微型计算机使用指南

户内顺一 著

# 磁盘 BASIC 入门

磁盘 BASIC 入门
磁盘 BASIC 入门
磁盘 BASIC 入门
磁盘 BASIC 入门

化学工业出版社

ISBN 7-5025-0264-5/TQ・216
定　价 5.20 元

微型计算机使用指南

# 磁盘BASIC入门

〔日〕 户内順一 著

王玉珊 译



化学工业出版社

微型计算机使用指南

# 磁盘BASIC入门

〔日〕 户内順一 著

王玉珊 译


化学工业出版社

## 内 容 提 要

本书以磁盘文件为核心，详细介绍了什么是磁盘文件以及如何利用BASIC语言建立磁盘文件，并对其使用方法作了较为详尽的说明。凡属本书应该涉及到的各种用于文件操作的命令、语句及函数，全都作了说明。有关章节列举了程序实例，并辑之以各种图表、画面及框图。本书可作为学习使用PC计算机人员的学习用书，也可供从事计算机管理工作人员参考。

本格的パーソナル・コンピュータ利用への手引き
入门DISK BASIC
户内顺一 著
アスキー出版局 1984年
计算机使用指南
磁盘BASIC入门
王玉珊 译

责任编辑：李诵雪
封面设计：许井立
*
化学工业出版社出版发行
（北京和平里七区十六号楼）
化学工业出版社印刷厂印刷
新华书店北京发行所经销
*
开本787×1092 1/32印张8.9 字数216千字
1990年1月第1版 1990年1月北京第1次印刷
印 数 1—3,300
ISBN 7-5025-0264-5/TQ·216
定 价 5.20 元

# 前　言

电子计算机是当代科学技术发展的最卓越的成就之一。尽管电子计算机从诞生到现在不过几十年的时间，但它却以十分惊人的速度渗透到科学研究、工农业生产、国防建设以及社会生活等各个领域，其应用范围之广，影响力之大，更是前所未有。如今，计算机的科学技术水平、生产规模和应用程度，已经成为衡量一个国家现代化水平的显著标志。

当前在我国，一个学习掌握计算机知识的热潮正在蓬勃兴起，愈来愈多的人们迫切希望能够运用计算机技术来解决各自工作中碰到的实际问题。为此，我们着手翻译了《磁盘BASIC入门》一书。

本书在内容上与一般介绍BASIC语言的书籍有所不同。全书以磁盘文件为核心，详细介绍了什么是磁盘文件以及如何利用BASIC语言建立磁盘文件，并对其使用方法做了较为详尽的说明。凡属本书应该涉及到的各种用于文件操作的命令、语句及函数，全都有所交待。在形式上力求做到系统化和完整化。

本书深入浅出，注重实用。有关章节同时还列举了一些程序实例，并辅之以各种图表、画面及框图。这些程序一般都具有很强的实用价值。

众所周知，计算机技术是提高各行各业管理效能的重要手段，对于需要涉及到大量信息处理的管理工作来说，应用磁盘文件可以大大提高计算机处理的快速性和准确性。我们希望本书能够对读者学习计算机磁盘文件起到一定的帮助作用，并为进一步学习掌握计算机技术打下一定的基础。

阅读本书的读者，应事先学习过基本BASIC语言，同时还应对计算机以及程序设计的一些基础知识有所了解。

在本书的翻译过程中，刘中运、束方珍、吴尚苏等同志做了大量的工作，在此谨向上述同志致以衷心的感谢。

限于译者水平，书中难免有错误和不妥之处，恳请读者批评指正。

译　者

# 序　言

近来，使用PC计算机的辅助存储装置——磁盘的用户正在增加。直至两三年前，磁盘还仅限于为少数用户所有。随着磁盘装置价格的下降，磁盘已开始广泛地普及了。

磁盘与磁带相比速度快，是一种大容量的辅助存储装置，但要使用磁盘必须有软件。本书介绍的磁盘BASIC就是在原有的BASIC中追加了使用磁盘的功能的一个软件。作为使用磁盘的软件，像MS-DOS和CP/M这样的被称做DOS(Disk Operating System) 的软件，得到了很高的评价。磁盘BASIC与这方面的软件相比，虽不能说有很强的功能，但我们能够以与使用原有的BASIC相同的感觉来使用磁盘，并且它还有简便、容易掌握的优点。

为使用磁盘而用的软件中，目前最普及的就是这个磁盘BASIC了。然而许多用户还未能充分有效地利用磁盘BASIC的功能。即使使用了磁盘BASIC，也仅仅是使用程序的装入和存储，其例不胜枚举。到目前为止，系统地讲解有关磁盘BASIC的书一本也没有，大概是其原因之一吧。就连能更详细地说明有关磁盘命令的手册也几乎没有。

本书是一本考虑如何更有效地使用磁盘BASIC的正式入门书。书中对于磁盘的全部命令，系统地边列出易于理解的实例边进行说明。当您读完本书时，就能充分地掌握有效地使用磁盘的基础知识了。本书作为微型软盘系统的磁盘BASIC全面的入门书，是以$N_{88}$磁盘BASIC作为标准来进行说明的。至于与其他机种的不同之处，可参看各命令的说明和附录。

请将本书依次读下去，再实际使用一下磁盘BASIC。作为使您踏入微型计算机世界的入门书，我想本书会起到它应有的作用的。

1984年1月户内顺一

# 目　录

# 1. 绪　　论

## 1.1　使用磁盘的优点

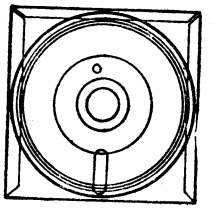
利用磁盘做外存储装置，要比磁带有很大的优越性。现将磁盘的特点进一步加以整理。

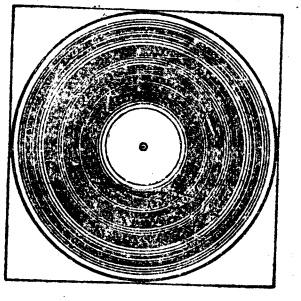
- 数据读写速度快。
- 存储容量大。
- 操作性能好。
- 可靠性高。

数据读写速度，是衡量计算机外存装置性能的重要标志之一。尽管主机处理速度再高，倘若和外部设备打交道的速度很慢，那也没有什么意义。

同磁带相比，对数据的读写速度快，有双重意义。一个意义是能够迅速直接的查寻到所需要的数据。我们用录音带和唱片来打比方，如要想听录音带当中的一首曲子，除了用快速倒带倒到这首曲子的地方之外，只能从头开始一首一首的听，一直听到这首曲子。而唱机，却不必这么浪费时间，我们可以拿起唱针，直接放到曲子前面一点的地方，就可以听了，因而做到了快速选曲。

这是因为两端之间的物理结构不同，而磁盘的形状恰似唱片一样都是圆盘形的。数据读写装置的结构也和唱机极为相仿。





另一个意义是在搜寻到数据后，向计算机内存中进行数据传输，这一点不必多说，大家都知道它要比磁带快得多。

第二个特点是，磁盘具有可观的数据容量，现在微型机所用磁盘的容量大约在100～10000K字节的范围（有关存储容量的详细情况请参阅2.1节）。现假定每一个字节可以记录一个字符，那么约可记录10～1000万字符，这相当于32开纸的140～14000页。一般8位微型机的内存空间为64K字节，相比起来，磁盘的存储容量是足够大了。

光有大容量还不够，还要与高速度相结合，这样就能充分发挥外部存储装置的效能了。假如有的外部存储装置容量是磁盘的几万倍，而读写速度却和磁带差不多，那么可以说，这种装置是没有什么意义的。

操作性能好，这也是能实际应用在微型机上的一个必不可少的重要因素。就拿读取磁带

上的程序来说吧，除了键入装入命令之外，还需要把磁带调到该程序的开头，再去按播放按钮。倘若使用磁盘的话，则只需要输入装入命令就可以了。

经常使用磁带的人都知道，读出，写入、更新数据时，磁带操作相当繁琐。使用磁盘则可免去这些不必要的麻烦。

一般来说，磁盘的可靠性是相当高的。当然，即使是磁盘，也可能出现读写错误，但这种概率要比磁带小得多，所以用户尽可以放心。

其实磁盘的这些优点，也可以说是使用微型机的优点。比如，我们想利用微型机来进行图书管理，就可以借助磁盘来存储大量的图书文献及资料，并对此进行高速而准确的检索，也能够方便地对其中的某些内容进行修改。有时候，还可对磁盘上的内容进行加工处理，从而得到所需要的结果。这一切，都是和磁盘所起的作用分不开的。

目前，有的人可能想利用微型机来建立个人用的数据库；也有的人想在自己的特定业务范围内利用微型机，以提高工作效率；还有的人正在做微型机局部网络方面的工作，以便实现数据共享，提高计算机的使用效率。凡此种种，都需要用磁盘来做外部存储装置。

## 1.2 磁盘BASIC

要想使计算机能胜任各种工作，除了要配置有关的硬件之外，尚须配备相应的软件。有的微型机在接通电源之后，便可立即进入BASIC的命令状态，这是因为机器内部业已装有BASIC解释程序。

使用磁盘的情况也是一样。不仅需要配备能对磁盘进行读写操作的硬件——磁盘驱动装置，而且还需要备有能够对这种装置进行控制以及对数据传输进行管理的软件。

我们把这种包含磁盘管理功能的软件称之为DOS（磁盘操作系统）。前面已经介绍过，从磁盘装入程序时，不必象使用磁带那样，先要找到程序的开头部分，然后再按播放键。这些手工操作，均已做为DOS功能的一部分程序化了。

目前，比较典型的磁盘操作系统有CP/M，MS-DOS，UNIX等。这些系统除掉要对磁盘装置进行控制之外，同时还要完成与磁盘有关的各种管理工作。

本书所介绍的DISK BASIC也是一种磁盘操作系统，虽然它在功能上不如上述三种操作系统那么强，但由于这种操作系统是在以往的BASIC的基础之上加进了磁盘管理功能，因此使用起来就象是使用BASIC一样，这对于那些习惯用BASIC语言设计程序的人员来说，自然是便于学习、易于掌握的了。

本书将具有磁盘管理功能的DISK BASIC称之为磁盘BASIC系统。

磁盘数据的读写，全部都是通过磁盘BASIC系统进行的。磁盘BASIC系统的主要功能是对磁盘命令进行解释以及对磁盘文件进行管理。

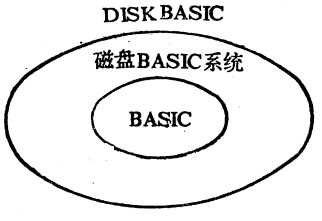



图 1.2.1 磁盘BASIC

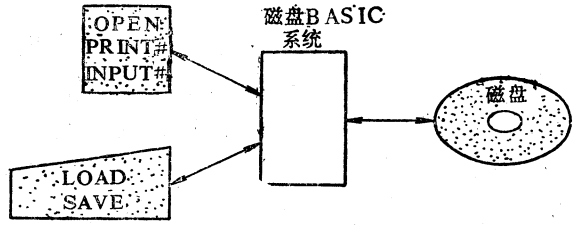



图 1.2.2 磁盘BASIC系统

# 2。使用磁盘BASIC前的准备工作

## 2.1 关于磁盘的基础知识

（1）磁盘的种类

磁盘一般有两种，即软磁盘和硬磁盘。软盘质地柔软，易于弯曲。硬盘——顾名思义是用非常坚硬的材料制成的，与软盘相比，容量要大得多，读写速度快，然而价格相应的也要贵得多。

目前大多数微型计算机都是使用的软磁盘，其规格主要是8英寸标准盘和$5^1/_4$英寸的小盘。最近，有些微型机可以使用规格更小的磁盘，如3英寸，3.5英寸，4英寸盘等，也就是所谓的微型软磁盘。软磁盘有单、双面之分，单面盘只能在一面记录信息，而双面盘则可在正反两面记录信息。此外，根据记录密度的不同，又有单密、双密之分，双密度盘记录的信息容量相当于单密度盘的两倍。(参照表2.1.1)*

最近，配有硬盘的微型机的数量有所增加，硬盘的存储容量比软盘大得多，普通约为5MB～20MB。要想建立一个真正的数据库，可以说是“非磁盘莫属”了。此外磁盘的数据传输速度快，运行声音小，这些特点也都颇具吸引力。当前由于价格比较高，尚未得到广泛的普及。本书涉及到的磁盘，主要是指8英寸盘或5英寸盘。

表 2.2.1 软磁盘的分类

| 种类 | | 容量(K字节) |
|---|---|---|
| 标准软盘（8英寸） | 单面单密度 | 250～300 |
| | 单面双密度 | 500～600 |
| | 双面单密度 | 500～600 |
| | 双面双密度 | 1000～1200 |
| 小型软盘（5英寸） | 单面单密度 | 70～80 |
| | 单面双密度 | 120～160 |
| | 双面单密度 | 120～160 |
| | 双面双密度 | 250～330 |
| 微型软盘(小于$5^1/_4$英寸) | | 32～500 |

（2）磁盘装置

前面曾经介绍过，磁盘装置是用来对盘上的数据进行读写的装置，磁盘装置亦有8英寸软盘和5英寸软盘装置之分。

磁盘装置中，对盘上的数据进行读写的部分叫做驱动器（driver)。驱动器的数目亦有所不同，一般来说，带有两个驱动器的磁盘装置最为多见。最近出现的一些微型机，在主机内部就已配备有一个驱动器。

有的磁盘装置带有两个以上的驱动器，本书分别将其命名为1号、2号…以示区别**。

---

* 最近采用双磁道方式，以求更进一步地高密度化。有关磁道的内容请参看节9.1(磁盘的内部构造)。

** 本书$N_{88}$-BASIC其称呼与此相同。F-BASIC为0号、1号驱动器，CP/M为A、B驱动器，采用这些称呼是为了加以区别。

**（3）磁盘的外观**

软盘是由磁性记录介质构成的，质地柔软外形呈薄盘状，外面有一个四方形的黑色衬套，其间衬有一层柔软的棉纸（参见图2.1.1）。

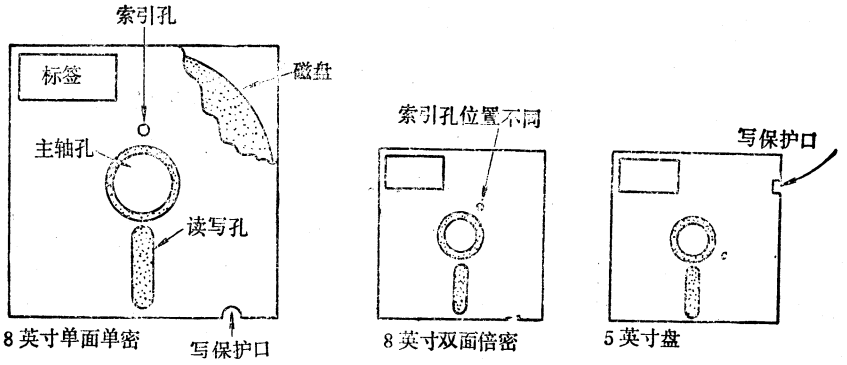



图 2.1.1 软盘的形状

通过磁头的直线运动和磁盘的旋转可以对磁盘上的数据进行读写。读写孔是供磁头进行读写用的窗口，主轴孔用来将磁盘加以固定，并使其旋转。索引孔的作用是供检测出第一扇区*的物理位置，以实现定位。可在标签上做些简要的记录，以备过后查找之用。写保护口起禁止写入的作用。对5英寸盘来说，在写保护口贴上锡箔胶片可以起到写保护作用，而8英寸盘则恰恰相反。

**（4）使用磁盘时的注意事项**

使用得当的话，一张盘可以用好几年。然而由于在一张薄薄的磁盘上面保存了大量信息，所以在使用的时候，应特别注意以下几点：

- 磁盘本身是一种磁性介质，易受其它磁力线的影响，且勿放置在靠近马达、电视机的地方。
- 应避免太阳光的直接照射，亦不适于放在高温潮湿的地方。
- 拿动磁盘时只能接触封套，不要触摸裸露在外面的磁盘表面。
- 小心不要让烟灰粉尘等细小颗粒落入磁盘内部。
- 磁盘应垂直的存放在磁盘柜里，以免受压，造成曲翘变形。

## 2.2 磁盘BASIC系统的装入

为使磁盘能够工作，至少还需要有主机，显示器以及磁盘装置。真要开展事务处理工作，还需要配置打印机。（参照图2.1.1）

将磁盘装置联接到主机上，并不意味着马上就可以使用，这是因为，主机最初只存有BASIC系统，要想使磁盘驱动器工作，则需要装入磁盘BASIC系统。磁盘BASIC一般都是装在一个由制造厂家提供的专用盘中，也就是所说的系统盘。因此，要使磁盘工作，首先必须

* 关于扇区请参看节9.1(磁盘的内部构造)。

从系统盘将磁盘BASIC系统读入主机的内存中*。

一般用下述两种方法，可以将磁盘BASIC系统，从系统盘装入内存。

- 接通主机电源
- 按复位按钮

执行上述任何一种操作，计算机都要访问磁盘装置。如若在磁盘装置中，放有系统盘的话，则将磁盘BASIC系统装入内存。

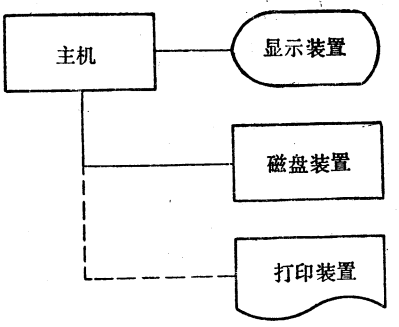



图 2.2.1 使用磁盘BASIC时的基本系统构成

实际操作步骤如下：

① 接通显示器的电源开关

② 接通磁盘装置的电源

③ 将系统盘放入磁盘驱动器中。若有两个以上驱动器的话，一般都是放在1号驱动器中**。

④ 接通主机电源

完成上述操作之后，几秒钟之内，便可自动地将磁盘BASIC装入内存。屏幕上则呈现出画面2.2.1所示的信息（画面为$N_{88}$磁盘BASIC）

```
Disk version [Feb 4,1982]
How many files(0-15)?
```

画面 2.2.1 启动磁盘BASIC时的画面小

计算机向用户提示：「在执行程序时，需要同时使用几个文件缓冲区？」，如果暂时弄不清这句话的含意，也问题不大，后面还要对此作进一步说明。在这里，我们先假定使用3个文件缓冲区。

⑤ 键入3↵

画有强调线的部分为用户键入部分。↵表示回车键。键入3↵，则屏幕显示出画面2.2.2所示的信息，并进入可接受磁盘BASIC命令的状态。（此时屏幕上所显示的信息，为进入磁盘BASIC状态的初始信息）

```
Disk version [Feb 4,1982]
How many files(0-15)? 3
NEC N-88 BASIC Version 1.0
Copyright (C) 1981 by Microsoft
45334 Bytes free
Ok
```

画面 2.2.2 启动磁盘BASIC时的画面（2）

---

* 象T-BASIC7和$N_{60}$扩展BASIC ROM中就具有磁盘BASIC系统均是例外。

** F-BASIC是放在0号驱动器中。

值得一提的是，热启动（即按复位按钮）一般是在下述两种情况下使用的。

·在使用基本BASIC的过程中，想要转入磁盘BASIC。

· 因某种事故，使装入内存中的磁盘BASIC系统受到破坏。

需要提醒大家的是，按复位按钮，也会使内存中存储的内容，全部消失。

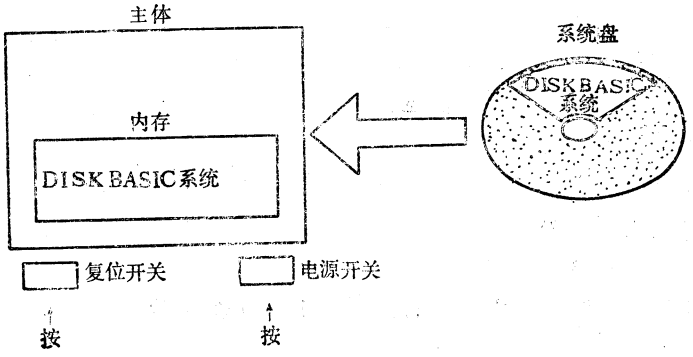



图 2.2.2 装入磁盘BASIC系统

## 2.3 磁盘格式化 (Formatting)

用户刚从商店买来的磁盘，不能马上使用，都需要进行格式化处理。

从软件的角度看，需要将磁盘划分为若干区域，在制作对这些区域的管理表之前，不可能对磁盘记录数据，我们把这种准备工作，称之为格式化，或者叫做初始化。刚刚购入的磁盘一般均未进行过格式化。

格式化一般分为两个阶段

① 物理格式化。

② 系统格式化。

所谓物理格式化是将磁盘划分成若干区段，以使磁盘装置可以对其进行读写，8英寸盘出厂时业已做过物理格式化，但5英寸盘尚未进行过物理格式化。

所谓系统格式化是指对执行过物理格式化的磁盘进行FAT、DIR、ID*的初始化工作。8英寸及5英寸盘均未作过这种格式化。

但也有些厂家出售的专用磁盘，业已作过上述两种格式化。

经过上述两个阶段处理的磁盘，称作数据盘，再复制上磁盘BASIC系统，则可以得到系统盘**。(参照图2.3.1)

由于系统盘备有磁盘BASIC系统，因而利用一张磁盘，即可装入磁盘BASIC系统，又可对磁盘上的数据及文件进行读写，但磁盘BASIC系统占用了一部分盘区，相应地减少了用户可以利用的磁盘空间。

