIC.可以设置错误陷阱和事件陷阱，在程序运行中如果发生了上述事件，便会转到相应事件的陷阱处理程序，处理完后，再返回到中断的程序，继续执行。

第一节　错误陷阱

一、设置错误陷阱

设置错误陷阱语句的格式为：

ON ERROR GOTO 行

该语句的功能是，设置错误陷阱并指定错误处理程序的入口。

语句中参量“行”就是错误处理程序第一行的行号或标号。当BASIC 遇到 ERROR GOTO 行语句时，程序运行过程中的任何错误，都会转到该语句指出的程序入口，执行错误处理程序。

在第十二章的第二节，我们看到过这样的例子：

ON ERROR GOTO Message

但是语句

ON ERROR GOTO 0

有特定的意义。当这个语句出现在错误处理程序之外时，它的作用是禁止错误陷阱。若是该语句出现在错误程序处理之内，则它将使BASIC给出错误信息并停止运行。

例如，在第十二章第二节中的那个程序，语句ON ERROR GOTO 0出现在错误处理程序（Message）之外，它的作用就是关闭错误陷阱。

当发生错误陷阱时，其他所有陷阱将自动被禁止。

二、错误的模拟与识别

Quick BASIC运行期间发生的错误，可以用ERROR语句来模拟。ERROR语句的格式为：

ERROR n

该语句的功能是模拟发生一个Quick BASIC错误或者用户自己定义的错误代码。语句中n是一个数值表达式，取值范围为0—255。如果n值与Quick BASIC使用的错误代码相同，则ERROR语句就模拟产生那种错误。例如，

```
T=54
ERROR T
```

模拟错误“Bad file mode”。

使用与Quick BASIC预定义的错误代码不同时，用户可以定义自己的错误代码。例如，

```
ON ERROR GOTO PP
INPUT "Enter your number", n
IF n>100 THEN ERROR 210
……
PP:
……
```

错误的识别，可用ERR函数和ERL函数来实现。这两个函数的格式是：

ERR

ERL

函数ERR的功能是返回错误代码，而函数ERL则是返回出现错误的行号。ERL函数只能返回行号，不能返回标号，如果发生错误的行没有行号，则ERL函数总是返回0。

这两个函数通常是放在错误处理程序中，用来确定发生错误的性质。

三、从错误处理程序返回

在执行完错误处理程序后，一般要从错误处理程序返回中断的程序，以便使程序继续运行下去。执行这种返回的语句是RESUME。它有以下三种格式：

(1) RESUME [0]

(2) RESUME NEXT

(3) RESUME 行号|标号

RESUME语句的功能：第一种格式，是从发生错误的语句处继续执行；第二种格式，是从发生错误的语句后面那个语句继续执行；第三种格式，是从指定的行号（或标号）处继续执行。

例13.1.1 下面是一个用打印机打印数据的程序。它能判断和处理下述四种不正常情况：

(1) “Out of DATA”：错误代码＝4

(2) “Device fault”：错误代码＝25

(3) “Out of paper”：错误代码＝27

(4) 其他意想不到的错误，终止程序的执行。

对于上述不同的错误，程序进行了不同的处理。程序如下：

```
'Data for this program:
```

```
DATA BASIC, Pascal, FORTRAN, C, Moduls, Forth
DATA LISP, ADA, COBOL
'Define symbolic constants:
CONST False =0, True=NOT False
EndofData=False
ON ERROR GOTO handler 'Active error trapping.
OPEN "LPT1:" FOR OUTPUT AS #1 'Open the printer.
DO
    READ Buffer $
    IF NOT EndofData THEN
        PRINT #1, Buffer $
    ELSE
        EXIT DO
    END IF
LOOP
CLOSE #1
END
handler:
  SELECT CASE ERR
    CASE 4
          PRINT "All data are printed."
          EndofData=True
          RESUME NEXT
    CASE 25
          PRINT "Turn printer on. then";
          PRINT "press any key to continue."
          pause $ =INPUT $ (1)
          RESUME
```

```
    CASE 27
        PRINT "Printer is out of paper. Insert"
        PRINT "paper, then press any key to continue."
        pause$ =INPUT$(1)
        RESTORE
        RESUME
    CASE ELSE
        ON ERROR GOTO 0
    END SELECT
```