另一方面，由于数据盘未写入磁盘BASIC系统，因而可以扩大用户使用区。但每当接通主机电源或按复位按钮时，均需从系统盘中，重新读入磁盘BASIC系统。

---

* 有关FAT、DIR、ID以及格式化后的磁盘状态请参看9章。

** 厂家提供的系统盘也包含磁盘BASIC的其它实用程序。

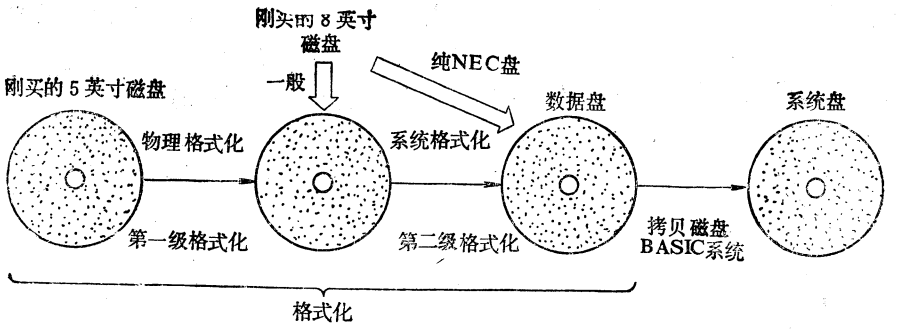



图 2.3.1 磁盘格式化

各种 统的格式化步骤也不尽相同，下面介绍的是$N_{88}$磁盘BASIC的格式化过程。

① 将系统盘插入1号驱动器

磁盘格式化程序**format. $n_{88}$**存放在由厂家提供的系统盘内。因此，第一步工作是将系统盘放入1号驱动器。

② 执行格式化程序**format. $N_{88}$**

首先将系统盘上的**format. $N_{88}$**程序装入内存。即键入

**load"format. $n_{88}$"**↵

**load**是从磁盘中将程序文件取到内存的命令，后面还会作详细的介绍。

然后将读入的程序**format. $n_{88}$**加以执行。

run↵

③ 将新买的磁盘放入2号驱动器

于是屏幕显示出以下的信息：

**Mount a new disk on even drive#**

**Format drive#?**

提示用户，应将新买的磁盘放到编号为偶数的驱动器中。然后，尚应输入该驱动器的编号。在这里，我们假定为2。

④ 进行物理格式化

5英寸盘需要进行物理格式化，输入驱动器编号后，即显示以下信息：

**Do you need physical formatting (y/n)?**

若该盘已作过物理格式化处理，则键入"n"。对刚买的尚未作过物理格式化处理的盘，则键入"y"。一但键入"y"，便立即显示如下信息，开始进行物理格式化。

**Formatting a disk 2**

8英寸盘业已作过物理格式化，故在步骤③输入驱动器编号后，便立即显示⑤所示的信息。

⑤ 制作系统盘

对④的提示信息，键入"n"或物理格式化结束之后，便显示以下提示，

**Create system disk(y/n)?**

提示用户是否建立系统盘，若输入"n"，则只进行系统格式化，不拷贝DISK BASIC系

统即结束格式化处理。

若输入“y”，则在系统格式化之后，紧接着提示用户输入放有系统盘的驱动器编号。

**Working**……系统格式化

**System disk on drwer#?**1↵……输入放有系统盘的驱动器编号。

**Sure(y/n)?**y↵……请用户加以确认

待完成系统格式化后，即从1号驱动器将磁盘BASIC系统拷贝到2号驱动器的磁盘上，即完成了一张系统盘的复制工作。

**Copying system**

**Comleted**

**Ok**

上述操作的全过程，均显示在画面2.3.1和2.3.2

```
run↵
Mount a new disk on even drive#
Format drive#? 2↵
Do you need physical formatting(y/n)? y↵
Formatting a disk 2 ·····物理格式化
Create system disk(y/n)? y↵
Working ·····系统格式化
System disk on drive#? 1↵
Sure(y/n)? y↵
Copying system ·····拷贝磁盘 BASIC系统
Completed.
Ok
```

画面 2.3.1 建立系统盘时的格式化步骤

```
run↵
Mount a new disk on even drive#
Format drive#? 2↵
Do you need physical formatting(y/n)? y↵
Formatting a disk 2 ·····物理格式化
Create system disk(y/n)? n↵
Working ·····系统格式化
Completed.
Ok
```

画面 2.3.2 建立数据盘时的格式化过程

**MOUNT**命令和**REMOVE**命令

使用**N**磁盘**BASIC**时，必须首先执行**MOUNT**命令，方能对磁盘进行读写，取出磁盘之前，一定要执行**REMOVE**命令。

**MOUNT**命令的作用是宣布“已将磁盘装入**X**号驱动器”。磁盘**BASIC**根据此命令，进行一系列必要的准备工作，以便能对指定为**X**号驱动器上的磁盘文件进行读写。因此，在输入**MOUNT**命令之前，不管输入任何命令，机器均不予以受理。**REMOVE**命令的作用是宣布「请从**X**号驱动器中将磁盘取出!」

若要将磁盘从指定的X号驱动器取出，则需使用REMOVE命令。其作用，是将有关信息从内存写入磁盘。否则，该盘不能正确反映执行前述命令所得到的结果。

## 2.4 磁盘副本

最后想谈一下制备磁盘副本的重要性。不管操作人员多么小心，稍不留意还是有可能破坏存储在磁盘上的重要文件*和磁盘的。如果是小程序，重新编制一下倒也无妨，然而一旦破坏了长达数百条、甚至上千条的较大程序时，重新编制就要困难多了。另外，倘若破坏了手中仅有的一张系统盘，那么就什么事情也干不成了。

为防止这种情况发生，需要事先制作文件或磁盘副本。这样，即使文件或磁盘遭到破坏，还可以立即进行再复制。我们把这种复制（copy）称作建立副本（back up）。把利用 Back up建立的文件或磁盘称作副本拷贝（back up copy）。

需要养成定期进行磁盘副本拷贝的习惯，万一发生事故，也可把造成的损失，降低到最小限度。参见图2.4.1

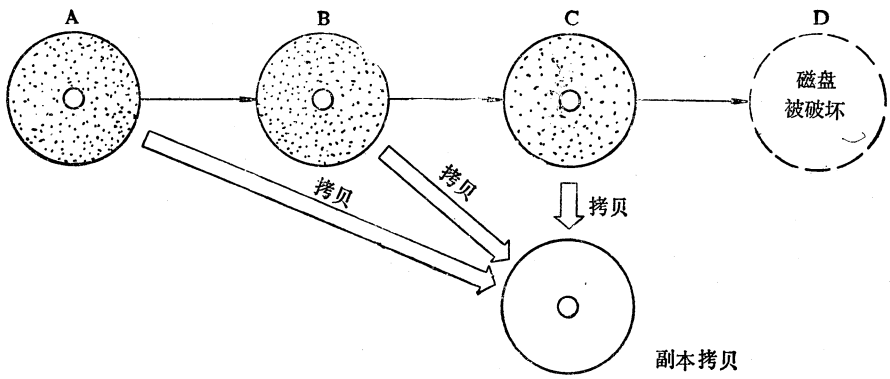



图 2.4.1 磁盘副本

倘若在D处，磁盘内容遭到破坏，如果是坚持了定期的副本拷贝，则只需要重新编制从C点以后的内容就可以了。反之，若没有副本拷贝，那就只好从头开始重新编写了。

下面介绍$N_{88}$磁盘BASIC副本拷贝步骤

① 将系统盘插入1号驱动器

副本拷贝程序**back up. $n_{88}$**存放在厂家提供的系统盘里。

② 执行副本拷贝程序**back up. $n_{88}$**

即把**back up. $n_{88}$**装入内存，并执行。

**load"back up. $n_{88}$"**↵

**run**↵

③ 设置主盘和副本拷贝盘驱动器号

将原来的拷贝盘（主盘）和新准备的待拷贝盘放到驱动器中，然后输入有关驱动器编号。

**Mount master disk on drive**

---

* 关于文件请参看节3.1(什么是文件)。

**Master drive#?1↵** (主盘驱动器编号)

**Mount new disk on drive**

**New drive#?2↵** (副本拷贝盘驱动器编号)

④ 副本拷贝

若副本拷贝用的磁盘未进行过物理格式化，则对下述提示回答"y"，待完成物理格式化之后，便可进行拷贝了。

**Do you need physical bormatting(y/n)?**

回答"n"，则不再进行物理格式化，立即直接进行拷贝。另外，在执行此程序时，若主盘为系统盘，则得到的拷贝盘也是系统盘。

```
Do you need physical formatting(y/n)? y↵
Physical format a disk 2
Sure(y/n)? y↵
Formatting a disk 2 ..... 物理格式化

Copying track 1
Copying track 2
Copying track 3
 . . . . .
Copying track 32
Copying track 33
Copying track 34
Copying track 35
Copying track 36
Copying track 37
Copying track 38
Copying track 39

Completed. ..... 拷贝结束
Ok
```

显示拷贝的磁道编号
{由于已拷贝了FAT、DIR、ID,
{所以不需要系统格式化了

画面 2.4.1 需要进行物理格式化的副本拷贝步骤

```
Do you need physical formatting(y/n)? n↵

Copying track 1
Copying track 2
Copying track 3
 . . . . .
Copying track 36
Copying track 37
Copying track 38
Copying track 39

Completed.
Ok
```

画面 2.4.2 不需要进行物理格式化的副本拷贝过程

# 3. 处理程序的文件

## 3.1 什么是文件

磁盘上存放的各种信息，如程序和数据等，均是采用文件的形式来进行管理的。其实文件这种形式，我们以前使用磁带时就曾碰到过，因而对什么是文件并不感到陌生。但是由于磁盘文件的使用，比起磁带文件来要复杂得多，所以这里有必要再来重新认识一下“什么是文件”。

**（1）磁盘文件**

公司或商店等部门，往往把汇总在一起的收支发票、人事记录以及商品的库存情况等等，分别称作发票文件，人事档案文件和库存文件。由此可见，所谓文件是指，为了某种目的汇集起来的信息的有组织的集合，或者说是按体系将数据汇总在一起的东西。如果把这样一些文件存放在磁盘上而不是放置在办公柜里的话，那么就称作是磁盘文件。

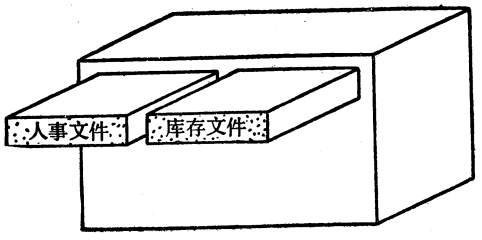



图 3.1.1 放置在文件柜里的文件


图 3.1.2 放置在磁盘上的文件

**（2）文件名**

在磁盘上可以存放许多文件（当然是有限的）。为了识别这些文件，必须要给它们起名字。

文件名用 6 个以内的字符指定。另外，还可以带不超过 3 个字符的扩展名*。

〈文件名〉[.][〈扩展名〉]

一般都是用文件的名字来作文件名，而用扩展名来表示文件的性质，当然并不一定非要如此才行，究竟起什么样的文件名，可以随心所欲自由选择。文件名与扩展名之间的句点可以省略。若指定的文件名超过 6 个字符，则只把前面的 6 个字符看作是文件名，后续的 3 个字符看作是扩展名，而把其余的字符忽略掉。

文件名可以使用的字符有：英文字母，数字，假名，特殊文字及图形符号。但最好不使用引用符，句点，空格等。不可以使用分号和用CHR $(0)，CHR $(255)所表示的字符。此外，请注意英文大小写字母是当作不同的字母来对待的。

---

* F-BASIC文件名在 8 个字符以内，没有扩展名的区别。

表 3.1.1 文件名的处理方式

| 指　　定 | 文　件　名 | 扩　展　名 |
|---|---|---|
| ABCDEF | ABCDEF | — — — |
| file | file | — — — |
| FILE.ABC | FILE | ABC |
| ABCDEF.GHI | ABCDEF | GHI |
| ABCDEFGHI | ABCDEF | GHI |
| アイウエオ | アイウエオ | — — — |
| 123456 | 123456 | — — — |
| abc.def.ghi | abc | def |
| 123456789ab | 123456 | 789 |
| ab.12345 | ab | 123 |

显然，在同一磁盘上不可重复出现同名文件（扩展名也相同）。例如，想要调用A文件时，若在磁盘上有两个以上的A文件，则系统就无法断定到底是要调用哪一个A文件了。

**（3）文件指定**

在调用文件时，由于磁盘驱动器中只能装入一个磁盘，故只要指定出文件名来就可以了。这是因为在一个磁盘上，只存在有一个与指定的文件名相同的文件。但是，一般磁盘装置往往设有两个以上的驱动器，可以同时装入几个磁盘。这时，单单靠指定文件名来调用文件，系统就无法断定应该检索哪一个磁盘。因此，在调用文件时，也应该同时指定放有该磁盘的驱动器编号。

〈驱动器号〉：〈文件名〉、〈扩展名〉

驱动器号的排法因系统不同而异。参见图3.1.3所示的两种配置。

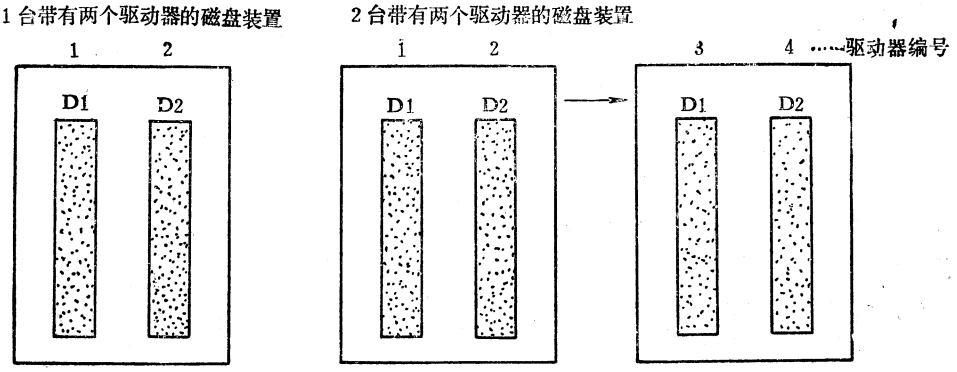



图 3.1.3 驱动器号的排法

我们把这种用驱动器号和冒号（：）所修饰的文件名称作是“文件指定”或“file descripter”，当驱动器为1号驱动器时，驱动器号和冒号可以省略。

**例**

**1：ABC.DE　1号驱动器的ABC.DE文件**

**ABC.DE　1号驱动器的ABC.DE文件**

**2：XYZ　2号驱动器的XYZ文件**

**（4）文件的种类**

根据组成文件的内容的不同，可以将文件划分成三类，即

- 程序文件
- 数据文件
- 机器语言文件

做为程序文件存贮的BASIC程序，是以中间语形式（压缩形式）或ASCII 码形式记录的。数据文件则记录除程序之外的数据。

机器语言文件是指以与主机内存中同样的形式记录的机器语言程序和数据。

## 3.2 什么是程序文件

如前节所述，DISK BASIC将存放BASIC程序的文件称作程序文件。磁盘上的程序文件是供存取程序用的。尽管磁带也可以完成这项工作，但相比之下，磁盘要方便得多。此外，利用程序文件不单是可以存放程序，还可以把两个程序连接起来合成一个程序，或者是连续执行几个程序。

首先介绍一些用于处理程序文件的命令。

## 3.3 存储程序 (SAVE)

通常，在断电以后，内存中的BASIC程序就要消失。为了保存住程序，在断电之前，就必须将程序记录到其他媒体上。我们把这一工作称之为存储程序，使用SAVE 命令可以把程序存储到磁盘上去。

在磁盘上可以用两种形式来存储程序。一种是以中间语形式，另一种是以ASCII 代码形式。

**（1）以中间语形式存储程序**

- **SAVE**〈文件指定〉

BASIC程序在内存中是以特殊形式（中间语形式）存放的。这样做，不仅可以减少程序所占有的存储空间，而且又可以使BASIC解释程序易于解释执行该程序。BASIC程序一般都是以这种中间语的形式存储在磁盘中的。例如，用xfile作程序文件名，以中间语形式把程序存储到磁盘上，可键入命令：

**SAVE "xfile"** ↵

如果磁盘上不存在SAVE命令所指定的那个文件的话，则SAVE命令建立一个新文件，并把BASIC程序存入其中。如果磁盘上业已存有同名文件，则该文件的内容被新存入的程序所更新，原来的程序就不存在了。因此，在存储文件时，要注意不要把文件名给错，稍有疏忽，就会把需要的文件给冲掉*，这种令人遗憾的失误是经常发生的。为了防止这种失误，可事先用SET命令**给文件赋予禁止写入属性，或是经常给文件复制副本。

下面，让我们来实际编制一段简单程序，并以afile作文件名，将其存储到磁盘中去看一看。请将一片已格式化过的磁盘放进 1 号驱动器。

---

* F-BASIC为防止这样的事故，而反回到 "Are you sure (YorN)?"信息。

** 有关SET命令参看节4.3 (属性的设定和解除)。

```
10 for i=0 to 10↵
20   print i↵
30 next i↵
save "afile"↵
Ok
```

画面 3.3.1 以中间语形式存储（1）

然后，再用同一文件名afile存储另一程序。

```
10 for i=1 to 10↵
20   s=s+i↵
30 next i↵
40 print s↵
save "afile"↵
Ok
```

画面 3.3.2 以中间语形式存储（2）

其结果，并未作成两个afile文件。先存入afile中的程序，被后存入的程序覆盖掉了。

**SAVE的例子**

**SAVE "afile" 以中间语形式将程序存放到1号驱动器的afile文件中**

**SAVE "1:afile" 以中间语形式将程序存放到1号驱动器的afile文件中**

**SAVE "2:bfile" 以中间语形式将程序存放到2号驱动器的bfile文件中**

**（2）以ASCII码形式存储程序**

- **SAVE 〈文件指定〉, A**

BASIC 程序也能以文字数据的形式存储到磁盘文件中。例如，如图 3.3.1 所示，可把“PRINT”命令中的字母转换成ASCII代码予以记录。

| P | R | I | N | T |
|---|---|---|---|---|
| 50 | 52 | 49 | 4E | 54 |

图 3.3.1 用ASCII码形式存储例

用ASCII码形式存储文件比用中间语形式存储要占用更多的磁盘空间。尽管如此，这种形式在一些场合却很有用，例如，利用ASCII码文件，可以进行程序的合并(Merge)*.还可以把程序当作数据文件来进行处理。

用ASCII码形式存储程序，只要在文件指定后加上〔a〕即可。例如，将程序存储到bfile文件中，则为

**SAVE "bfile", a↵**

下面，编写一段程序，以ASCII码形式存储到磁盘上去。

* 有关程序的合并参看节3.7(程序的合并)。

```
10 for i=1 to 20↵
20   s=s+i↵
30 next i↵
40 print s↵
save "bfile",a↵
Ok
```

画面 3.3.3 用ASCII码形式存储文件

**SAVE（带A选择）的例子**

**SAVE "bfile", A 用ASCII码形式将程序存储到1号驱动器上的bfile文件中**

**SAVE "1: bfile", A 用ASCII码形式将程序存储到1号驱动器上的bfile文件中**

**SAVE "2: cfile", A 用ASCII码形式将程序存储到2号驱动器上的bfile文件中**

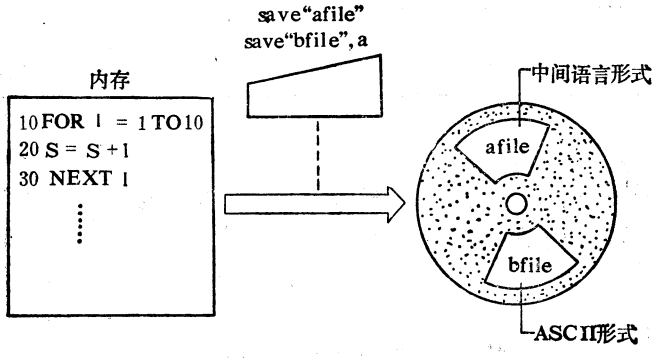



图 3.3.2 SAVE命令

## 3.4 磁盘目录的显示与打印（FILES, LFILES）

在使用磁带文件时，我们必须把已记录在磁带上的文件的名字记在纸上。否则一旦忘了的话，要查找磁带上已有哪些文件就很麻烦了。

然而，对于磁盘来说，由于有了显示或打印磁盘文件目录的命令FILES和LFILES，我们可以随时查看磁盘的内容，因此也就没有必要再在纸上记录文件名了。

**（1）在屏幕上显示磁盘目录**

- **FILES[〈驱动器号〉]**

FILES 命令的作用是在屏幕上显示磁盘上已有文件的目录，若打入

**files 1↵**

则1号驱动器中的磁盘上的所有文件的文件名便显示出来。这里显示的是前节存入的afile和bfile两个文件。

```
files 1↵
afile .     1     bfile       1
Ok
```

画面 3.4.1 FILES命令的执行结果

仔细观察画面3.4.1就会发现afile的后面带有句点，而bfile的后面则为空白。这表明afile和bfile两个文件分别是以中间语和ASCII代码形式记录的。

文件名和扩展名之间的符号不同，表示文件的种类不同，如表3.4.1所示。

表 3.4.1 区分符号与文件种类

| 分隔符 | 文件种类 | 例 |
|---|---|---|
| 句号(.) | 中间语言形式的程序文件<br>(非ASCII形式的文件) | abc.xyz<br>123456. |
| 空白 | ASCII形式的文件<br>数据文件 | abc xyz<br>123456 |
| 星号（*） | 机器语言文件 | abc*xyz<br>123456* |

另外，不管是用星号还是用空格作区分符号，在调用文件时，应一律采用句点（当有扩展名时），若没有扩展名，则不必指定区分符号。

**例**

**abcdef ghi——>load "abcdef·ghi"**

**abc——>load "abc"**

在画面3.4.1中，文件名的右边显示有数字，该数字用来表示文件的大小，其单位为束。如前面的文件 "afile"，便占用了一束。有关束的概念将在第九章介绍，这里只要先记住它是表示文件大小的单位就可以了。

使用FILES命令时，也可以省略驱动器编号，这样则表示，指定的是1号驱动器。如

**files ↵**

这和执行**files 1↵**的效果是一样的。

**FILES的例子**

**FILES 显示1号驱动器中磁盘上的文件目录**

**FILES 1 显示1号驱动器中磁盘上的文件目录**

**FILES 2 显示2号驱动器中磁盘上的文件目录**

**（2）在打印机上打印磁盘目录**

**·LFILES〔〈驱动器编号〉〕**

LFILES 命令在功能上与 FILES 命令非常类似，不同之处仅在于结果是输出在打印机上。

**LFILES的例子**

**LFILES 打印1号驱动器中磁盘上的文件目录**

**LFILES1 打印1号驱动器中磁盘上的文件目录**

**LEILES2 打印2号驱动器中磁盘上的文件目录**

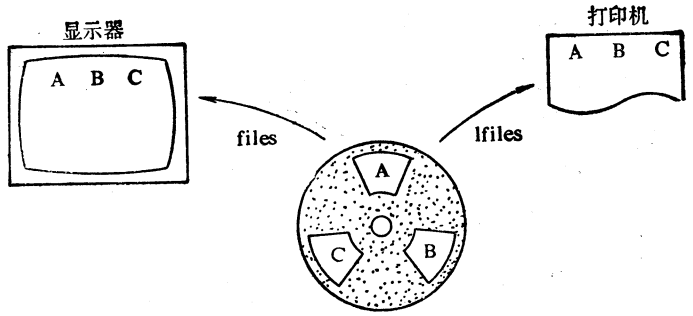



图 3.4.1 FILES命令和LFILES命令

## 3.5 装入程序 (LOAD)

不言而喻，程序要先从程序文件中读到内存后才能执行。将程序从程序文件中读到内存的操作就叫作装入。

装入程序使用LOAD命令。LOAD 命令可将程序从程序文件中读到内存，而且两种以不同形式存储的程序，都用LOAD命令予以装入，程序装入内存后，均以中间语形式存在。

**（1）装入程序**

- **LOAD**〈文件指定〉

例如要装入afile程序，则需按下述方式进行：

**load "afile"** ↵

具体情况见画面3.5.1。再将程序装入内存后，可用LIST命令将程序显示在屏幕上加 以确认。

```
new↵ ·····为了确认正在装入的是新的
Ok
load "afile"↵
Ok
list↵
10 FOR I=1 TO 10
20   S=S+I
30 NEXT I
40 PRINT S
Ok
```

画面 3.5.1 装入用中间语形式存储的文件

画面3.5.1最上面的NEW命令是用来清除内存的。事实上，在用 LOAD 命令装入一个新程序以后，原先在内存中驻留的程序就将不复存在了，因此这里本来是不需要使 用 NEW命令的，之所以用到它纯粹是为了说明，即表示事先已清除了内存。

下面，装入以ASCII代码形式存储的程序bfile。具体做法同前面是一样的。

```
load 'bfile'
Ok
list
10 FOR I=1 TO 20
20   S=S+I
30 NEXT I
40 PRINT S
Ok
```

**画面 3.5.2** 装入用ASCII代码形式存储的文件

怎么样，结果正常吧？可以看出，不管是以中间语形式存储的程序，还是以ASCII代码形式存储的程序，两者均以相同的操作步骤装入内存。

**LOAD的例子**

**LOAD "afile"** 从1号驱动器装入afile

**LOAD "1:afile"** 从1号驱动器装入afile

**LOAD "2:bfile"** 从2号驱动器装入bfile

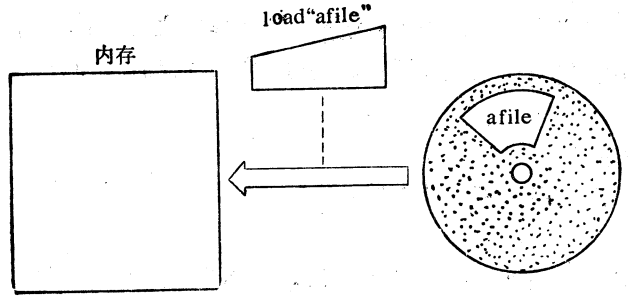



图 3.5.1 LOAD命令

## 3.6 装入程序并立即执行 (RUN)

RUN命令的功能是从程序文件中装入程序，并立即执行该程序。

**（1）装入并执行程序**

- **RUN〈文件指定〉**

例如，为了装入并执行程序afile，可键入命令：

**RUN "afile"** ↵

事实上，上述命令的功能和执行

**LOAD "afile"** ↵

**RUN** ↵

效果完全相同，但可以看出，使用前者要比使用后者来得更方便一些，因为前者一条命令可以同时完成后者两条命令的功能。因此，对于那些不需要修改的程序文件，若使用 RUN 命令在装入后加以立即执行的话，就可以减少打键的次数。

下面，让我们利用程序afile，来实际操作一下。

```
run "afile"↵
 55
Ok
list↵
10 FOR I=1 TO 10
20    S=S+I
30 NEXT I
40 PRINT S
Ok
```

画面 3.6.1 程序的装入和执行

如果在程序中使用RUN命令的话，则可连续执行几个程序。但是，一旦执行RUN命令，前面程序中使用过的变量即全部被清零而无法继续在新程序中使用。这一问题可用程序合并的方法来加以解决，详细介绍请参看第7章（程序结合）。

**RUN的例子**

**RUN "afile"　装入1号驱动器中的afile，并执行**

**RUN "1:afile"　装入1号驱动器中的afile，并执行**

**RUN "2:bfile"　装入2号驱动器中的afile，并执行**

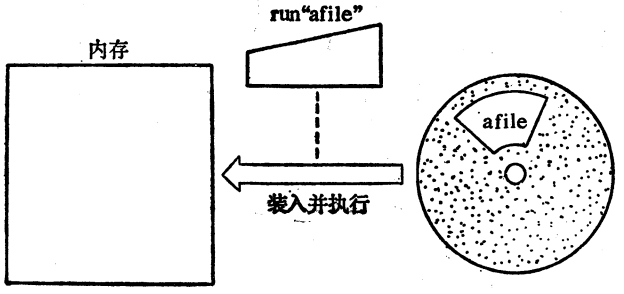



图 3.6.1 RUN命令

## 3.7 程序的合并（MERGE）

在编制大程序时，为了提高效率，通常是将其先分成若干个小程序，分别进行编写和调试，待通过后，再将它们合并成为一个程序。

MERGE命令具有把两个程序连接成一个程序的功能。把程序连接起来叫做合并，也叫做MERGE。

**（1）合并程序**

- **MERGE**〈文件指定〉

将两个程序文件直接合并是不可能的。要合并的两个程序，一个应是已在内存中的程序，另一个则必须是以ASCII代码形式存储在程序文件中的程序。

由于，要合并的程序中，有一个已驻留在内存中，所以在使用MERGE 命令时，只需要指定出另一个待合并的程序就可以了。例如，要求把内存中的程序与 bfile 程序合并，则执行：

Merge "bfile" ↵

上述命令的执行结果将保存在内存之中。

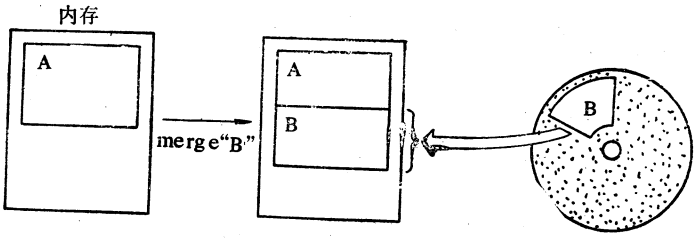



图 3.7.1 MERGE命令

下面，让我们来看一下合并的实际例子，见画面3.7.1。

```
10 print 'b'↵
20 print 'a'↵          } a
30 print 's'↵
save 'mfile',a ·····用ASCII代码存储 上述 程序
Ok
new↵
Ok
30 print 'c'↵
40 print 'i'↵          } b
50 print 'c'↵
merge 'mfile'↵ ·····合并a和b
Ok
list↵
10 PRINT 'b'
20 PRINT 'a'
30 PRINT 's'           } 合并后形成的程序
40 PRINT 'i'
50 PRINT 'c'
Ok
```

画面 3.7.1 MERGE命令的使用例

请注意上面的事例，原驻留在内存程序中的第30号语句被磁盘程序中的30号语句所代替。也就是说，利用MERGE 命令将磁盘上的一个程序和原驻留在内存中的程序合并到一起，实际上是按照行号顺序把磁盘程序插入到驻留程序中去。如果磁盘程序中的行号和驻留程序行号相同，就用磁盘程序中的内容代替原驻留行的内容。因此，要想把两个程序完全合并起来，则须利用RENUM命令对两个程序重新分别编号，使两个程序不存在相同的语句号。

若待合并的程序不是以 ASCII 代码形式存储在程序文件中的话，执行MERGE命令将出现错误信息。

**Sequential I/O only**

**MERGE的例子**

**MERGE "afile"　将1号驱动器内的afile程序和内存中的程序合并**

**MERGE "1:afile"　号1将驱动器内的afile程序和内存中的程序合并**

**MERGE "2:bfile"　将2号驱动器内的bfile程序和内存中的程序合并**

## 3.8 程序文件所用命令汇总

下表所示为前面介绍过的程序文件所用命令的汇总，通过对比可以看出$N_{88}$ DISK BASIC和F-BASIC在命令书写格式上的差别。表中PC表示$N_{88}$ DISK-BASIC的命令书写格式，FM则表示F-BASIC的命令书写格式，在归纳以后各章的命令时，我们也将全部采用这种形式。有关其他DISK BASIC的命令书写格式请参照附录3 (APPENDIX3)。

| | | |
|---|---|---|
| 以中间语言的形式将程序存储到文件中 | PC | SAVE〈文件指定〉 |
| | FM | SAVE〈文件指定〉 |
| 以ASCII代码形式将程序存储到文件中 | PC | SAVE〈文件指定〉,A |
| | FM | SAVE〈文件指定〉,A |
| 装入程序 | PC | LOAD〈文件指定〉 |
| | FM | LOAD〈文件指定〉 |
| 从程序文件中装入程序并立即执行 | PC | RUN〈文件指定〉 |
| | FM | RUN〈文件指定〉 |
| 合并程序 | PC | MERGE〈文件指定〉 |
| | FM | MERGE〈文件指定〉 |
| 在屏幕上显示磁盘目录 | PC | FILES〈驱动器号〉 |
| | FM | FILES"〈驱动器号〉*:" |
| 在打印机上打印磁盘目录 | PC | LFILES〈驱动器号〉 |
| | FM | FILES"〈驱动器号〉*:",L |

* F-BASIC驱动器编号从0开始。

# 4. 文件通用命令

## 4.1 文件的删除（KILL）

尽管磁盘存储容量很大，但它毕竟有一定的限度。而且，每张磁盘上，所能存放的文件个数，也是一定的。有时，会发生这种情况，我们想要把一个新文件存储到磁盘上去，但这时，磁盘上已没有足够的空间，能容纳这个新文件了。因此，如果我们能够确认当前在磁盘上的某些文件已无继续保留价值的话，就应该把它们从磁盘上删除掉，以腾出空间来存放新的文件。KILL命令即是用来供删除文件用的。

**（1）删除文件**

**·KILL〈文件指定〉**

比如，若要删除1号驱动器中的文件afile，则可输入命令：

**KILL "afile"** ↵

或

**KILL "1:afile"** ↵

下面，我们来实际操作一下，先删除afile，然后用FILES命令来检查一下结果。

```
files↵
afile .    1    bfile      1    mfile      1
Ok
kill "afile"↵
Ok
files↵
bfile      1    mfile      1
Ok
```

画面 4.1.1 KILL命令的执行和验证

怎么样，彻底删除掉了吧。请注意文件一旦被删除就不能再恢复，因此使用KILL命令时应该小心，千万不要把文件指定搞错*。

若找不到KILL命令所指定的文件，屏幕上则出现下列错误信息：

**File not found**

```
kill "zfile"↵
File not found
Ok
```

画面 4.1.2 对不存在的文件使用KILL命令时的情况

* F-BASIC在直接方式时，可显示为了检验用的"Are you sure(Yor N)?" 的信息。

此外，若指定的文件或磁盘，是处于写保护*状态时，执行 KILL 命令将出现下列错误信息：

**File write protected**

**KILL的例子：**

**KILL "afile"　删除1号驱动器中的文件afile**

**KILL "1:afile" 删除1号驱动器中的文件afile**

**KILL "2:bfile" 删除2号驱动器中的文件bfile**

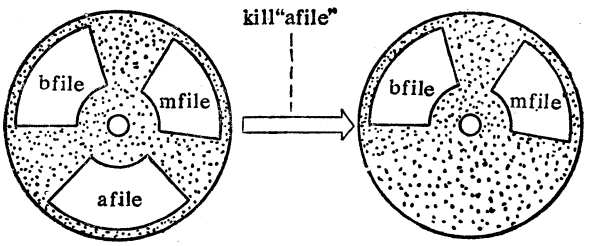



图 4.1.1　文件的删除

## 4.2　更改文件名（NAME）

执行SAVE 命令之后，若发现文件名指定错了或是由于某种原因想要变更文件名时，可以简便地利用NAME命令，进行更改。

**（1）更改文件名**

**• NAME〈旧文件指定〉AS〈新文件指定〉**

可按下述方式将 1 号驱动器中的文件bfile改名为cfile：

**name "1:bfile" as "1:cfile"**

在as之前为〈旧文件指定〉，在as之后为〈新文件指定〉。另外，也可以将上述命令行改写成以下形式：

**name "bfile" as "cfile"**

**name "1:bfile" as "cfile"***

下面，让我们把 1 号驱动器中的bfile改名为cfile，然后用EILES命令加以验证。

```
files↵
bfile      1     mfile      1
Ok
name 'bfile' as 'cfile'↵
Ok
files↵
cfile      1     mfile      1
Ok
```

画面 4.2.1　NAME命令的执行和验证

---

* 有关写保护在节4.3（属性的设定与解除）中将详细说明。

* 在作name "2:bfile"as"cfile" 时，其功能如何，取决于BASIC。N-BASIC将新文件指定解释为 "2:cfile"；N$_{88}$-BASIC解释为"1:cfile"，则显示Rename across disks。

若找不到〈旧文件指定〉所指定的文件bfile，则显示错误信息：

**File not found**

反之，倘若做为〈新文件指定〉的cfile已经存在于磁盘上的话，则显示错误信息：

**File already exists**

**NAME的例子**

**NAME "afile" As "1:bfile"　将1号驱动器中的afile改名为bfile**

**NAME "1:afile" As "1:bfile"　将1号驱动器中的afile改名为bfile**

**NAME "1:afile" As "bfile"　将1号驱动器中的afile改名为bfile**

**NAME "2:xfile" As "2:yfile"　将2号驱动器中的xfile改名为yfile**

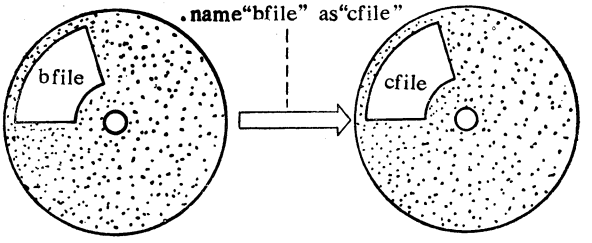



图 4.2.1　文件名的更改

## 4.3　属性的设定与解除（SET）

尽管我们总是努力设法避免对文件的误操作，但发生各种误操作的可能性总是存在的，比如由于我们一时的精神不集中，可能会把某个很重要的文件删除掉。

为了有效地避免对文件的误操作，可以给文件设置保护特征，我们称这种保护特征为属性，可以给文件设定以下两种属性

- 写保护属性
- 写检验属性

如果给文件赋予了写保护属性，就不可能再进行文件删除、数据写入等文件修正工作了。因此，也就不可能再出现误删除、误改写文件的错误了。

如果给文件赋予了检验属性，则在将数据写入文件之后，立即再把数据读出，并与被写入文件的数据进行比较，用以检验写入文件的数据是否正确。给文件赋予这种属性，是为防止磁盘读写装置由于偶然因素发生变动，而不能写入正确数据。尽管这种情况一般很少出现，但为了防止万一起见，采取这种措施有时还是很必要的。

对文件进行属性设定，使用SET命令。可以利用以下三种设定方式：

- 通过文件指定的设定
- 通过文件编号的设定（编号应指Buffer号）
- 通过驱动器编号的设定

下面按顺序分别予以介绍。

**（1）用文件指定进行属性的设定和解除**

- **SET**〈文件指定〉，〈属性字〉

利用上述命令可将〈属性字〉所指示的属性赋予〈文件指定〉所指定的具体文件，不同种类属性字的选择请参照表4.3.1。注意，属性字必须用大写字符。

表 4.3.1 属性字和属性种类

| 属性种类 | 属 性 字 | 属 性 概 要 |
|---|---|---|
| 写保护 | P（大写） | 禁止对文件进行写入 |
| 写检验 | P（大写） | 写入后立即进行读出，检验写入的内容是否正确 |

例如，若给1号驱动器中的文件cfile赋予写保护属性，则打命令：

**set "cfile", "P"**

**或**

**set "1:cfile", "P"**

倘若，我们真的给cfile文件赋予写保护属性，那么就已经无法再利用 KILL 命令删除掉这个文件了。

```
set 'cfile','P'↵ ..... 对文件cfile 设定写保护属性
Ok
kill 'cfile'↵
File write protected ..... 不可能删除 cfile
Ok
```

画面 4.3.1 利用文件指定对文件进行写保护属性设定并加以验证

但是，即使设定了写保护属性，文件名仍然是可以更改的。

```
name 'cfile' as 'bfile'↵
Ok
files↵
bfile      1    mfile
Ok
```

画面 4.3.2 在设定了写保护属性后更改文件名

另外，不能同时对一个文件设定两种属性。如果对一个文件的属性，进行多次设定的话，则总是以最后设定的属性为有效设定。如画面4.3.3所示，在对 bfile 连续进行了两次不同的属性设定以后，起作用的是最后设定的R(写检验) 属性。

```
set 'bfile','P'↵
Ok
set 'bfile','R'↵
Ok
```

画面 4.3.3 利用文件指定多次设定文件属性

解除文件所具有的属性同样是使用SET命令。这时，只要用P及R以外的字符作属性字就可以了。例如要解除文件bfile的写入属性，可以采用下列任意一种方法：

**set "bfile", "p"**

**set "bfile", "Y"**

**set "bfile", "Z"**

请考虑这个建议：重要的文件最好是赋予写保护属性。

**SET的例子（1）**

**SET "afile", "P" 对1号驱动器中的afile赋予写保护属性**

**SET "1:afile", "R" 对1号驱动器中的afile赋予写检验属性**

**SET "2:xfile", "P" 对2号驱动器中的xfile赋予写保护属性**

**SET "afile", " " 解除1号驱动器中afile的属性**

**（2）用文件编号进行属性的设定和解除**

**·SET#〈文件编号〉,〈属性字〉**

利用文件编号也可对文件进行属性设定*。例如可按下述方式对文件编号为1的文件进行写检验属性指定。

**set#1, "R"**

请参看下面的程序。

**程序表 4.3.1 利用文件编号进行属性设定**

```
10 OPEN 'xfile' FOR OUTPUT AS #1
20 SET #1,'R'
30 PRINT #1,10,20,30
  .
  .
  .
100 CLOSE #1
```

由于，将文件名与文件编号建立对应关系的OPEN(打开文件）命令，通常都是在程序中使用，因此，利用文件编号进行文件属性设定的SET命令，一般也是在程序中使用。

在上面一段程序中，首先在10号语句用OPEN命令将1号文件缓冲区分配给xfile，这样xfile与文件编号1便建立了对应关系，因此20号语句中对1号文件编号所进行的写检验属性设定，实际上也就是对xfile进行的设定。30号语句是把数据写入到文件xfile中去，此时每写进一个数据后都要再将其读出，用以检验写入是否正确。

解除属性的方法，仍然是用P和R以外的字符来作属性字，例如：

set#1, "X"

除此之外，执行CLOSE命令，也可以解除利用文件编号所进行的文件属性设定。

**SET的例子（2）**

**SET#1, "R" 对文件编号为1的文件进行写检验属性设定**

**SET#2, "P" 对文件编号为2的文件进行写保护属性设定**

**SET#1, "X" 解除对文件编号为1的文件的属性设定**

**（3）用驱动器号进行属性的设定和解除**

---

* 有关文件编号将在节5.3（顺序文件的建立）中详细说明。

·**SET**〈驱动器号〉，〈属性字〉

有时候有必要对一张盘上的所有文件进行保护。利用SET命令可以对处于特定驱动器中的磁盘进行属性设定。这种设定即可将同一磁盘上的全部文件赋予相同的属性*。

例如，欲对1号驱动器中的磁盘，设定写保护属性则作法如下：

**set** 1, "P"

若想删除已被设定为具有写保护属性的，磁盘上的一个文件，则显示出错误信息。

```
set 1,"P"↵ ……对1号驱动器的磁盘设定写保护属性
Ok
kill "bfile"↵ ……不能删除1号驱动器的bfile文件
File write protected
Ok
```