最后，要特别指出的是，如果不是在 Quick BASIC 集成环境下生成的带有 ON ERROR 或 RESUME 语句程序的 EXE 文件，应使用编译选择项/e 或/x。

第二节　事件陷阱

一、设置事件陷阱

设置事件陷阱的语句是：

ON 事件 GOSUB {行号|行标号}

语句的功能是设置事件陷阱并指定事件处理子程序的入口。当“事件”发生时，该子程序被执行。在事件陷阱中可以被捕获的事件有下列几种：

(1) COM(n)：n 是通讯接口号，取值为1或2。当通讯接口收到字符时，转移到子程序。

(2) KEY(n)：n 是整型表达式，它是功能键、方向键或用户定义键的编号。按下指定的键时，转移到子程序。

(3) PEN：当启动光笔时转移到子程序。

(4) PLAY(n)：n是整型表达式，取值范围为1—32。当音乐缓冲区中的音符从n变到n－1时，转移到子程序。

(5) STRIG(n)：n是控制杆触发器号。在第一个控制杆上，0表示下面的按钮，4表示上面的按钮，在第二个控制杆上，2表示下面的按钮，6表示上面的按钮。当控制杆触发器被按下时，转移到子程序。

(6) TIMER (n)：n是整型表达式，取值范围为1—86400(1秒—24小时)。到达n秒时转移到子程序。

二、事件陷阱的打开与关闭

“ON 事件”语句只能设置陷阱和指定子程序入口，是否调用子程序则由打开事件陷阱和关闭事件陷阱语句来决定。这两个语句的格式是：

事件 ON

事件 OFF

事件 STOP

语句“事件 ON”的功能是允许事件陷阱，只有执行此语句之后，才能发生事件陷阱。

语句“事件 OFF”的功能是禁止事件陷阱，当事件陷阱被禁止时，所发生的事件被忽略。

语句“事件 STOP”的功能是暂停事件陷阱，执行此语句后不发生事件陷阱，但它可以记住所发生的事件，并能在执行“事件 ON”语句时进行事件的处理。

当事件陷阱发生并调用子程序时，Quick BASIC 自动执行一个“事件 STOP”语句，以防止递归陷阱。如果在子程序中没有使用语句“事件 OFF”，则陷阱子程序的 RETURN 语句自动执行一个“事件 ON”语句。

三、事件的处理

为了用事件陷阱捕获并处理一个事件，必须做下述三件事：

(1) 定义一个事件处理子程序；

(2) 设置事件陷阱；

(3) 启动事件陷阱。

我们已经知道，Quick BASIC 可以捕获六种事件，对于每一种事件，其处理程序是不同的。

下面着重介绍功能键陷阱、用户定义键陷阱、时钟陷阱和音乐事件陷阱。

第三节　功能键陷阱和时钟陷阱

一、功能键陷阱

功能键陷阱是用 ON KEY (n) 语句设置的。利用功能键陷阱，可以在按下某个指定键时，转移事件处理子程序。语句中 n 是功能键的键号，其定义如下：

1—10	功能键 F1—F10
11	方向键↑
12	方向键←
13	方向键→
14	方向键↓
15—25	用户自定义的功能键
30	功能键 F11
31	功能键 F12

例 13.3.1　下面的程序是在屏幕的上端显示时间。但是，每当按下 F1键时，在屏幕的下端显示“Trap”。这个程序就是用功能

键陷阱实现的。

程序如下：

```
CLS
ON KEY(1) GOSUB handler
DO
    KEY(1) STOP
    LOCATE 1,36
    PRINT TIMEY
    KEY(1) ON
LOOP
handler:
    LOCATE 24,1
    PRINT "Trap"
    RETURN
```

二、用户定义键陷阱

键的定义是用 KEY 语句实现的。语句的格式为：

KEY n，CHR $（键盘标志）+CHR $（扫描码）

其中 n 是整型表达式，取值范围为15—25。“键盘标志”的十六进制值如下：

&H00	无键盘标志
&H01—&H03	SHIFT 键
&H04	Ctrl 键
&H08	Alt 键
&H20	Numlock 键
&H40	Capslock 键
&H80	101键盘的扩展键