画面 4.3.4 利用驱动器号进行磁盘写保护属性设定并加以验证

另外，在对同一驱动器中的磁盘进行多次属性设定时，总是最后设定的属性有效。如画面4.3.5所示的设定中，即以R（写检验）属性为有效。

```
set 1,"P"↵
Ok
set 1,"R"↵
Ok
```

画面 4.3.5 多次对同一驱动器中的磁盘进行属性设定

我们还可以对同一文件进行双重属性设定，方法如画面4.3.6所示。在那里，即使用了文件指定又使用了驱动器号指定，其结果，使得1号驱动器中的bfile文件同时具有了写保护和写检验两种属性。

```
set 1,"P"↵
Ok
set "bfile","R"↵
Ok
```

**画面 4.3.6** 利用文件指定和驱动器号指定进行双重属性设定

磁盘的写保护，除了可以利用下面的命令

**set**1, "P"

之外，尚可采用在磁盘的写保护缺口处贴上锡箔的方法（仅对五英寸磁盘而言）。

解除磁盘属性，同样也是利用SET命令，只不过是用P和R以外的字符来作属性字罢了，例如，执行下述命令

**set**1, "Q"

即可解除1号驱动器中磁盘的属性。

---

* 以驱动器号设定属性时，就是指定写保护属性，也必须加以注意，因为对已存在的数据文件有可能更改。

此外，对于用MOUNT，REMOVE方式的系统（PC-8001)来说，也可以用REMOVE命令来解除磁盘属性。

**SET的例子（3）**

**SET1，“P” 对1号驱动器中的磁盘设定写保护属性**

**SET2，“R” 对2号驱动器中的磁盘设定写检验属性**

**SET1，“p” 解除1号驱动器中的文件属性**

**SET2，“ ” 解除2号驱动器中的文件属性**

下面通过表4.3.2对SET命令进行归纳。

表 4.3.2 SET命令

| | 属性设定 | 属性解除 |
|---|---|---|
| 文件指定（对文件） | SET〈文件指定〉，{“R” “P”} | SET〈文件指定〉，“X” |
| 文件编号指定（对用OPEN打开的文件） | SET〈文件编号〉，{“R” “P”} | SET〈文件编号〉，“X”<br>CLOSE#〈文件编号〉 |
| 驱动器指定（对磁盘） | SET〈驱动器号〉，{“R” “P”} | SET(驱动器号)，“X”<br>REMOVE(MOUNT，REMOVE方式时) |

**X**表示除**R**和**P**以外的字符（包括空格）

## 4.4 回答属性的函数(ATTR$)

我们已经知道，可以利用FILES命令来显示文件目录，但利用该命令却无法显示文件的属性，一旦我们忘记了某文件的属性时，我们有办法检查它吗？

回答是肯定的。在这种场合下ATTR $ 函数将会给我们提供帮助。我们可以通过该函数带回的值来获得文件的属性。请参看表4.4.1

表 4.4.1 ATTR $ 函数的带回值

| 属性 | 用ATTR $ 带回的值 |
|---|---|
| 写保护 | ␣␣P |
| 写检验 | R␣␣ |
| 无属性 | ␣␣␣ |

␣表示空格

与设定属性的SET命令相对应，用ATTR $ 函数获得属性同样也有三种方式。即

- 通过文件指定获取属性
- 通过文件编号获取属性
- 通过驱动器号获取属性

**（1）用文件指定得到属性**

- **ATTR $**（〈文件指定〉）

把文件指定做为ATTR $ 函数的带入值，则其返回值即反映了该文件的属性。例如，为

了得到1号驱动器中的afile文件的属性，可调用函数

**attr $ (“afile” )**

画面4.4.1表示，在对文件bfile赋予了写检验属性之后，利用 ATTR $ 函 数显示其属性值。

```
set 1,"P"↵ ..... ..... 设定写保护属性
Ok
print attr$(1)↵
 P
Ok
```

画面 4.4.1 用文件指定得到属性值（1）

结果显示的是R，说明已对bfile赋予了写检验属性，当然若赋予的是写保护属性，则显示P，如画面4.4.2所示。

```
set "bfile","P"↵ ..... ..... 设定写保护属性
Ok
print attr$("bfile")↵
 P
Ok
```

画面 4.4.2 用文件指定得到属性值（2）

**ATTR $ 的例子（1）**

**ATTR $ (“afile”) 获得1号驱动器中afile的属性**

**ATTR $ ( “1:afile” )获得1号驱动器中afile的属性**

**ATTR $ (“2:xfile” ) 获得2号驱动器中xfile的属性**

**（2）用文件编号得到属性**

**• ATTR $ (#〈文件编号〉)**

对已用OPEN命令打开的文件来说，用赋予该文件的文件缓冲区编号，做为带入值来调用ATTR $ 函数，即可得到该文件的属性。例如可用下述方式得到 1 号文件的属性。

**attr $ (#1)**

我们再通过下面的一段程序来进一步看一下。

```
10 open "xfile" for output as #1↵
20 set #1,"R"↵ ..... 设定写检验属性
30 print attr$(#1)↵
40 set #1,"x"↵ ..... 解除属性
50 print attr$(#1)↵
60 close↵
run↵
R
 
Ok
```

画面 4.4.3 利用文件编号来获得属性

我们看到，程序中30语句的执行结果显示出R，这是由于在20号语句中对1号文件设定了写检验属性，接着在40号语句中又解除了属性，因此，执行50号语句就只显示出空格了。

**ATTR $ 的例子（2）**

**ATTR $（#1） 得到1号文件的文件属性**

**ATTR $（#2） 得到2号文件的文件属性**

**（3）用驱动器号得到属性**

**· ATTR $（〈驱动器号〉）**

用磁盘驱动器号做ATTR $ 函数的带入值，即可得到在该驱动器中的磁盘所具有的属性。例如，下面的调用即可得到1号驱动器中磁盘的属性。

attr $ (1)

画面4.4.4表示，在设定1号驱动器中的磁盘以写保护属性后，再显示其属性值的情况。

```
set "bfile","R"↵ ..... 设定写检验属性
Ok
print attr$("bfile")↵
R
Ok
```

画面 4.4.4 用驱动器号得到属性（1）

显示结果为P，这表明处于1号驱动器中的磁盘已被赋以写保护属性。当然，如果设定的是写检验属性的话，则显示R，如画面4.4.5所示。

```
set 2,"R"↵
Ok
print attr$(2)↵
R
Ok
```

画面 4.4.5 用驱动器号得到属性（2）

**ATTR $ 的例子（3）**

**ATTR $ (1) 得到1号驱动器中磁盘的属性**

**ATTR $ (2) 得到2号驱动器中磁盘的属性**

## 4.5 文件通用命令的汇总

下表所示为文件通用命令的汇总

| | | |
|---|---|---|
| 从磁盘中删除文件 | PC | KILL〈文件指定〉 |
| | FM | KILL〈文件指定〉 |
| 更改文件名 | PC | NAME〈旧文件指定〉AS〈新文件指定〉 |
| | FM | NAME〈旧文件指定〉AS〈新文件指定〉 |
| 通过文件指定对文件设定属 | PC | SET〈文件指定〉,〈属性字〉 |
| 性或解除属性 | FM | — |
| 通过文件编号对文件设定属 | PC | SET#〈文件编号〉,〈属性字〉 |
| 性或解除属性 | MM | — |
| 通过驱动器号对文件设定属 | PC | SET〈驱动器号〉,〈属性字〉 |
| 性或解除属性 | FM | — |
| 获得指定文件的文件属性 | PC | ATTR$(〈文件指定〉) |
| | FM | — |
| 获得指定文件号所对应的 | PC | ATTR$(#〈文件编号〉) |
| 文件的文件属性 | FM | — |
| 获得指定驱动器中的文件的文件属性 | PC | ATTR$(〈驱动器$〉) |
| | FM | — |

P表示写保护属性；R表示写检验属性；其它字符表示解除属性。

# 5. 处理数据的文件〔1〕
## ——顺序文件——

## 5.1 什么是数据文件

我们已经知道，程序文件中记录的是程序，而对于诸如电话号码，住址及姓名等这样一些数据信息，若把它们记录在文件中的话，这种文件就叫做数据文件。在数据文件中，构成数据的字符和数字均是以ASCII代码的形式记录的。例如，图5.1.1表示了「ヤマダ,608-1456」这个数据在数据文件中的记录形式。

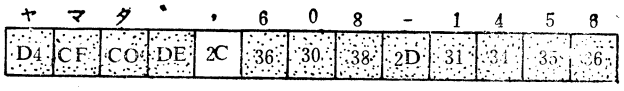



图 5.1.1

数据文件根据其记录形式的特点和存取方式的不同可分为顺序文件和随机文件两种。

数据文件 { 顺序文件 / 随机文件 }

只能从文件的开头按物理顺序进行数据读写的文件叫做顺序文件。不用考虑物理顺序，可随意进行数据读写的文件叫做随机文件。

例如，一些数据如图5.1.2所示，排列在一起。

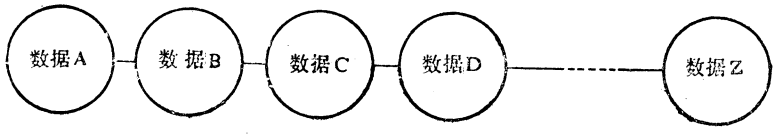



图 5.1.2

为了读出数据D，对顺序文件来说，必须从数据A开始按顺序读到D，如果在读出D后又要读C，仍须再一次从头开始读到C。而对随机文件来说，则可跳过A、B、C直接读取D，接着再直接读取C。

不能简单地断言顺序文件和随机文件究竟哪一个更好，而应根据不同的使用目的来加以选择。一般说来，需要按顺序进行数据读写的文件或是变动较少的文件，最好采用顺序文件方式；需要随机进行数据读写的文件或是经常要变动内容的文件，则最好采用随机文件方式。

## 5.2 使用顺序文件前的准备

在实际建立和使用顺序文件之前，先要做一些准备工作。

**（1）且莫忘记打开和关闭文件**

我们不妨把文件看做是存放数据和程序的袋子，而且假定这个袋子通常是关着的。因此，当要向袋子中装入数据或是从袋子中取出数据时，首先必须将袋子打开。而在装入数据或是取出数据之后，还应照原样把袋子关好。

我们称打开袋子（文件）的操作为「打开文件」或「OPEN文件」，称关闭袋子（文件）的操作「关闭文件」或「CLOSE文件」。

实际上，打开文件意味着进行读写数据前的各种准备工作，而关闭文件则意味着对文件的读写操作业已结束。

存取程序文件时，由系统自动打开或关闭文件；读写数据文件时，则须由用户自己来打开或关闭文件。

综上所述，读写数据文件的步骤可归纳为：

① 打开数据文件

② 读写数据文件

③ 关闭文件

**（2）处理方式**

读写顺序文件时，必须指定处理形态（方式），共有3种处理方式：

- 建立新文件，顺序写入数据（OUTPUT方式）
- 从头开始顺序读取已存在文件的数据（INPUT方式）
- 在已存在文件的末尾追加数据（APPEND方式）

对顺序文件来说，可指定上述三种方式中的任何一种。但要注意，不可对已指定为OUTPUT方式的文件读取数据，也不可对已指定为INPUT方式的文件写入数据。

对文件的处理方式是在打开文件时指定的。因此，顺序文件的操作顺序应为下面三种情况之一：

**建立文件并顺序写入数据：**

1．用OUTPUT方式打开文件

2．将数据写入文件

3．关闭文件

**读取数据：**

1．用INPUT方式打开文件

2．从文件中读取数据

3．关闭文件

**追加数据：**

1．用APPEND方式打开文件

2．追加数据到文件末尾

3．关闭文件

## 5.3 顺序文件的建立

我们先来介绍建立顺序文件的步骤，并通过这些介绍来学习按OUTPUT方式处理顺序文件的方法。

**（1）建立电话簿文件**

程序表5.3.1即为用顺序文件的形式建立电话簿文件的程序。

**程序表 5.3.1 建立顺序文件形式的电话簿文件的程序**

```
10 OPEN "tel" FOR OUTPUT AS #1
20 FOR I=1 TO 3
30   READ NM$,TEL$
40   PRINT #1,NM$;",";TEL$
50 NEXT I
60 CLOSE #1
70 DATA ﾅﾑﾗ,630-2121,ｻｲﾄｳ,291-3784,ﾐﾔﾓﾄ,333-7128
```

让我们来详细分析一下上面的程序。

**打开文件（OUTPUT方式）**

- **OPEN〈文件指定〉FOR OUTPUT AS#〈文件编号〉**

打开文件使用OPEN命令，参见10号语句。

**10 OPEN "tel" FOR OUTPUT AS#1**

这条语句的功能是用OUTPUT方式打开文件tel，并将文件tel的文件编号设定为1。因为是按OUTPUT方式打开文件的，所以tel是一个新文件。假如原来已存在有以tel命名的文件的话，那么执行10号语句便会将该文件删除*。前面出现了“文件编号”这种叫法，这只不过是用文件编号这种称呼来代替文件名那种称呼罢了，就象用称呼“5号先生”来代替称呼“山田先生”一样。文件编号是和文件相对应的。为了区分文件编号和驱动器编号，通常都是在文件编号前面记上一个#符号。事实上，文件编号恰好是分配给该文件的文件缓冲区的编号，有关缓冲区的介绍请参看节5.4（顺序文件的进一步阐述）。

另外，以OUTPUT方式打开文件之后，文件指针（后述）是指向这个新建文件的开头部位的。

**数据写入**

- **PRINT#〈文件编号〉，〈输出名表〉**

使用PRINT#命令即可将数据写到文件中去。PRINT#命令需要指定出文件编号和数据输出名表。请参看40号语句。

**40 PRINT#1,NM$；“, ”；TEL$**

上述语句将指定的数据（名字和电话号码）写入1号文件，即文件tel。

我们已经知道，用OUTPUT方式打开的文件，数据是从开头按顺序写入文件的。因此在这里，数据是按图5.3.1所示的样子，写入文件tel的。

用PRINT#命令将数据写入文件后，文件指针总是移到刚被写入的数据的后面，使得数

*F-BASIC显示“File Already Exists”，意为错误。

据写入总能按顺序进行。

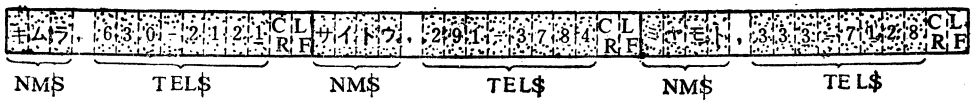



图 5.3.1

在图5.3.1中$C_R$(回车)和$L_F$(换行)是PRINT＃命令自动加在数据输出名表后面的。

**关闭文件**

• **CLOSE〔＃〈文件编号〉，…〕**

CLOSE命令的功能是关闭由文件编号所指定的文件。

**60 CLOSE＃1**

执行60号语句，便可关闭1号文件，即文件tel。

在CLOSE命令中，即可同时指定几个文件编号，又可省略掉文件编号。如果省略掉文件编号，即将已经打开的文件全部关闭。

**CLOSE的例子：**

**CLOSE＃2　关闭2号文件**

**CLOSE＃1，＃2　关闭1号和2号文件**

**CLOSE　关闭全部文件**

**程序的执行**

执行上述程序，则在磁盘中建立tel文件，如画面5.3.1所示。

```
run↵
Ok
files↵
bfile     1     mfile     1     tel     1
Ok                                  新建文件
```

画面 5.3.1　文件的建立和确认

**（2）OPEN/CLOSE命令的作用**

前面，我们曾经形象地把OPEN比做是打开袋子，把CLOSE比做是关闭袋子。这里，我们再进一步对这两个命令的作用加以归纳。

OPEN命令具有如下作用：

- 设定文件的处理方式。
- 建立文件名和文件编号的对应关系。
- 分配和文件编号相同编号的缓冲区。
- 完成其它的准备工作

CLOSE命令具有如下作用：

- 解除文件名与文件编号之间的对应关系
- 释放分配给文件的缓冲区

·完成其它收尾工作

因此，执行CLOSE命令，可以使曾被前面文件使用过的文件编号和缓冲区，空出来以供其它文件使用。

**（3）数据的写入形式**

我们已经知道，用OUTPUT方式打开文件之后，利用PRINT＃语句可以将数据写入文件。这里我们将详细描述一下，通过PRINT＃语句写入的数据，是以怎样的形式被记录在磁盘上的。

**用分号（；）分隔数值数据**

**PRINT＃1，10；-5；6**

当以分号来做数值数据的分隔符时，数据在被写入文件后，在数字的前面形成了符号，在数字的后面形成了做为区分标记的空格。如果数值为正，则符号表现为空格，如图5.3.2所示。

**用分号(；)分隔字符数据**

**PRINT＃1,"ABC"；"DEF"**

与数值数据不同，当以分号来做字符数据的分隔符时，数据在被写入文件后，在其前后均不产生任何标记，如图5.3.3所示。

用分号（；）分隔混合在一起的数值数据和字符数据。

**PRINT＃1,10；"AB"；"XY"；－5**

当以分号来做混合在一起的数值数据和字符数据之间的分隔符时，数据如图5.3.4所示。

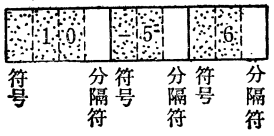



图 5.3.2

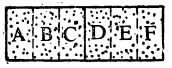



图 5.3.3

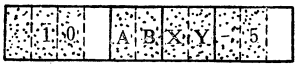



图 5.3.4

**用逗号(,)分隔数据**

**PRINT＃1,10,"AB",－5**

当用逗号分隔数据时，数据之间将被插入额外的空格，如图5.3.5所示。


图 5.3.5

**使用WRITE＃语句**

除了**PRINT＃**以外，还有**WRITE＃**命令也可以把数据写入到顺序文件中去。

**WRITE＃**语句能够自动地在各个数据之间写入一个分隔符","，同时删除掉多余的空格。例如：语句 **WRITE＃**1,10；"AB"；"XY"；－5的执行结果，系将数据按下面的形式记录

若上述**WRITE**#语句中数据不是用分号而是用逗号加以分隔的话，即为**WRITE**#1，10，“**AB**”,“**XY**”，－5，其执行结果仍然是相同的。


在多数情况下，使用**WRITE**#语句具有命令简单、节省文件空间等优点。但该语句并不是对所有的**DISK BASIC**全都适用，因此，在本书所列的实例当中，主要采用**PRINT**#语句。

**使用PRINT#，USING语句**

使用**PRINT**#，**USING**语句的指定格式，能够更精细地控制写入数据的格式。例如，语句为

**PRINT**#1，**USING**“￥￥####.#”；123.456；34.6

则其记录的格式如图5.3.6所示。


图 5.3.6

## 5.4 顺序文件的进一步阐述

在进一步阐述顺序文件之前，让我们先来学习一下有关数据文件的三个重要概念，这就是“记录”，“文件指针”和“文件缓冲区”。

**（1）什么是记录**

对于顺序文件来说，我们把用回车符和换行符分隔开来的一组数据叫做一个记录，而把记录当中用逗号或空格符分隔开来的各个数据叫做数据项。参照图5.4.1

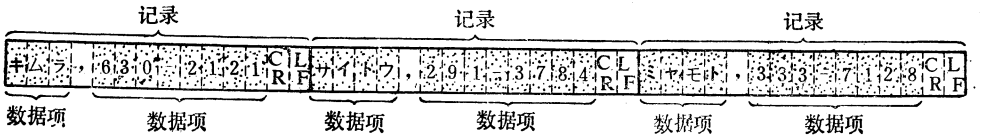



图 5.4.1

可以这样说，文件是记录的集合，而记录又是相关数据项的集合。

由于顺序文件的数据项长度参差不齐，相应地，顺序文件的记录长度也就「因“记”而异」了。我们把记录长度不定的记录形式称作是可变长记录形式。

**（2）文件指针**

在系统中具有一个总是指向文件的下一个将被读（或写）的数据项的指针。该指针称作是文件指针。

文件指针是在打开文件时按打开文件的方式设定的。按 INPUT和OUTPUT方式打开的文件，文件指针指向文件的开头，按 APPEND 方式打开的文件，文件指针指向文件的末尾。

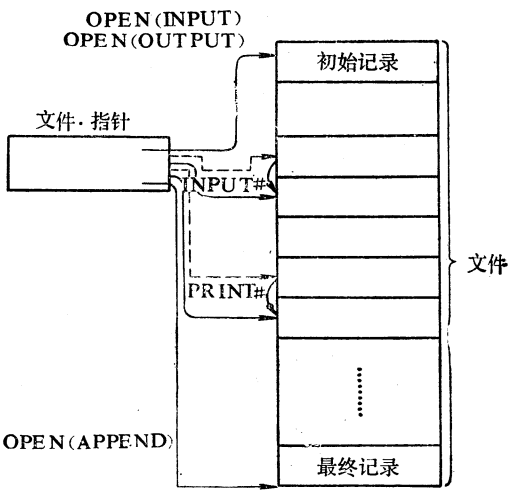
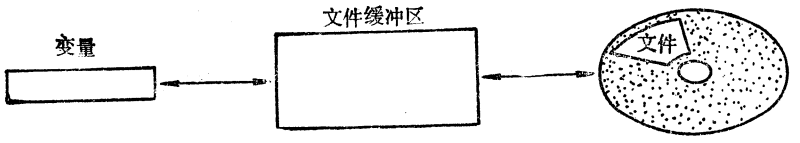



图 5.4.2 文件、指针

另外，随着每一次读写数据操作的完成，文件指针所指向的位置都要作相应的改变，用以指向新的位置。




见 5.4.3 访问文件时，途经文件缓冲区

**（3）文件缓冲区**

虽说对磁盘数据进行读写操作的速度比对磁带要快得多，但仍然比不上对内存的读写速度。为了协调磁盘和内存之间在数据读写速度上的不一致，就需要设置一个旨在调整二者之间数据传输节奏的东西，这就是文件缓冲区。因此，我们在读写数据文件中的数据时，被读写的数据肯定要经过这个被称作文件缓冲区的中间区域的。

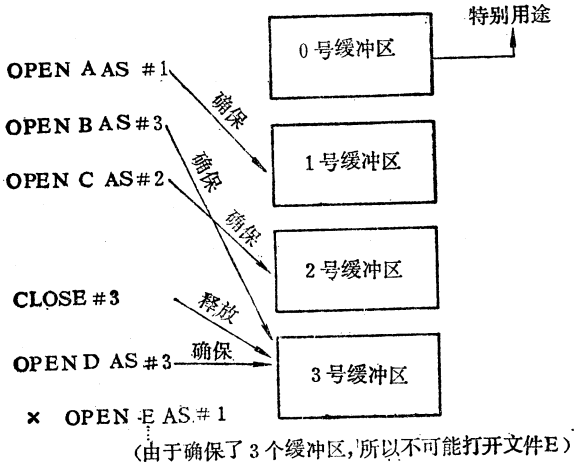



图 5.4 4 要为同时使用的每个文件确保一个缓冲区

还记得从系统盘上装入DISK BASIC 系统时，屏幕上出现的提示信息吗？即

**How many files(0—15)?**

这是DISK BASIC 系统在询问用户要建立多少个文件缓冲区来进行磁盘的读写处理。例如，输入3，BASIC系统就设置 4 个缓冲区，编号为0、1、2、3。但要注意这时可供用户读写文件使用的缓冲区只有1～3号三个缓冲区。这是因为 0 号缓冲区是供 BASIC 系统在执行那些需要同文件打交道的命令（如BASIC程序的输入LOAD、存储 SAVE、合并MERCE等）时使用的。显然，为了程序的输入输出，不管用户是否使用文件，0 号缓冲区总是需要的。

以上提示表明，建立文件缓冲区的数目是有限的，这也就是说，用户在程序中能够同时打开的文件是有限的。另外，一个缓冲区不能同时为几个文件所共有。例如对「How many files(0—15)?」回答 3，则同时可以打开的文件数亦为 3。

## 5.5 顺序文件的读入

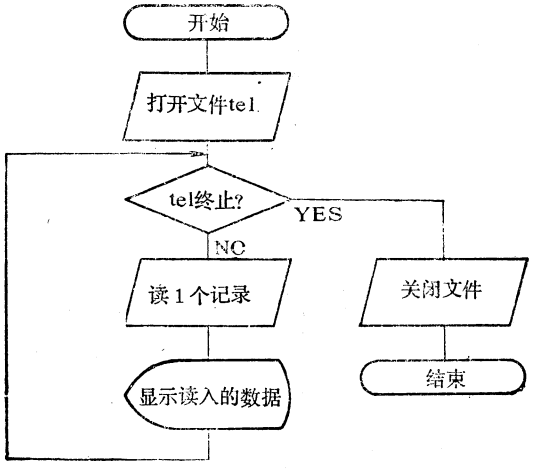
下面介绍从顺序文件中读入数据的步骤，借以学习按 INPUT 方式处理顺序文件的方法。

**（1）从电话簿文件中读取记录**

程序表5.5.1所示的程序，可将文件tel 中的记录全部读入。现在我们来详细分析一下该程序。

程序表 5.5.1 将文件tel的记录全部读入的程序

```
10 OPEN "tel" FOR INPUT AS #1
20 IF EOF(1) THEN 60
30 INPUT #1,NM$,TEL$
40 PRINT NM$,TEL$
50 GOTO 20
60 CLOSE #1
70 END
```




程序表 5.5.1 所列程序的流程图

**打开文件（INPUT方式）**

· **OPEN〈文件指定〉 INPUT AS#〈文件编号〉**

在从文件中读取记录之前，首先要按INPUT方式打开文件。参见10号语句。

**10 OPEN"tel"FOR INPUT AS #1**

上述语句系按INPUT方式打开顺序，因此文件指针是指向文件的开头部位的，从而可按顺序文件从文件的开头读取记录。如果被指定要打开的文件不存在的话，则显示错误信息：

**File not found**

此外，如前所述，通过执行10号语句，便将文件tel的文件编号指定为1，也就是将1号文件缓冲区分配给该文件供其读入记录使用。

**文件结束检查**

**· EOF (〈文件编号〉)**

EOF是用来检查文件是否结束的函数。在顺序输入时，利用这个函数可以避免“输入越界”的错误发生。该函数的带入值是一个待检验文件的文件编号。当已将该文件的全部记录读入后，EOF函数的带出值为真（-1），否则为假（0）。

**20 IF EOF(1)THEN 60**

20号语句表示，若已读到文件末尾，则转向60号语句，否则执行下一条语句。如果没有设置20号语句，那末就会出现已到达文件末尾而仍要继续读记录的情况，这时，将出现错误

信息：

**INPUT past end**

同时程序终止执行。当执行20号语句，且EOF函数的值为真时．即关闭文件（60号语句）并结束程序执行。

**记录的读入**

**・INPUT＃〈文件编号〉，〈输入名表〉**

从文件中读取记录，使用INPUT＃命令。参见30语句。

30 **INPUT ＃1，NM＄，TEL＄**

执行上述语句，可从一号文件（也就是 tel 文件）中读取记录，并将读出记录的姓 名数据项赋给变量NM＄，将电话号码数据项赋给 TEL＄，然后，修正文件指针使其指向下一个记录的开头。

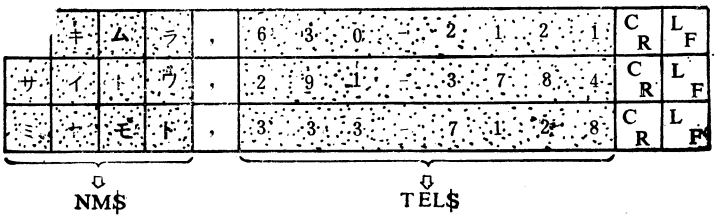



图 5.5.1 数据的读入

接着，40号语句将读入的记录显示到屏幕上，50号语句转去读下一个记录。执行这个程序，则在屏幕上显示出如画面5.5.1所示的结果。

```
run↵
ｷﾑﾗ          630-2121
ｻｲﾄｳ         291-3784   显示出电话簿文件的全部记录
ﾐﾔﾓﾄ         333-7128
Ok
```

画面 5.5.1 表5.5.1所列程序的执行结果

**（2）读入记录的分隔符**

在INPUT＃命令中，若是用逗号来分隔变量的话，那么在读取数据时，下述符号均视为分隔符：

**・读入字符数据时：逗号，$C_R$**

**・读入数值数据时，逗号，空格，$C_R$**

INPUT＃命令忽视在执行DRINT＃命令时，跟随在$C_R$后面的$L_F$。另外，若想把逗号也作为文字数据的一部分读入的话，则应在写入数据时，用引号把包括逗号在内的字符数据括起来。（参照图5.5.2）这时，引号将不会被当做字符数据读入到字符变量中。

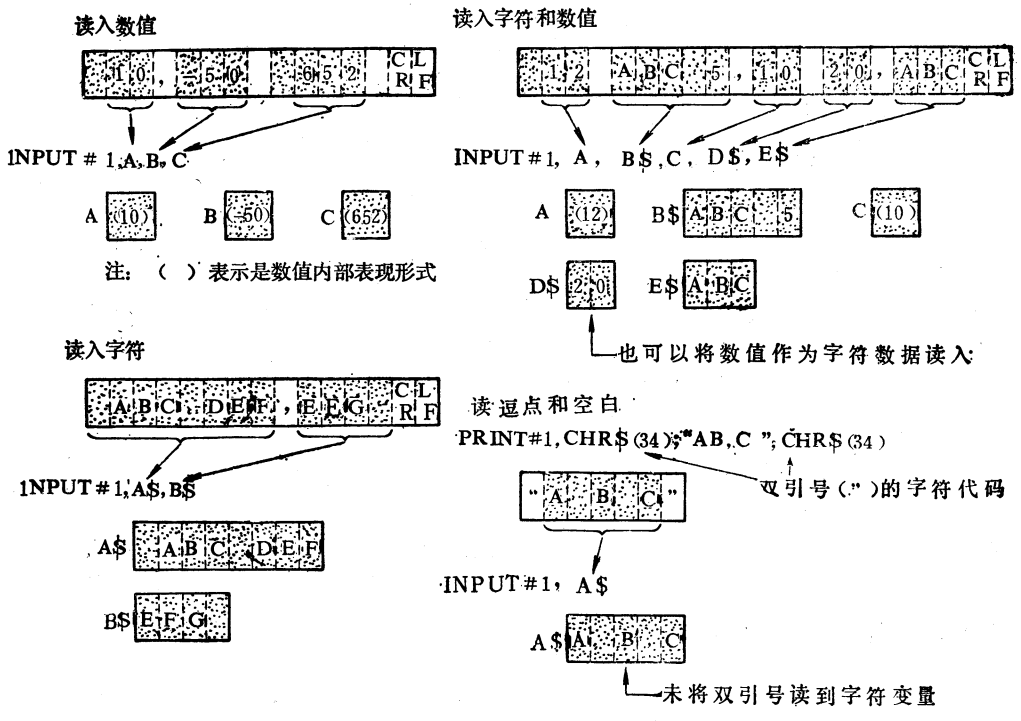



图 5.5.2

### （3）一次读一个记录

**· LINE INPUT #〈文件编号〉,〈字符变量〉**

使用INPUT#语句读入数据时，数据按被逗号及空格分隔的情况读入到变量中去。而使用LINE INPUT#语句读入数据时，则是把直到一个 $C_R$ 为止之前的全部字符不间断地作为一个数据读入变量。注意，此时的变量必须为字符型*。

现将前面介绍过的电话簿文件读入程序的30、40号语句变更如下：

**30 LINE INPUT #1, A$**

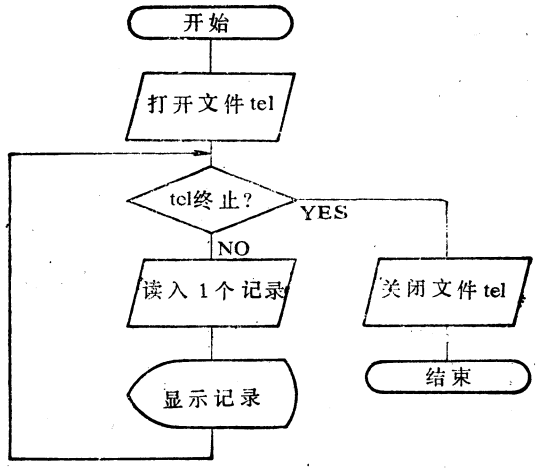
**40 PRINT A$**

使用INPUT语句时，姓名和电话号码是分别被读入不同的变量的。而使用LINE INPUT #语句，则将姓名和电话号码连接起来作为一个字符串数据读入变量。

程序表 5.5.2 用LINE INPUT语句读入数据的程序

```
10 OPEN "tel" FOR INPUT AS #1
20 IF EOF(1) THEN 60
30 LINE INPUT #1,A$      变更
40 PRINT A$
50 GOTO 20
60 CLOSE #1
70 END
```

* LINE INPUT#字符，将数据读入字符变量，一次只能读入255个字符数据。因此对于一个记录在256个字符以上的顺序文件由于应将记录分隔开读入，以不使用LINE INPUT#字符为好。




程序表 5.5.2　所列程序的流程图

执行上述程序，每次可将1个记录全部读入变量A $ 。如图5.5.3所示。

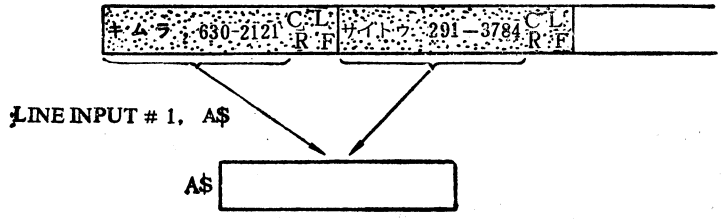



图 5.5.3　利用LINE INPUT#语句读入数据

执行结果如画面5.5.2所示

```
run ↵
ﾔﾏﾓﾄ    630-2121
ｻｲﾄｳ    291-3784
ﾐﾔﾓﾄ    333-7128
Ok
```

画面 5.5.2　表5.5.2所列程序的执行结果

另外，使用LINE INPUT#语句，可以一行一行地读入以ASCII代码形式存储的BASIC程序文件。

**（4）读任意个字符**

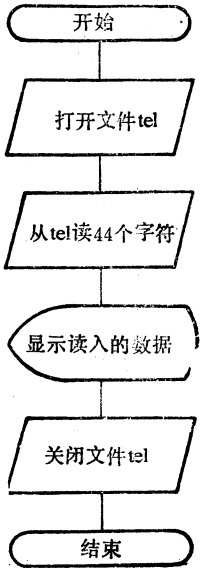
**• INPUT $ (〈字符数〉, 〈文件编号〉)**

INPUT $ 是这样一个函数，它从用〈文件编号〉所指定的文件中，顺序读入用〈字符数〉指定的数个字符。其中不仅逗号及空格等分隔符，就连$C_R$和$L_F$也均被当做数据予以读入。

请参照下表。

程序表 5.5.3　利用INPUT＄函数读入数据的程序

```
10 OPEN "tel" FOR INPUT AS #1
20 A$=INPUT$(44,1)
30 PRINT A$
40 CLOSE #1
```




程序表 5.5.3　所列程序的流程图

这是一个从文件tel中读取44个字符，并加以显示的程序，其执行结果如画面5.5.3所示。

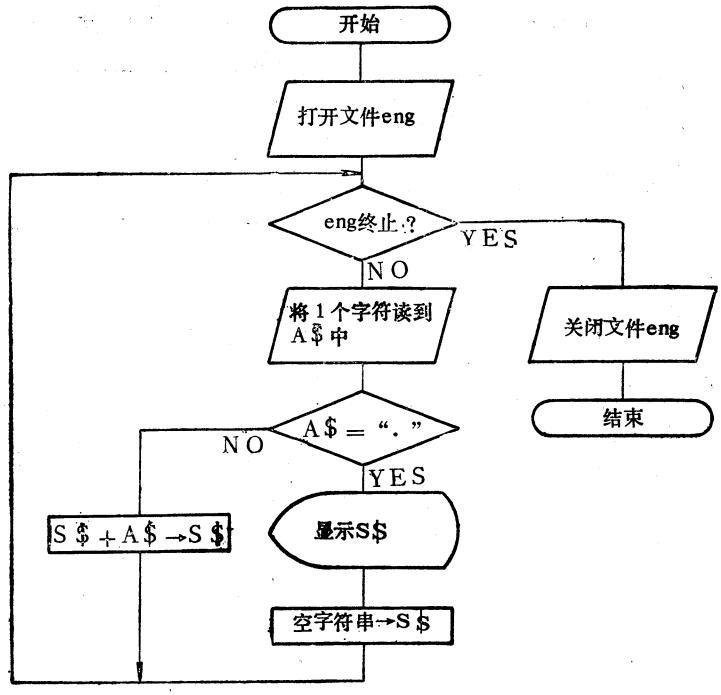
```
run↵
ｷﾑﾗ,630-2121
ｻｲﾄｳ,291-3784
ﾐﾔﾓﾄ,333-7128

Ok
```

画面 5.5.3　表5.5.3所列程序的执行结果

使用INPUT＄函数，可将任意字符作为分隔符使用。例如程序表5.5.4所列的程序就是以句号来作为分隔符的。

程序表 5.5.4　以句号作为分隔符读入数据的程序

```
100 OPEN "eng" FOR INPUT AS #1
110 IF EOF(1) THEN 190
120 A$=INPUT$(1,1)·····读入1个字符
130 IF A$="." THEN 160
140 S$=S$+A$
150 GOTO 110
160 PRINT S$
170 S$=""
180 GOTO 110
190 CLOSE
```




程序表 5.5.4 所列程序的流程图

上述程序中，用120号语句的 INPUT $ 函数，一个字符一个字符地读入，接着用130 号语句判别读入的字符是否为句号。


图 5.5.4

文件eng的构成如图5.5.4所示。因此，程序的140号语句里的 $，则如图 5.5.5 所示的样子被赋值。

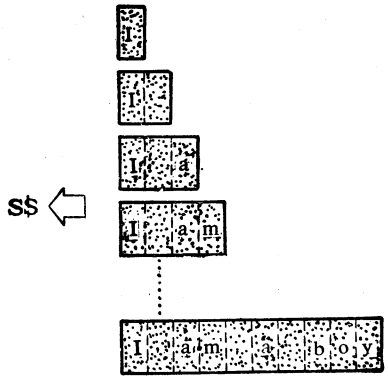



图 5.5.5

画面5.5.4表示程序的执行结果。

```
run↵
I am a boy
You are a girl
Ok
```

画面 5.5.4　表5.5.4所列程序的执行结果

## 5.6　对顺序文件追加记录

现在我们来介绍对顺序文件进行追加记录的步骤，借以学习按 APPEND 方式处理顺序文件的方法。

**（1）对电话簿文件追加记录**

下面是一段在顺序文件tel后面追加两个记录的程序。

**程序表 5.6.1　电话簿文件的追加记录程序**

```
10 OPEN "tel" FOR APPEND AS #1
20 FOR I=1 TO 2
30    READ NM$,TEL$
40    PRINT #1,NM$;",";TEL$
50 NEXT I
60 CLOSE #1
70 DATA ﾔﾏﾀﾞ,826-3310,ｳｴﾀﾞ,999-1137
```

**打开文件（APPEND方式）**

**·OPEN〈文件指定〉FOR APPEND AS#〈文件编号〉**

当要对文件追加记录时，先用APPEND方式打开文件，请参看10号语句。

**10 OPEN"tel"FOR APPEND AS#1**

上述语句即按APPEND方式打开顺序文件tel。注意，利用APPEND方式只能打开已存在的文件，倘若文件不存在的话，则显示出错误信息：

**File not found**

如果文件存在，打开文件后，文件指针将指向文件的末尾。

**追加记录**

**40 PRINT #1, NM$; ","; TEL$**

通过40号语句，即可将两个记录追加到tel文件上去，具体情况如图5.6.1所示。这时，文件指针移到新追加的记录之后，继续指向文件的最后面。

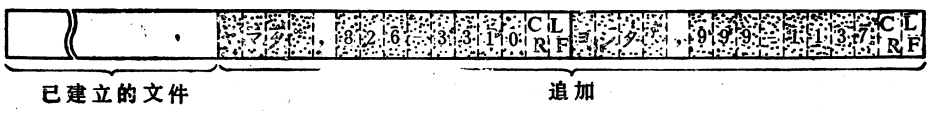



图 5.6.1　对tel文件追加记录

**关闭文件**

60 CLOSE #1

通过CLOSE命令，释放1号缓冲区，解除文件tel和1号文件的对应关系，至此，对文件进行追加记录的工作便宣告结束。

## 5.7 顺序文件的更新

通过前面的介绍，我们看到，建立一个顺序文件并对其进行记录追加是非常简单和方便的。然而，如果要对顺序文件的数据进行置换或删除等更新工作的话，相对来说，就不是那么容易了。之所以这样，是因为没有可供更新顺序文件用的特殊命令。因此，顺序文件的更新往往采取边向另一文件复制边进行更新的方法，下面就来具体说明一下如何进行顺序文件的更新。

**（1）删除电话簿文件中的记录**

为了删除文件中特定的记录，可按图5.7.1所示的样子，将不需要删除的记录，复制到另一个文件上去。

程序表5.7.1所示为删除电话簿文件tel中任意记录的程序。

这个程序是用来从tel文件中，删除具有指定姓名的那个记录的。凡是不应被删除的记录，统统被复制到一个叫做telw的文件中去，而当遇到需要删除的那个记录时，则跳过去不予复制。如此进行下去，直到tel文件中的所有记录均被读完为止。最后删除文件tel，并将文件telw的文件名改为tel。