83个键的键盘扫描码如表13.3.1所示。

表13.3.1

键	代码	键	代码	键	代码	键	代码
ESC	01	U	16	|或\	2B	F6	40
!或1	02	I	17	Z	2C	F7	41
@或2	03	O	18	X	2D	F8	42
#或3	04	P	19	C	2E	F9	43
$或4	05	} 或]	1A	V	2F	F10	44
%或5	06	{或 [	1B	B	30	Numlock	45
^ 或6	07	RETURN	1C	N	31	scrolllock	46
&或7	08	Ctrl	1D	M	32	Home 或7	47
*或8	09	A	1E	<或,	33	↑或8	48
(或9	0A	S	1F	> 或.	34	PgUp 或9	49
) 或0	0B	D	20	?或/	35	—	4A
-或-	0C	F	21	右 SHIFT	36	←或4	4B
+或=	0D	G	22	Prtsc 或 *	37	5	4C
←	0E	H	23	Alt	38	→或6	4D
TAB	0F	J	24	spacebar	39	+	4E
Q	10	K	25	capslock	3A	END 或1	4F
W	11	L	26	F1	3B	↓或2	50
E	12	: 或;	27	F2	3C	PgDn 或3	51
R	13	“或’	28	F3	3D	Ins 或0	52
T	14	~或 ‘	29	F4	3E	Del 或.	53
Y	15	左 SHIFT	2A	F5	3F		

可以把某些值加在一起来测试多种变换状态。例如键盘标志：&H12=&H04+&08

用来测试 Ctrl 和 Alt 键。十六进制数和十进制数可以通用。例如

```
KEY 20,CHR $ (0)+CHR $ (25)
```

把大写字母 P 设置为陷阱功能键。

使用用户定义键陷阱的步骤一般为：

(1)用 KEY 语句定义键：

KEY n,CHR $ (键盘标志)+CHR $ (扫描码)

(2)用 ON KEY(n) 语句设置陷阱：

ON KEY(n) GOSUB 行

(3)用 KEY(n) ON 语句打开陷阱：

KEY(n) ON

例 13.3.2 对 Ctrl+S 和向下光标键（方向键↓）设置陷阱。Ctrl+S 的处理程序是记录按此键的次数。↓键的处理程序是打印 Ctrl+S 的陷阱次数并终止程序的运行。

程序如下：

```
i=0
PRINT "Press DOWN direction key to end."
KEY 15,CHR $ (&H4)+CHR $ (&H1F)
KEY (15) ON
KEY (14) ON
ON KEY (15) GOSUB Keytrap
ON KEY (14) GOSUB Endprog
Idle: GOTO Idle
Keytrap:
    i=i+1
    RETURN
Endprog:
    PRINT "Ctrl+s trapping"; i; "times."
    END
```

RETURN

三、时钟陷阱

在处理与时间有关的问题时，可以利用时钟陷阱。设置时钟陷阱的语句格式为：

ON TIMER (n) GOSUB {行号|标号}

其中 n 的取值范围为1—86400秒 (24小时)。它表示的是时间间隔 n 已到事件。

时钟陷阱通常用于需要定时进行操作的程序中，例如定时数据采样，定时回答等软件。

时钟陷阱的一般操作步骤为：

(1) 编制一个时钟事件的处理程序；

(2) 用 ON TIMER(n)设置时钟事件的时间间隔；

(3) 用 TIMER ON 启动时钟计时。

当时间计到设定的时间间隔时，发出时钟中断，捕获事件并转到事件处理子程序。此时，时钟归零，重新开始计时。当遇到 TIMER OFF 语句时，上述过程停止。

例 13.3.3 下面的程序利用时钟陷阱，每2秒钟在屏幕上修改一次用 TIME $ 函数显示的系统时钟的时间。程序如下：

```
CLS
LOCATE 24, 1
PRINT "Press any key to end."
TIMER ON
ON TIMER(2) GOSUB Display
DO WHILE INKEY$ = "": LOOP
END
Display:
```