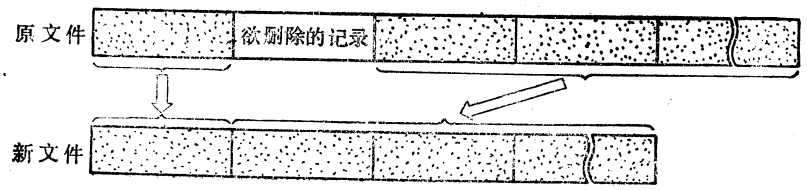
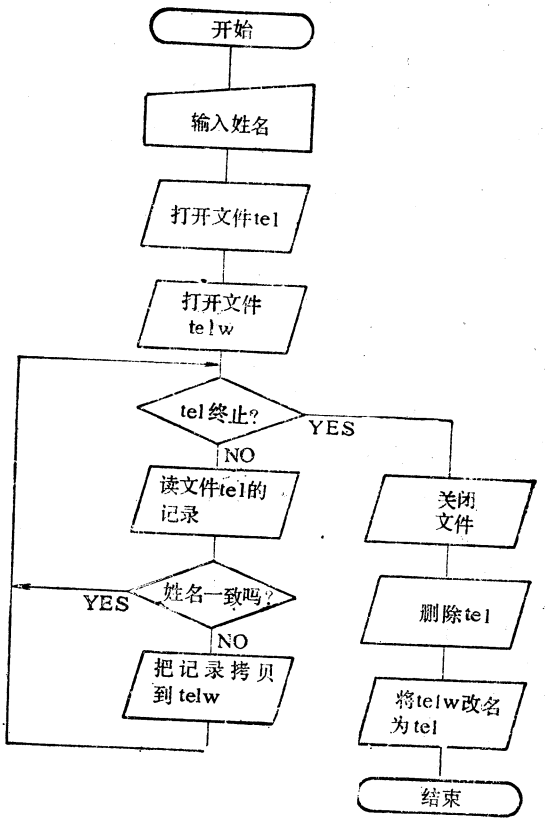



图 5.7.1 记录的删除

**程序表 5.7.1 电话簿文件记录删除程序**

```
100  *** delete record from tel
110 INPUT "delete name";DN$
120 OPEN "tel" FOR INPUT AS #1
130 OPEN "telw" FOR OUTPUT AS #2
140 IF EOF(1) THEN 190
150 INPUT #1,NM$,TEL$
160 IF NM$=DN$ THEN 140
170 PRINT #2,NM$;",";TEL$
180 GOTO 140
190 CLOSE
200 KILL "tel"
210 NAME "telw" AS "tel"
220 END
```




程序表 5.7.1 所列程序的流程图

下面来看一个实际例子，即从文件tel中删掉姓名为“サイトラ”的那个记录。

```
run↵
delete name? サイトウ↵
Ok
```

画面5.7.1表 5.7.1所列程序的执行实例

执行上述程序之后，文件tel的更新结果，如图5.7.2 所示。

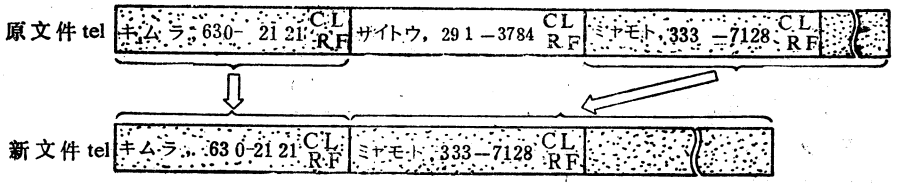



图 5.7.2 删除文件tel的记录

（2）置换电话簿文件中的数据

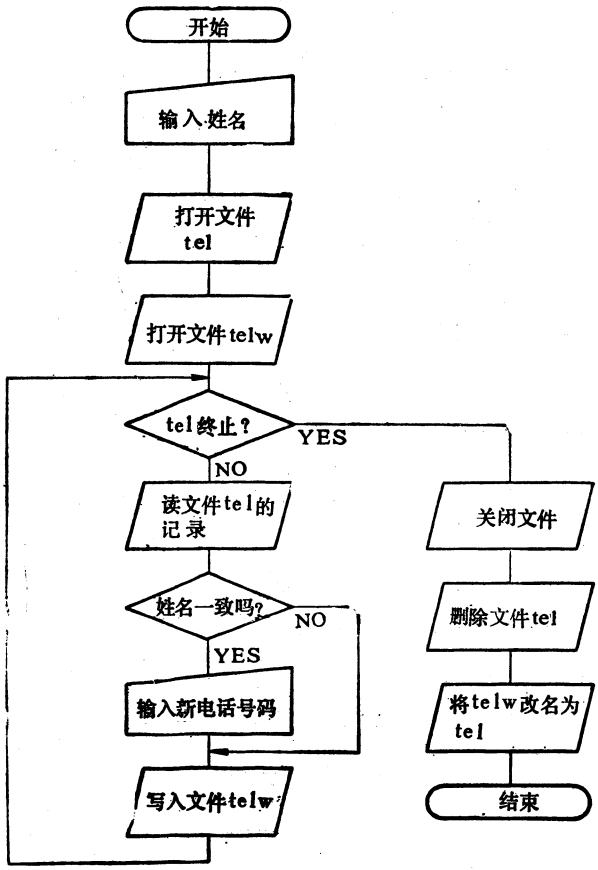
我们通常是这样来置换顺序文件中的数据的：依次读文件中的每一个记录，凡是不需要被置换的记录就简单地复制到另一个文件中，而对那些需要被置换的记录，则通过键盘输入新的数据来代替原记录中的数据，然后把修改后的记录也写入到另一个文件中去。请参看图5.7.3。

下表列出了，用以更换电话簿文件中任意电话号码的程序。

程序表 5.7.2 电话簿文件的数据置换程序

```
100 '*** change record from tel
110 INPUT 'change name';DN$
120 OPEN 'tel' FOR INPUT AS #1
130 OPEN 'telw' FOR OUTPUT AS #2
140 IF EOF(1) THEN 190
150 INPUT #1,NM$,TEL$
160 IF NM$<>DN$ THEN 170
165 INPUT 'new tel no.';TEL$
170 PRINT #2,NM$;',';TEL$
180 GOTO 140
190 CLOSE
200 KILL 'tel'
210 NAME 'telw' AS 'tel'
220 END
```

变更




程序表 5.7.2 所列程序的流程图

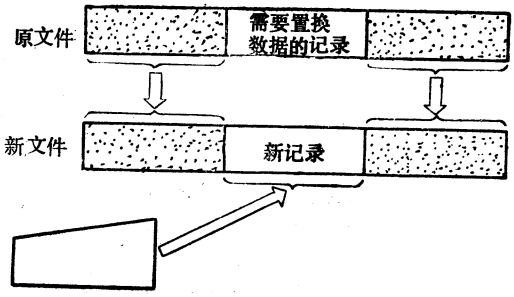



图 5.7.3 文件记录的置换

我们注意到，在上述程序中那些用长方框围起来的语句与前面的删除程序不同，也就是说对置换程序来讲，当找到具有指定姓名的那个记录时，应马上从键盘上输入新的电话号码，用以替换原来的电话号码，并将其写入到文件中去。

下面是用045-111-2345来置换“ミセモト”原来的电话号码的实例。

```
run↵
change name? ミヤモト↵
new tel no.? 045-111-2345↵
Ok
```

画面5.7.2　表5.7.2所列程序的执行实例

图 5.7.4　表示文件tel更新后的样子。

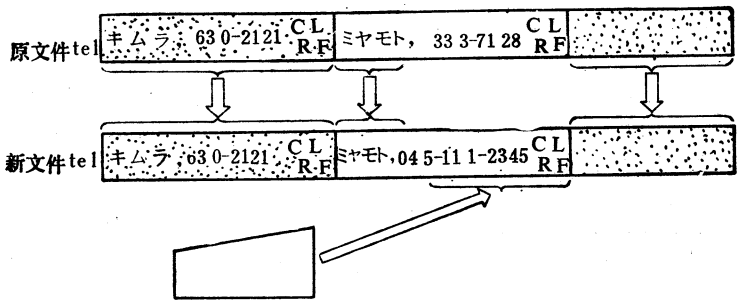



图 5.7.4　tel文件的数据置换

（3）向电话簿文件中插入记录

所谓插入记录是指，在顺序文件的现有记录之间插入新的记录。如图5.7.5所示。

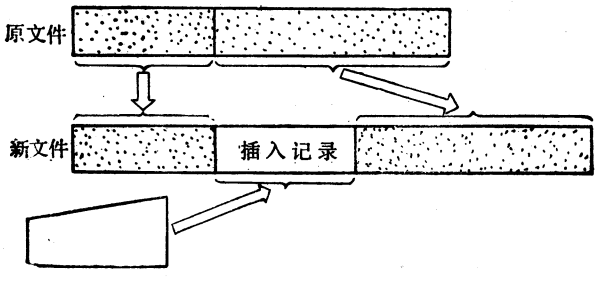



图 5.7.5　记录的插入

表5.7.3所示为向电话簿文件tel中插入新记录的程序。

程序表 5.7.3 电话簿文件的记录插入程序

```
100 '*** insert record from tel
110 INPUT "insert name";ON$
120 OPEN "tel" FOR INPUT AS #1
130 OPEN "telw" FOR OUTPUT AS #2
140 IF EOF(1) THEN 190
150 INPUT #1,NM$,TEL$
160 IF NM$<>ON$ THEN 170
162 INPUT "new name";NNM$
165 INPUT "new tel no.";NTEL$
167 PRINT #2,NNM$;",";NTEL$
170 PRINT #2,NM$;",";TEL$
180 GOTO 140
190 CLOSE
200 KILL "tel"
210 NAME "telw" AS "tel"
220 END
```

变更

在上述程序中，长方框围住的部分为置换程序的变更部分。此程序用于，在具有特定姓名的记录之前插入新的记录。

下面在姓名为ミセモト的那个记录之前插入一个新记录，参见画面5.7.3

```
insert name? ミヤモトノ ·····指定记录插入位置
new name? トダ゛ノ
new tel no.? 608-1439ノ  } 插入记录
Ok
```

画面5.7.3表 5.7.3 所列程序的执行实例

程序执行后，文件tel的变化情况如图5.7.6所示。

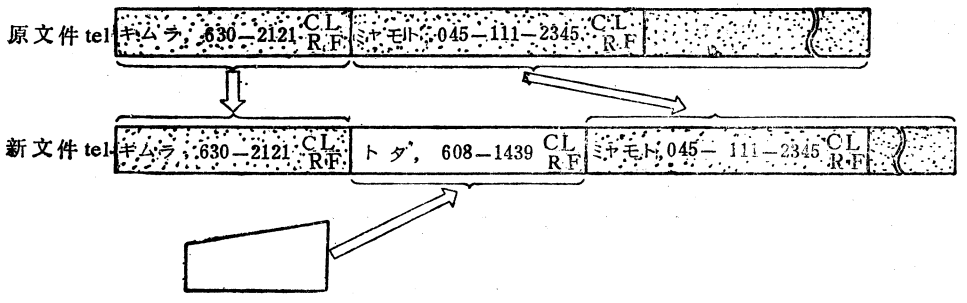



图 5.7.6 向tel文件中插入记录

## 5.8 库存管理程序

本书介绍一个简单的库存管理程序实例，作为学习顺序文件的小结，（程序详见表5.8.1），本程序包括库存文件（stock）的建立、追加、更新、删除和读入等五项功能。

库存文件stock的结构如图5.8.1所示。

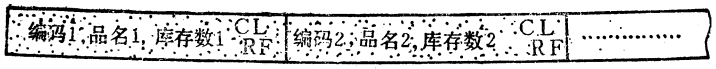



图 5.8.1 库存文件stock

执行库存管理程序，则在屏幕上出现如画面5.8.1所示的信息，用户可以选择其中任何一种功能。

```
****** menu ******
1 建立 ..... 建立库存文件
2 追加 ..... 向库存文件追加数据
3 更新 ..... 更新库存数目
4 删除 ..... 删除库存记录
5 读入 ..... 读记录
0 结束 ..... 程序结束
?
```

画面 5.8.1 库存管理程序的功能提示菜单

下面分别对该程序的各个处理部分加以说明。

**库存文件的建立**

文件stock是按OUTPUT方式打开，作为一个新文件建立的。输入的数据为编码、品名和库存数。这些数据按输入的顺序依次被写入顺序文件stock。当一旦输入了空编码（仅按回车键）之后，这部分处理即告结束。

现在我们来建立起stock文件，并输入3个库存记录，具体情况如画面5.8.2所示。

```
***** 建立 *****
编码 ? 10↵
品名 ? pencil↵
库存数 ? 40↵

编码 ? 20↵
品名 ? book↵
库存数 ? 22↵

编码 ? 30↵
品名 ? clip↵
库存数 ? 100↵

编码 ?↵ ..... 若输入编码为空编码时，处理结束
complete
hit any key
```

画面 5.8.2 库存管理程序的建立方式

**库存记录的追加**

这部分处理是在已建立的库存文件的后面追加新的库存记录。输入的方法和建立库存文件时的情况相同。参见画面5.8.3。

**库存数的更新**

在这部分，需要输入编码和入、出库数。编码必须按顺序从处于文件最前面的那个编码开始输入。一旦输入了空字符（仅按回车键）或未定义的编码以后，便结束更新处理。另外要注意，入库数须输入正值(+)，而出库数则输入负值(-)。

```
***** 追加 *****
编码 ? 40↵
品名 ? pen↵
库存数 ? 67↵

编码 ? 50↵
品名 ? file↵
库存数 ? 10↵

编码 ?↵
complete
hit any key
```

画面 5.8.3 库存管理程序的追加方式

**库存记录的删除**

删除记录很容易，只需输入该记录的编码，即可删除该记录。

```
****** 更新 ******
编码 ? 10↵
编码 =10      品名=pencil      库存数= 40
入库(+)出库 (-)数? 10↵
编码 =10      品名=pencil      库存数= 50

编码 ? 20↵
编码 =20      品名=book        库存数= 22
入库(+)出库 (-)数? -4↵
编码 =20      品名=book        库存数= 18

编码 ?↵
complete
hit any key
```

画面 5.8.4 库存管理程序的更新方式

**库存记录的读取**

输入编码，则读入相应的记录，并将该记录的内容显示在屏幕上。若输入“all”作为编码的话，则读入全部记录。

```
****** 删除 ******
编码 ? 20↵
deleted
编码 ? 40↵
deleted
编码 ?↵
complete
hit any key
```

画面 5.8.5 库存管理程序的删除方式

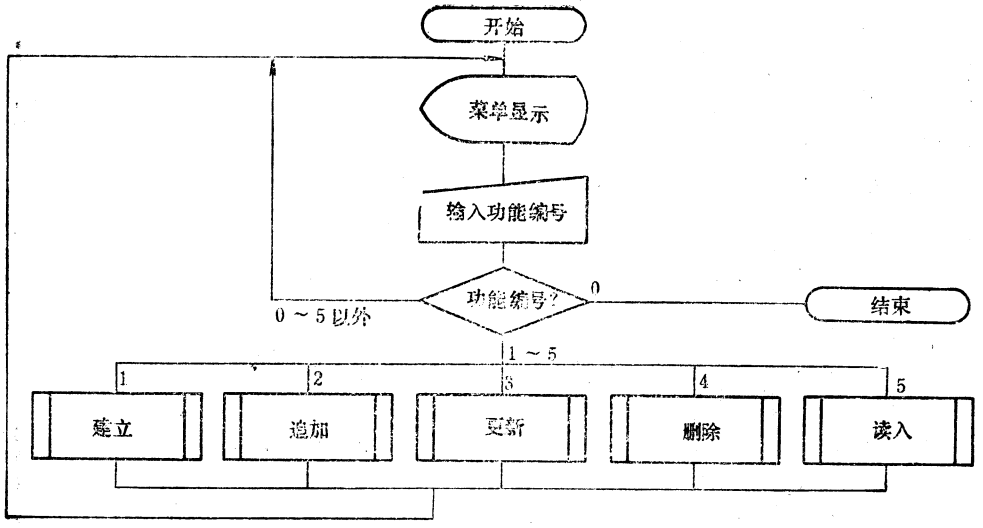
```
******* 读入 *******
code   no.? 30↵
编码              品名          库存数
30                clip           100
code   no.? all↵
编码              品名          库存数
10                pencil          50
30                clip           100
50                file            10
code   no.? ↵
complete
hit any key
```

画面 5.8.6 库存管理程序的读入方式

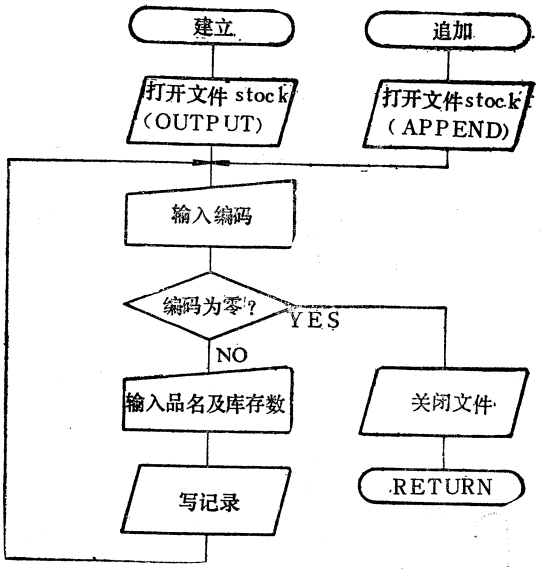
**程序表 5.8.1 顺序文件的应用实例库存管理程序**

```
1000 '****************************
1010 '*          库存管理          *
1020 '****************************
1030  CLS
1040  PRINT "****** menu ******
1050  PRINT "1 建立
1060  PRINT "2 追加
1070  PRINT "3 更新
1080  PRINT "4 删除
1090  PRINT "5 读入
1100  PRINT "0 结束
1110  INPUT F
1120  IF F=0 THEN END
1130  IF F<0 OR F>5 THEN 1040
1140  CLS
1150  ON  F GOSUB 1220,1180,1310,1670,1480
1160  GOTO 1030
1170  '
1180  PRINT "**** 追加 *****"
1190  OPEN "stock" FOR APPEND AS #1
1200  GOTO 1250
1210  '
1220  PRINT "*****   建立    *****"
1230  OPEN "stock" FOR OUTPUT AS #1
1240  PRINT
1250  CD$="":INPUT "编码";CD$
1260  IF CD$="" THEN 1780
1270  INPUT "品名 ";NM$
1280  INPUT "库存数 ";Q
1290  PRINT #1,CD$;",";NM$;",";Q
1300  GOTO 1240
1310  '
1320  PRINT "****** 更新 ******"
1330  OPEN "stock" FOR INPUT AS #1
1340  OPEN "stockw" FOR OUTPUT AS #2
1350  C$="":INPUT "编码";C$
1360  IF C$="" THEN C$="99999"
1370  IF EOF(1) THEN 1780
1380  INPUT #1,CD$,NX$,QX
1390  IF C$ <> CD$ THEN 1450
1400  PRINT "编码=";C$;"      品名=";NX$;"      库存数=";QX
1410  INPUT "入库(+)出库 (-)数 ";QN
1420  QX=QX+QN
1430  PRINT "编码=";C$;"      品名=";NX$;"      库存数 =";QX
1440  PRINT
1450  PRINT #2,CD$;",";NX$;",";QX
1460  IF C$=CD$ THEN 1350
1470  GOTO 1370
1480  '
1490  PRINT "****** 读入 *******"
1500  C$="":INPUT "code   no.";C$
```

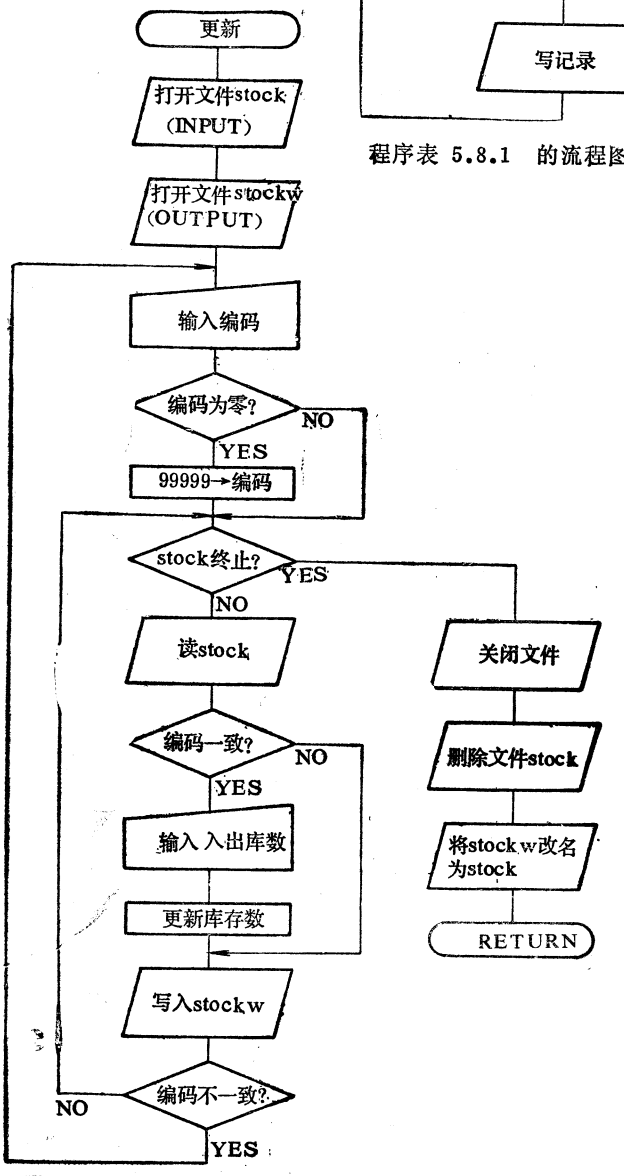
```
1510  IF C$="" THEN 1780
1520  OPEN "stock" FOR INPUT AS #1
1530  IF C$="all" THEN GOSUB 1610:GOTO 1590
1540  IF EOF(1) THEN 1590
1550  INPUT #1,CD$,NM$,Q
1560  IF CD$ <> C$ THEN 1540
1570  PRINT "编码","品名","库存数
1580  PRINT CD$,NM$,Q
1590  CLOSE
1600  GOTO 1500
1610  '****** 读入  ******
1620  PRINT "编码  ","品名","库存数
1630  IF EOF(1) THEN RETURN
1640  INPUT #1,CD$,NM$,QX
1650  PRINT CD$,NM$,QX
1660  GOTO 1630
1670
1680  PRINT "****** 删除   ******"
1690  OPEN "stock" FOR INPUT AS #1
1700  OPEN "stockw" FOR OUTPUT AS #2
1710  C$="":INPUT "编码";C$
1720  IF C$ ="" THEN C$="99999"
1730  IF EOF(1) THEN 1780
1740  INPUT #1,CD$,NM$,QX
1750  IF C$=CD$ THEN PRINT "deleted":GOTO 1710
1760  PRINT #2,CD$;",";NM$;",";QX
1770  GOTO 1730
1780  PRINT "complete"
1790  PRINT "hit any key"
1800  IF INKEY$="" THEN 1800
1810  CLOSE
1820  IF F<3 OR F>4 THEN RETURN
1830  KILL "stock"
1840  NAME "stockw" AS "stock"
1850  RETURN
```



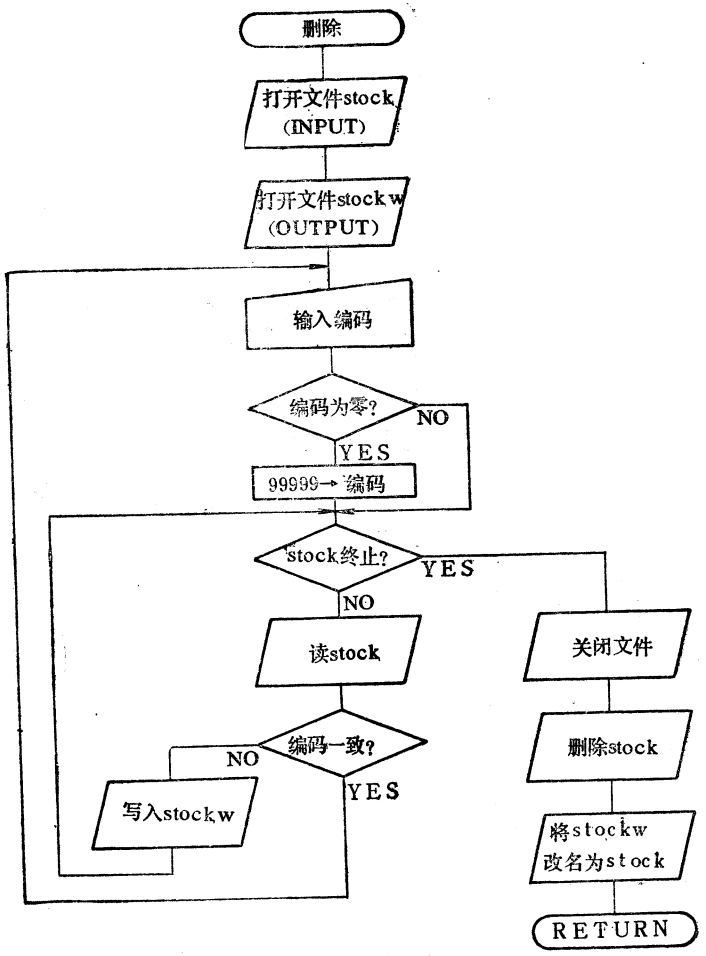

程序表 5.8.1 所列程序的流程图（其一）



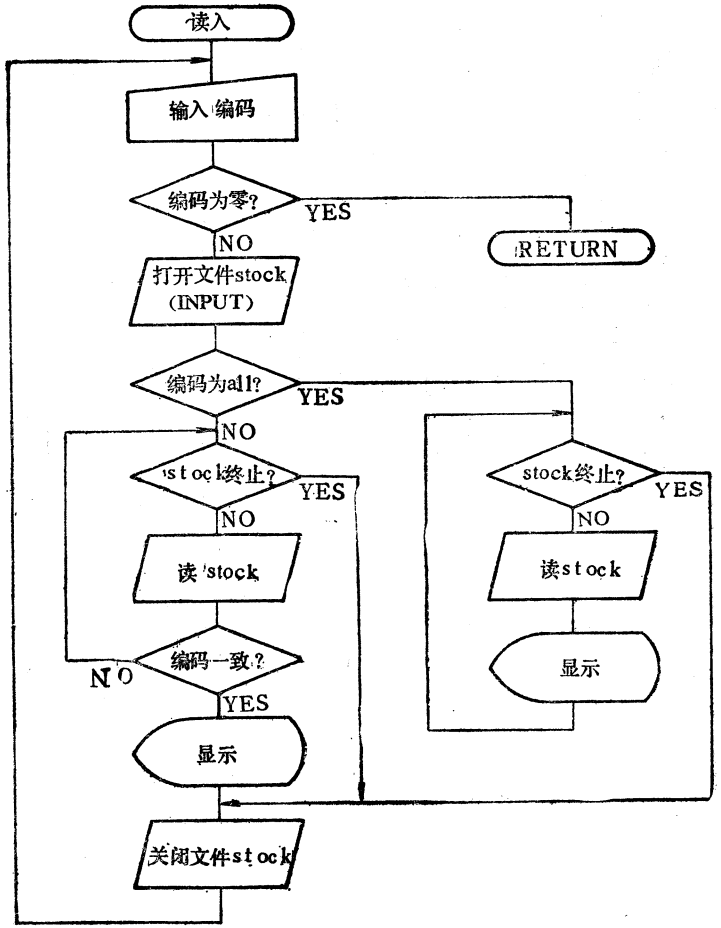

程序表 5.8.1 的流程图（其二）




程序表 5.8.1 的流程图（其三）




程序表 5.8.1 的流程图（其四）




程序表 5.8.1 的流程图（其五）

## 5.9 顺序文件的汇总

最后，我们将顺序文件的特点和各种命令加以汇总如下：

顺序文件的特点：

- 记录用回车符加以分隔
- 各个记录其长度可能不一样（可变长记录形式）
- 数据一般都是从文件的开头按顺序进行读写
- 置换或删除文件中的数据比较困难
- 由于记录之间没有空隙，故比较节省空间
- 与随机文件相比，读写操作比较简便顺序文件所用命令小结

| | | |
|---|---|---|
| 用OUTPUT方式打开顺序文件 | PC | OPEN〈文件指定〉FOR OUTPUT SA#〈文件编号〉 |
| | FM | OPEN"O",#〈文件编号〉,〈文件指定〉 |
| 用INPUT打开顺序文件 | PC | OPEN〈文件指定〉FOR INPUT AS#〈文件编号〉 |
| | FM | OPEN"I",#〈文件编号〉,〈文件指定〉 |
| 用APPEND方式打开顺序文件 | PC | OPEN〈文件指定〉FOR APPEND AS#〈文件编号〉 |
| | FM | OPEN"A",#〈文件编号〉,〈文件指定〉 |
| 关闭文件 | PC | CLOSE[#〈文件编号〉,…] |
| | FM | CLOSE[#〈文件编号〉,…] |
| 向顺序文件输出数据 | PC | PRINT#〈文件编号〉,〈输出名表〉 |
| | FM | PRINT#〈文件编号〉,〈输出名表〉 |
| 用指定的写方式向顺序文件输出数据 | PC | PRINT#〈文件编号〉,USING〈写方式控制字符串〉,〈输出名表〉 |
| | FM | PRINT#〈文件编号〉,USING〈写方式控制字符串〉,〈输出名表〉 |
| 向顺序文件写入数据 | PC | WRITE#〈文件编号〉,〈输出名表〉 |
| | FM* | WRITE#〈文件编号〉,〈输出名表〉 |
| 从顺序文件读入数据 | PC | INPUT#〈文件编号〉,〈输入名表〉 |
| | FM | INPUT#〈文件编号〉,〈输入名表〉 |
| 将整个1行不加分隔地由顺序文件读入字符变量 | PC | LINE INPUT#〈文件编号〉,〈字符变量〉 |
| | FM | LINE INPUT#〈文件编号〉,〈字符变量〉 |
| 从文件读任意字符数的字符 | PC | INPUT $ (〈字符数〉,#〈文件编号〉) |
| | FM | INPUT $ (〈字符数〉,#〈文件编号〉) |
| 检查文件的结束 | PC | EOF(〈文件编号〉) |
| | FM | EOF(〈文件编号〉) |

* F-BASIC仅用Ver4.0支持。

# 6. 处理数据的文件〔2〕

## ——随机文件——

## 6.1 随 机 文 件

顺序文件是按物理顺序从头到尾读取文件记录的。而随机文件，顾名思义是可以随意读取文件记录。例如，可在读过第5个记录之后，写入第3个记录，然后再读取第6个记录。

利用磁盘刚好可以实现随机文件。一旦使用熟了非常灵活、方便。但从另一方面讲，一些规定要比顺序文件复杂一些。

**（1）记录形式**

我们把文件内，长度相等的记录形式称为定长记录形式。顺序文件内的记录长度并不一定全都相等，但随机文件的记录长度却是一定的。即随机文件是属于定长记录形式的文件。PC-8801等许多微型机的磁盘BASIC的随机文件记录的长度均为256字节*。

记录内的各个数据项称做字段（field），对于随机文件来说，字段的长度都是预先设定好的。由于随机文件的记录长度业已设定为256字节，因此几乎所有文件记录当中均有一部分区域没有被利用。

文件内的各记录均按物理顺序从一开始编号，该号码便称为记录号。在处理随机文件，读取所需要的数据时，可根据该记录号进行指定。

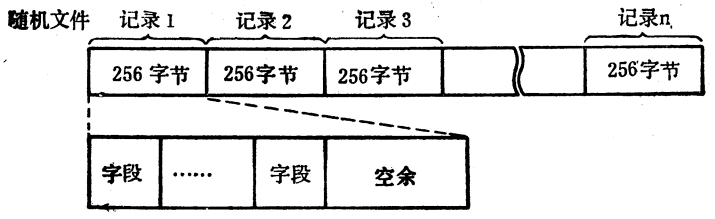



图 6.1.1 随机文件的构成

随机文件的一个记录一般为256字节，那么这个数据是根据什么确定的呢？这是因为软磁盘都是以扇区为单位来分配空间的，每一个扇区为256字节**。双面双密度的5英寸软磁盘大约划分成1200个扇区，双面双密度标准软磁盘约划分成4000个扇区。随机文件的记录长度为256字节，恰恰和一个扇区的字节相当。有些BASIC可任意设定随机文件的记录长度，以减少记录内的空白区。但这样一来，一个记录有可能会跨越两个扇区，于是指定记录号时，必须计算该记录处于第几号扇区，显然系统处理也变得复杂多了。关于磁盘的内部构造，请参照第9章。

**（2）读写步骤**

随机文件的读写步骤和顺序文件的读写步骤基本上是相同的。即首先打开文件，然后读

---

* IBM-PC等计算机的BASIC可对随机文件的记录长度进行指定。

** IBM-PC等计算机是512字节为一个扇区。

写数据，最后关闭文件。但在对随机文件进行读写时，必须充分意识到缓冲区的存在。

第五章业已介绍过，读写文件时，肯定要经由缓冲区。即使是读写顺序文件也需要经由缓冲区，只不过一些复杂的操作都由系统承担罢了，用户无须多加考虑。然而，处理随机文件，则必须由用户自己考虑内存和缓冲区之间以及缓冲区和文件之间的数据传送。

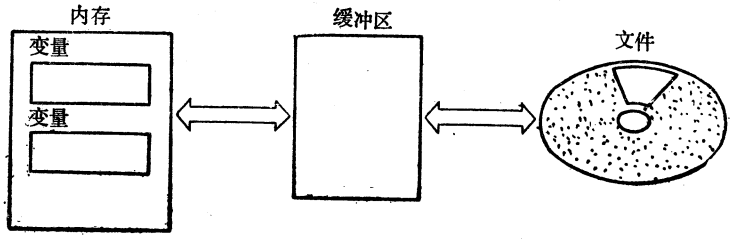



图 6.1.2 处理随机文件，应意识到缓冲区的存在

更具体点说，读写数据时，需要遵从以下步骤。

**向磁盘文件写入数据：**

① 打开文件。
② 定义缓冲区的字段。
③ 将数据置入缓冲区字段。
④ 将缓冲区的内容写入文件。
⑤ 关闭文件。

**从磁盘文件内读取数据：**

① 打开文件。
② 定义缓冲区的字段。
③ 从文件中将数据读入缓冲区。
④ 将数据从缓冲区的字段中传送给变量。
⑤ 关闭文件。

## 6.2 随机文件的建立

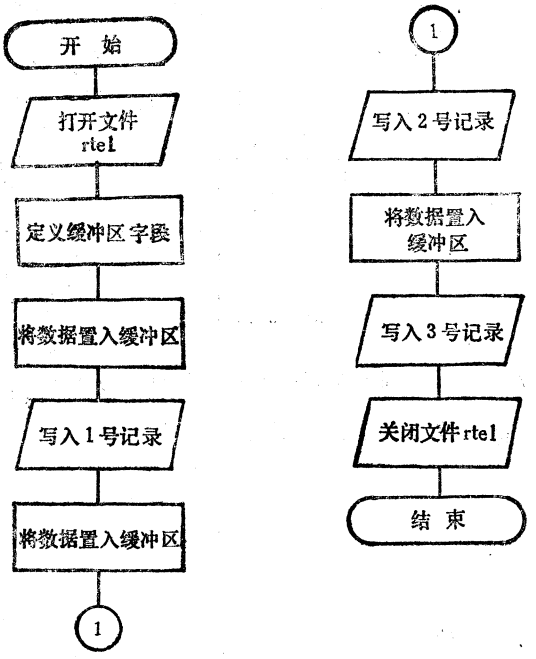
我们准备用一个简单的程序来说明建立随机文件的步骤。和介绍顺序文件时的情况一样，仍然是利用建立电话簿文件用的程序。这样即可以和顺序文件进行对比，又可以了解到随机文件的一些特点。

**（1）建立电话簿文件**

下述程序为建立随机电话簿文件的程序。

**程序表 6.2.1 建立随机电话簿文件的程序**

```
100 OPEN "rtel" AS #1
110 FIELD #1,5 AS NM$,10 AS TEL$
120 LSET NM$="ﾔﾏﾓﾄ"
130 RSET TEL$="630-2121"
140 PUT #1,1
150 LSET NM$="ｻｲﾄｳ"
160 RSET TEL$="291-3784"
170 PUT #1,2
180 LSET NM$="ﾐﾔﾓﾄ"
190 RSET TEL$="333-7128"
200 PUT #1,3
210 CLOSE #1
```




程序表 6.2.1 的流程图

**打开随机文件**

**·OPEN**〈文件指定〉**A $ #**〈文件缓冲区号〉

要想打开随机文件，可以使用不带输入输出方式指定的OPEN命令，参见100号语句。

100 **OPEN "rtel" A $ #1**

上述语句表示打开随机文件rtel，并将rtel的文件编号设定为 1 。

一旦打开顺序文件，直到关闭该文件之前，只可按打开时指定的方式进行工作。例如，若要用INPUT方式打开文件，则只能写入数据。然而，用OPEN命令打开随机文件之后即可读取，又可写入，还可以追加。也就是说，用一个OPEN命令可同时指定INPUT、OUTPUT、APPEND三种方式。

另外，对磁盘BASIC系统来说，打开文件时，若该文件存在，则将文件指针设置到文件的起始位置。若该文件不存在，则建立新的文件，并将文件指针也设置在该文件的起始位置。

**缓冲区字段的定义**

**·FIELD #**〈文件编号〉，〈字段长度〉

**AS**〈字符变量〉〈字段长度〉**AS**〈字符变量〉……

前面已讲过，读写随机文件时，必须以字段为单位，将缓冲区加以划分。随机文件是用FIELD语句对字段进行定义的。

110**FIELD #1, 5AS NM $, 10TEL $**

这表示，从缓冲区的头上开始，将 5 个字节定义为NM $，再将后10个字节定义为TEL $ 。

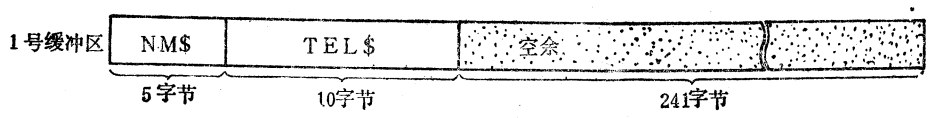



图 6.2.1

这里是用字符变量名来表示字段名的。另外，由于缓冲区长度业已设定为 256 字节，故字段的累计字节数不得超过256。随机文件是用FIELD语句来分割数据的，这一点和顺序文件不一样，不需要专门使用数据分割符号。

**向缓冲区字段设置数据**

- **LSET**〈字符变量〉＝〈字符串〉
- **RSET**〈字符变量〉＝〈字符串〉

定义缓冲区的字段之后，尚应对字段设置数据。可以使用LSET和RSET命令。LSET 是从字段的左边起，RSET是从右边起将数据置入缓冲区字段的。

```
120 LSET NM$ = "ヤマモト"
130 RSET TEL$ = "630-2121"
```

120行是从字段的左边开始，将“ヤマモト”置入NM$，130行则是从字段右边开 始 将“630-2121”置入TEL$。

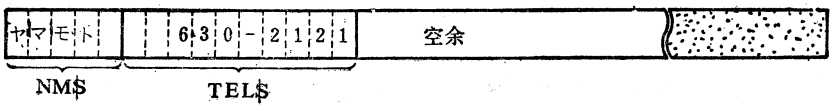



图 6.2.2

在使用LSET或RSET命令时，若数据位数比字段位数少，则用空白填补字段中多余的位数，反过来说，若数据位数大于字段位数，则将不能进入字段内的数据加以舍弃。

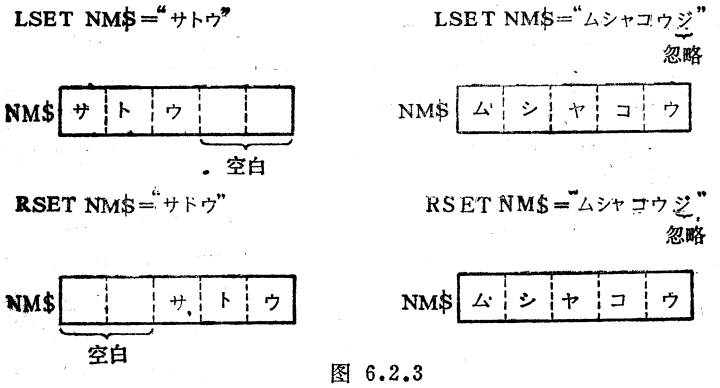



图 6.2.3

* 对RSET命令来说，当字符串长度大于字段宽度时，也是忽略字符串右侧部分

提醒大家注意的是，用一般的赋值语句、READ语句、INPUT 语句等，不可能将数据

置入缓冲区的字段。即

```
NM$ = "ヤマモト"
TEL$ = "630-2121"
READ A$
INPUT C$
```

等均不可行（NM$，TEL$，A$，C$分别为字段名）。也就是说将数据置入字段变量时，必须使用LSET命令或RSET命令。*

**将缓冲区内容写入磁盘文件**

**·PUT#〈文件编号〉，〈记录编号〉**

将数据送进缓冲区之后，再利用PUT命令将缓冲区内容写入文件。PUT命令可将指定编号的缓冲区内容写入指定编号的记录中去。

```
140  PUT#1,1
```

上述语句是将1号缓冲区的内容写到1号记录中去。

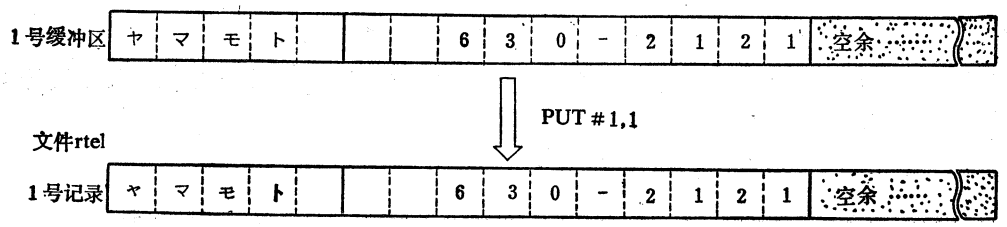



图 6.2.4 利用PUT命令将缓冲区内容写入文件

使用PUT命令，亦可省略记录号，此时，隐含说明的记录号为刚刚被PUT或GET命令所访问过的下一个记录号。

下面接着在150～170行将数据写入记录2，在180～200行将数据写入记录3。

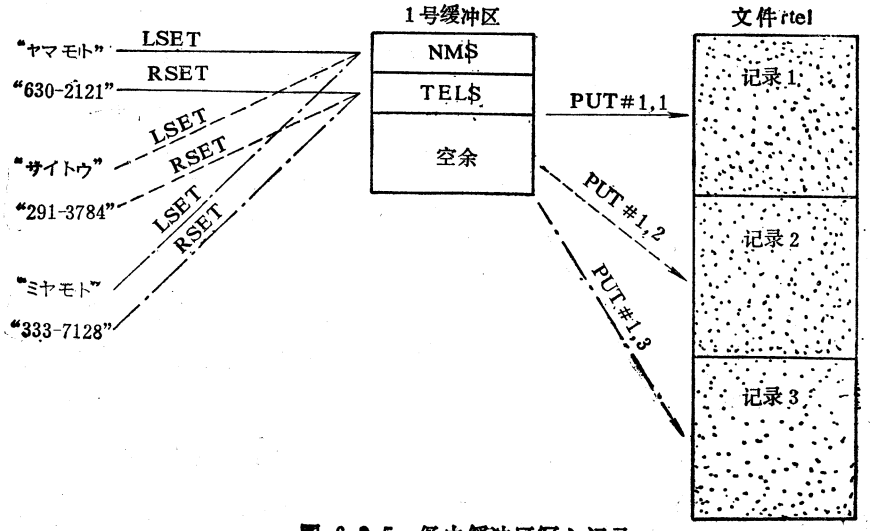



图 6.2.5 经由缓冲区写入记录

**随机文件的关闭**

关闭随机文件，也使用CLOSE命令，这和顺序文件完全一样。

```
210 CLOSE#1
```

**程序的执行**

执行此程序，则建立了随机文件rtel。

```
run↵
Ok
files↵
bfile      1    mfile      1    tel        1    rtel       1
Ok
                                                rtel已建立
```