LOCATE 12，35：PRINT TIME$

RETURN

在程序的 DO…LOOP 处，可以编写要完成的其他任务的程序，使程序既能显示时间，又能作其他事情。

第四节　音乐事件陷阱

利用音乐事件陷阱，可以以“后台方式”来演奏音乐，而不影响其他操作在“前台”进行，因此，音乐事件陷阱很有实用价值。

一、音乐的演奏

在说明如何实现音乐事件陷阱之前，先介绍一下如何用 Quick BASIC 在计算机上演奏音乐。音乐的演奏是通过 PLAY 语句实现的。该语句的格式为：

PLAY 命令字符串

语句的功能是演奏由给定的命令字符串所表示的音乐。“命令字符串”是一种音乐语言，它由一些单个字符命令组成，用以表示音符的音高、音长、速度及演奏方式等。

和普通乐谱的情况相似，音乐语言中音调用字母 A—G 来表示。在普通乐谱中，升半音和降半音分别用“#”和“b”表示，而在音乐语言中则用“#”和“—”表示。

同一个音调中的音高，从低到高依次为 C，D，E，F，G，A，B。如果要升高或降低一个或几个八度音，需使用命令：

On

其中 n 的取值范围为0—6，因此，可以设置七个八度音程。普通中音用命令 O3设置。

此外，提高或降低一个八度，也可以用“>”或“<”来实

现。

从O0—O6共有七个八度，加升、降符，每个八度有12个音调，因此，总共有84个音调。用命令Nn可以演奏84个音调中的任何一个。n的取值范围为0—84。当n＝0时，表示休止符。

音的长度，或称音的时值，是用命令Ln和Pn给定的。n的取值范围为1—64，分别表示1—1/64的全音符，例如，

L1　全音符

L2　半音符

L3　三分之一音符

一个音符的时值，也可以直接写在音符后面，而不用命令Ln。例如，L4A和A4是等效的。又如，C4D4E4C4DE和L4CDECD是等效的，但用Ln命令更为方便。

附点音符仍用“.”表示。一个点表示时值延长50%。例如，“A.”为原长的3/2，“A..”为原长的9/4，“A...”为原长的27/8。

Pn命令表示休止符的时长，其计算方法与Ln命令相同，例如P1是全休止符。

升号和降号应写在时值之前。

乐曲演奏的节拍速度，用命令Tn给定。n的取值范围为32—255。它表示的是每分钟内演奏的四分音符的数量。Tn命令的缺省值为T120。

除Tn命令外，还可以用下列三个命令控制演奏速度：

(1) MN：正常，每个音符的演奏时间是L命令定义的时间的7/8；

(2) ML：连奏，每个音符的演奏时间等于L命令设定的时长；

(3) MS：断奏，每个音符的演奏时间是L命令定义的3/4。

演奏方式分为两种：前台方式（MF）和后台方式（MB）。前台方式是缺省演奏方式。

后台演奏方式不影响前台的正常操作。

和 DRAW 语句一样，在 PLAY 语句中可以使用子字符串命令 X。例如，

```
scale $ = "cdefgab"
PLAY "x" +VARPTR $ (scale $ )
```

上述命令 On，Nn，Pn，Tn，Ln 中的 n 可以是常数，也可以是变量。常数可以直接写在命令之后，而变量则不能这样做。例如节拍速度用 Tempo 变量来表示，应写为：

```
PLAY "T=" +VARPTR $ (Tempo)
```

例 13.4.1 在例13.3.3的时钟陷阱程序中，以前台方式，反复演奏两个八度的音调 "cdefgab" 和一句著名乐句（贝多芬第五交响曲的第一句）。

程序如下：

```
CLS
LOCATE 24, 1
PRINT "Press any key to end."
TIMER ON
ON TIMER (2) GOSUB Display
scale $ = "cdefgab"
listen $ = "T180 02 P2 P8 L8 GGG L2 E-"
Fate $ = "P24 P8 L8 FFF L2 D"
DO WHILE INKEY $ = ""
    PLAY "00 X" +VARPTR $ (scale $ )
    PLAY ">X"+VARPTR $ (scale $ )
    PLAY listen $ + Fate $
LOOP
END
Display:
```

```
    LOCATE 12, 35: PRINT TIME$
RETURN
```

二、音乐事件陷阱——后台演奏

后台演奏需要使用陷阱技术。与之有关的语句有：

```
PLAY ON
PLAY OFF
PLAY STOP
ON PLAY (n)
```

PLAY ON 的功能是允许捕获音乐中断事件，PLAY OFF 是禁止捕获音乐事件。PLAY STOP 是暂停音乐事件的捕获，但一旦执行了 PLAY ON 语句，立即执行事件处理子程序。

当音乐事件被捕获时，即执行 GOSUB 语句，然后，自动执行一个 PLAY STOP，以避免发生死循环。执行 RETURN 时，又自动执行一个语句 PLAY ON。

语句 ON PLAY (n) 语句的格式为：

ON PLAY (n) GOSUB {行号|标号}

它的功能是在程序执行期间以后台方式连续演奏音乐。其中 n 是数值表达式，取值范围在1—32之间，表示要捕获的音符个数。

在后台演奏方式下，音符放入缓冲区，缓冲区的最大容量为32个音符。在程序执行了 PLAY ON 语句后，程序每执行一个新语句，Quick BASIC 都要对音乐缓冲区进行检查，看一看音符是否从 n 变为 n－1，如果是，则执行陷阱处理子程序并将音符依次填入谱子，实现整首乐曲的后台演奏。

例 13.4.2 以后台方式演奏乐曲，演奏的内容是用变量 Lizzie$ 表示。前台执行的是例12.7.5的一个图形程序。程序如下：

```
ON PLAY(3) GOSUB Background
PLAY ON
```

```
Lizzie$ = "03 L8 E D+ E 02 B 03 D c L2 O2 A"
PLAY "MB X" + VARPTR$(Lizzie$)
Plot$ ="BR30 BU25 C1 R54 U45 L54 D45 BE20 P1,1 G20
        C2 G20"
Plot$ =Plot Y+ "R54 E20 L54 BD5 P2, 2 U5 C4 G20 U45
        E20 D45 BL5 P4,4"
Apage%=1: Vpage%=0
Angle%=0
DO WHILE INKEY$ = " "
    SCREEN 7,, Apage%, Vpage%
    CLS 1
    DRAW "TA" +STR$(Angle%)+Plot$
    Angle%=(Angle%+15) MOD 360
    SWAP Apage%, Vpage%
LOOP
END
Background:
    PLAY "MB X" +VARPTR$(Lizzie$)
RETURN
```

小　　结

错误陷阱和事件陷阱，在程序设计中，有着重要的地位。它可以提高程序的可靠性，使程序的输入、输出操作更加灵活，使程序的功能发挥的更加充分，鉴于陷阱技术在程序设计中的重要性，本章对 Quick BASIC 的陷阱技术作了全面介绍。

本章首先介绍了错误陷阱技术的应用和有关语句，给出了程序设计的例子。

其次，介绍了 Quick BASIC 可以捕获的六种事件中的三种，它们是：功能键事件，时间到达事件和音乐演奏事件。这三种事件的陷阱应用较多。

习　　题

13-1　试编写一个处理负数开方的陷阱程序。

13-2　试编写一个程序，令其每隔4秒钟在屏幕上画一个大小不同的圆。

13-3　编写程序，用后台方式演奏音乐（旋律自己设定），同时在屏幕上输出随机数。

参 考 文 献

[1] Microsoft Quick BASIC 4.0 学习 Quick BASIC　　希望电脑技术公司

[2] Microsoft Quick BASIC 4.0 编程专题　　希望电脑技术公司

[3] Microsoft Quick BASIC 4.0 BASIC 语言参考手册　　希望电脑技术公司

(京)新登字 162 号

Quick BASIC 语言程序设计

编　著:张　富
责任编辑:郑　捷

*

北京邮电学院出版社出版
新华书店北京发行所发行　各地新华书店经售
河北高碑店市印刷厂印刷

*

850×1168 毫米　1/32　印张　12.75 字数　329 千字
1994 年 11 月第一版　1994 年 11 月第一次印刷
印数:1—3200 册
ISBN　7-5635-0191-6/TP・17　定价:11.25 元