画面 6.2.1 确认文件建立与否

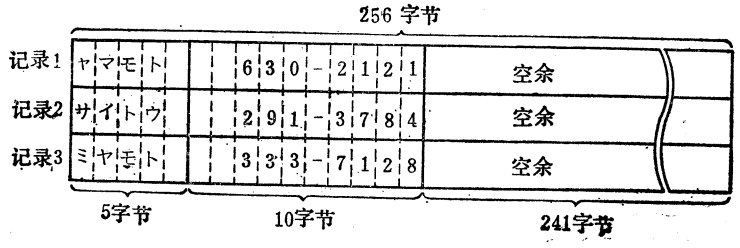



图 6.2.6 随机文件rtel

**另一个程序**

表6.2.1的程序是从文件开头，依次将数据写入记录的。但随机文件并不一定要求按顺序从头开始写入。按表6.2.2的顺序，即先记录 2 的记录 1，再记录 3 的顺序，写入也是可以的。

**程序表 6.2.2 建立随机电话簿文件的程序**

```
100 OPEN "rtel" AS #1
110 FIELD #1,5 AS NM$,10 AS TEL$
120 LSET NM$="ｻｲﾄｳ"
130 RSET TEL$="291-3784"
140 PUT #1,2
150 LSET NM$="ﾔﾏﾓﾄ"
160 RSET TEL$="630-2121"
170 PUT #1,1
180 LSET NM$="ﾐﾔﾓﾄ"
190 RSET TEL$="333-7128"
200 PUT #1,3
210 CLOSE #1
```

## 6.3 随机文件的读入

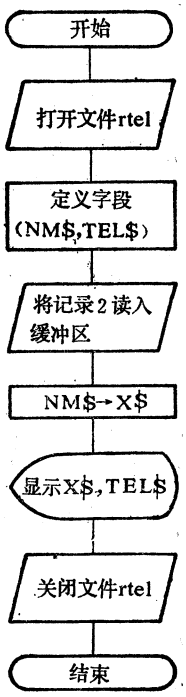
对随机文件来说，通过指定记录号，可立即读入数据。

**（1）读电话簿文件的任意记录**

程序表6.3.1是读取rtel文件 2 号记录的程序。

**程序表 6.3.1 电话簿文件读入程序**

```
10 OPEN "rtel" AS #1
20 FIELD #1,5 AS NM$,10 AS TEL$
30 GET #1,2
40 X$=NM$
50 PRINT X$,TEL$
60 CLOSE #1
```




程序表 6.3.1 的流程图

**文件的打开**

10 **OPEN "rtel" AS#1**

对于随机文件来说，不论是读盘还是写盘，OPEN命令的格式均不改变。

**缓冲区字段的定义**

**20FIELD#1,5AS NM$,10AS TEL$**

照建立该文件时所采用的格式定义缓冲区字段。

**读取记录**

• **GET#**〈文件编号〉,〈记录编号〉

GET命令是PUT命令的反命令，即将指定编号的记录内容从磁盘文件读到指定的缓冲区。

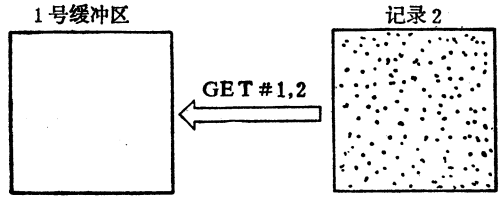



图 6.3.1 将 2 号记录读到 1 号缓冲器

**30 GET#1,2**

30行语句系将记录 2 读到 1 号缓冲区（参照图6.3.1）。

GET命令亦可省略记录号，其效果和PUT命令时的情况完全一样。

**从缓冲区字段取出数据**

取出缓冲区字段的内容，不需要使用特殊命令，一般的赋值语句就可以了*。

* 数值数据则需要使用特殊命令，详细情况参看节6.7（数值数据的读写）。

**40X $ = NM $**

上述行是将缓冲区字段NM $ 的内容递至字符变量X $ 。

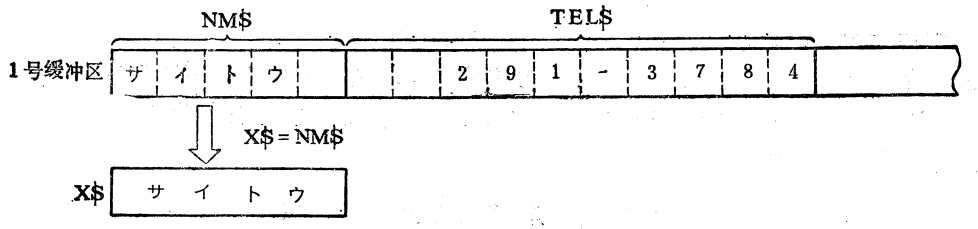



图 6.3.2 将缓冲区字段内容送入变量

50行为将读入的内容，显示在屏幕上。

**50 PRINT X $ , TEL $**

可以看出，直接用PRINT语句也可指定字段名TEL $ ，但文件关闭后，TEL $ 内容即被清除*。如在文件关闭后，仍想利用字段内容，则必须利用赋值语句将其内容赋值给变量。

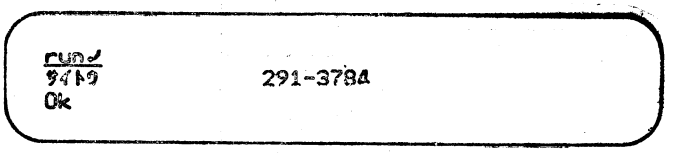
**关闭文件**

**60 CLOSE#1**

最后关闭文件。

**程序的执行**

执行此程序，则读入记录 2 ，并显示出画面6.3.1




画面 6.3.1 表6.3.1的执行结果

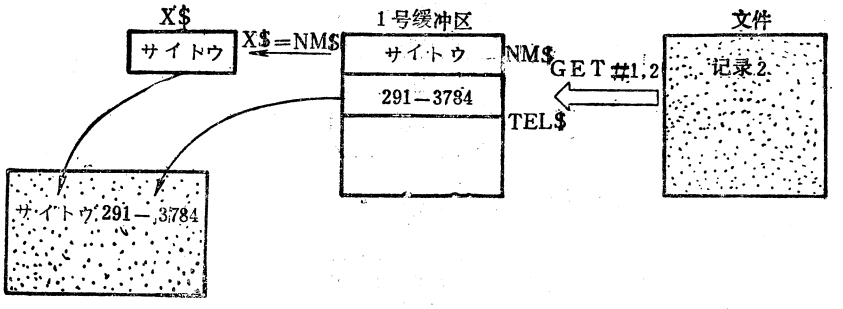



图 6.3.3 读入记录 2

**（2）读入电话簿文件的所有记录**

随机文件的特点在于通过指定记录号，可立即读入任意记录，当然也可从文件的起始位

* F—BASIC不清除，继续保留。

置读取记录。

从文件的起始位置按顺序一个一个读取随机文件的记录，当记录读完时，会出现错误信息。

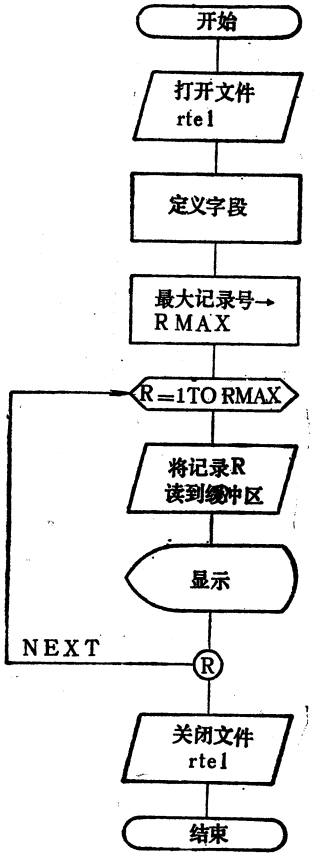
**INPUT past end**

这种情况和顺序文件相同，为避免出现这种错误，顺序文件使用EOF函数进行鉴别，然而由于随机文件不能使用EOF函数*，因而用LOF函数来检验文件的终止。

首先，请看程序表6.3.2。

**程序表 6.3.2 读入的电话簿文件全部记录**

```
10 OPEN "rtel" AS #1
20 FIELD #1,5 AS NM$,10 AS TEL$
30 RMAX=LOF(1)
40 FOR R=1 TO RMAX
50   GET #1,R
60   PRINT NM$,TEL$
70 NEXT R
80 CLOSE #1
```




程序表 6.3.2 流程图

* N—BASIC虽然对于随机文件也使用EOF函数，但为了检验随机文件的终止。使用LOF函数是可以的。

30行使用了LOF函数。

**·LOF(〈文件编号〉)**

LOF函数带回指定文件的最大记录号。

30 **RMAX=LOF(1)**

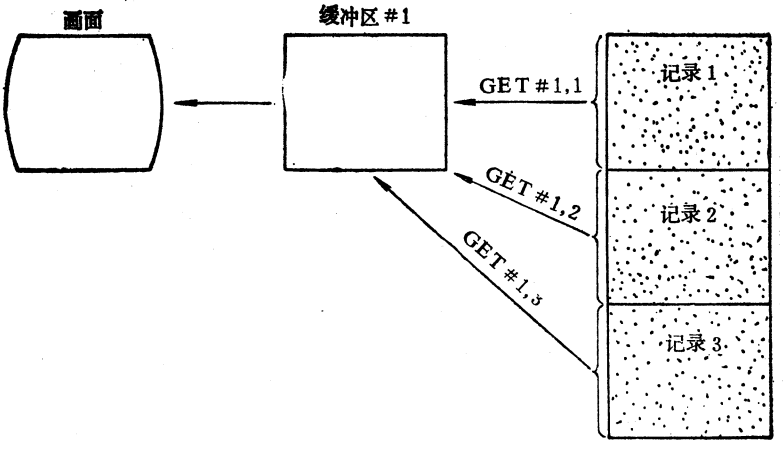
上述行是将1号文件即rtel的最大记录号（此处是3）代入RMAX，由于是从1开始，按顺序分配记录号的，所以最大记录号就等于记录数。(但不一定是已被写入数据的记录数后述)

40～70行，是按记录数读取全部记录。

执行结果如下：

```
run↵
ｷﾑﾗ,630-2121
ｻｲﾄｳ,291-3784
ﾐﾔﾓﾄ,333-7128
Ok
```

画面 6.3.2 表6.3.2的执行结果




6.3.4 读入全部记录

**（3）读空记录**

如前所述，向随机文件写入记录时，不一定要按顺序从头开始。在写入1号记录之后，可以写记录5。那么，此时若要读取2～4号记录，即尚未写入数据的记录时，结果会如何呢

参见程序表6.3.3。

程序表 6.3.3 读空记录的子程序

```
100  OPEN "rtest" AS #1
110  FIELD #1,5 AS A$
120  RSET A$="abcde"
130  PUT #1,1
140  RSET A$="fghij"
150  PUT #1,5
160
170  FOR R=1 TO 5
180    GET #1,R
190    PRINT R,A$
200  NEXT R
210  CLOSE
```

此程序为，先将数据写入记录 1 和记录 5，然后读取1～5号记录。执行结果如下：

```
run
 1            abcde
 2
 3
 4
 5            fghij
Ok
```

画面 6.3.3 表6.3.3执行结果

可以看出，记录 2， 3， 4 被读了进来，但读的是什么，单从这个结果来看，尽管不太明白，但至少未出现错误信息INPUT past end。

也就是说，小于最大记录号的记录，倘若未用PUT命令写入数据，则是以哑形式存在，哑记录通常写入16进制的FF。

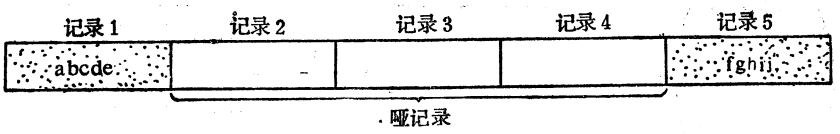



图 6.3.5 哑记录的形式

## 6.4 随机文件记录的追加

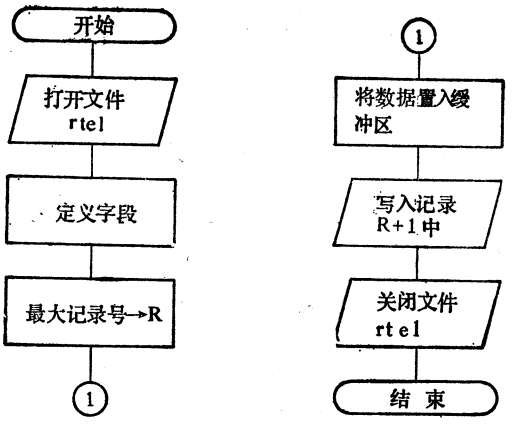
尽管随机文件的OPEN命令，没有APPEND方式，但可简单地追加记录。

（1）对电话簿文件追加记录

程序表6.4.1是对电话簿文件rtel追加新记录的程序。

程序表 6.4.1 电话簿文件记录追加程序

```
10 OPEN "rtel" AS #1
20 FIELD #1,5 AS NM$,10 AS TEL$
30 R=LOF(1)
40 LSET NM$="YI"
50 RSET TEL$="341-9696"
60 PUT #1,R+1
70 CLOSE #1
```




程序表 6.4.1 的流程图

为了检验文件终止，再次使用了读全部记录时使用过的LOF 函数，参见30行。即将rtel最大记录号（此时为 3）代入R，实际写入记录是在60行。

60 **PUT** #1, R+1

这样便向R+1（即 4）号记录，写入了新的数据，也就是在当前文 件 的末尾，又追加了新记录。

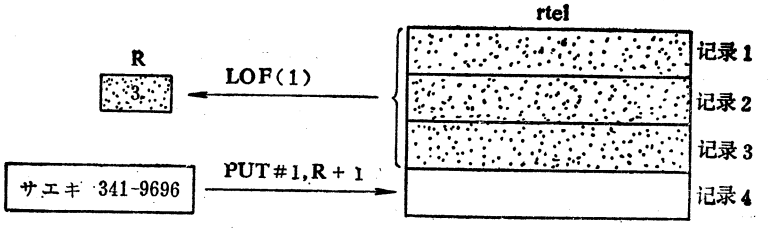



图 6.4.1 追加记录

执行该程序，则文件rtel发生如下图所示的变化。

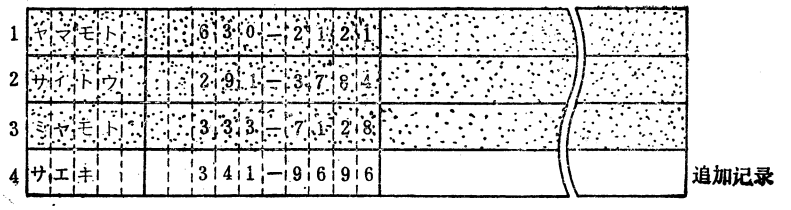



图 6.4.2 追加过记录的文件

## 6.5 随机文件记录的修改

由于随机文件的记录长度是固定的，因而便于修改记录。另一方面，为了从物理上删除或插入记录，必须进行文件复制，这和顺序文件的情况是一样的。另外，由于记录长度是固定的，因而可以采用另一种手法，即对删除的记录做一记号，以标示该记录已被删除。待要

插入时，则将新的记录追加到标有删除记号的记录位置上。

**（1）修改电话簿文件的记录**

表6.5.1为修改电话簿文件1号记录的程序。

**程序表 6.5.1 修改的电话簿文件记录**

```
10 OPEN "rtel" AS #1
20 FIELD #1,5 AS NM$,10 AS TEL$
30 LSET NM$="ミヨシ"
40 RSET TEL$="711-1234"
50 PUT #1,1
60 CLOSE #1
```

从上述程序可以看出，没有专门使用新的命令。仅仅指定了记录号，就可以把新的内容写在原来的记录上，这样，也就完成了记录的修改。换句话说，想要修改记录时，只要指定记录号，重新写入新的内容就可以了。

执行程序，则文件rtel发生如图6.5.1所示的变化。

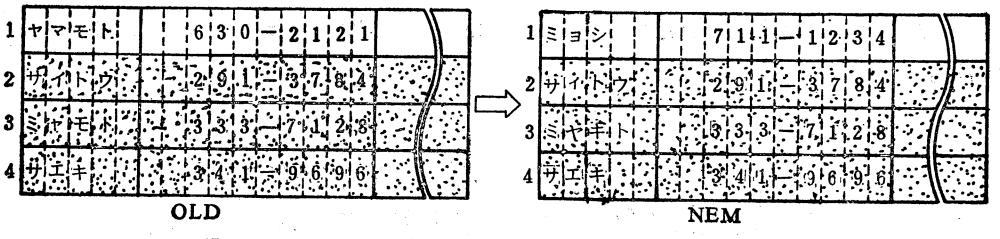



图 6.5.1 记录1的修改

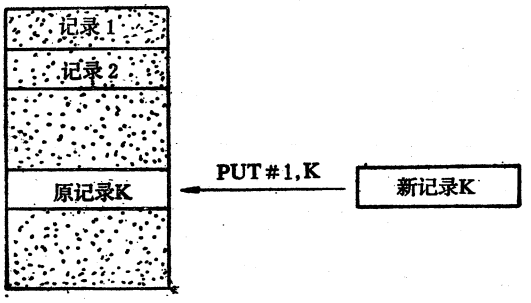



图 6.5.2 记录K的修改

## 6.6 随机文件字段内容的修改

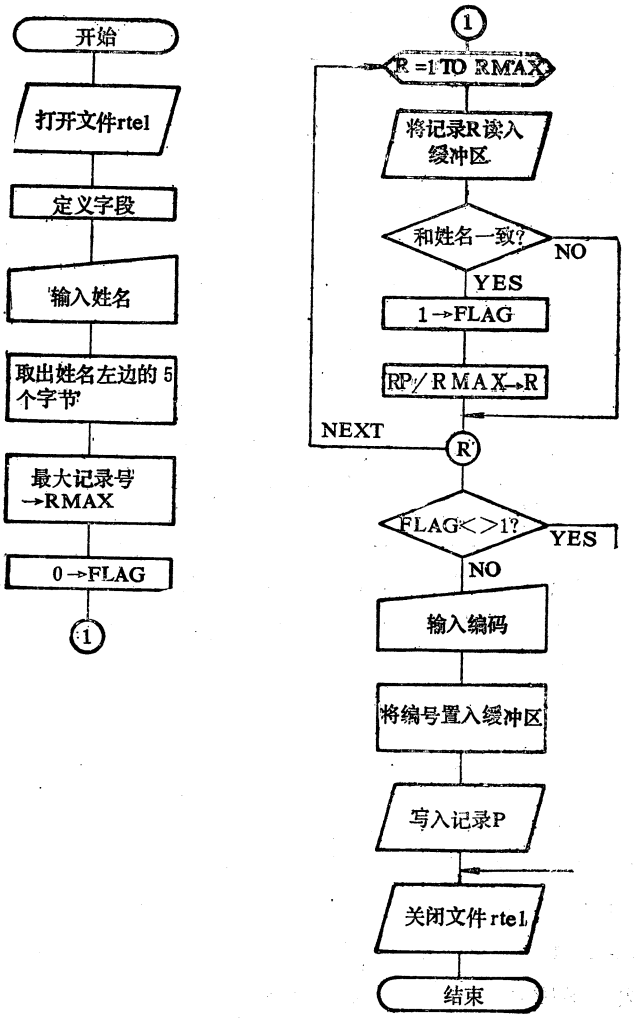
随机文件不仅记录长度是一定的，甚至连字段长度也是一定的，因而也便于对字段的内容进行修改。

**（1）修改电话簿文件字段的内容**

表6.6.1所示为，通过从键盘输入姓名，任意修改，该人电话号码的程序。

程序表 6.6.1 修改电话簿文件字段的程序

```
100 OPEN "rtel" AS #1
110 FIELD #1,5 AS NM$,10 AS TEL$
120 INPUT "姓名";N$
130 N$=LEFT$(N$+"     ",5)
140 RMAX=LOF(1)
150 FLAG=0
160 FOR R=1 TO RMAX
170   GET #1,R
180   IF NM$=N$ THEN FLAG=1:P=R:R=RMAX
190 NEXT R
200 IF FLAG<>1 THEN 240
210 INPUT "编号    ";T$
220 RSET TEL$=T$
230 PUT #1,P
240 CLOSE #1
```




程序表 6.6.1 的流程图

120行为从键盘输入姓名，130行为在输入名字的右面添加空白符号，使其长度达到5个字节。150～190行则检索和输入名相一致的记录，210～230行为用新输入的电话号码替换检索到的纪录中的电话号码。

下面将サイトウ的电话号码291-3784变更为608-1439。

```
run↵
姓名？ 库存数↵
编号    ？ 608-1439↵
Ok
```

画面 6.6.1　表6.6.1的执行结果

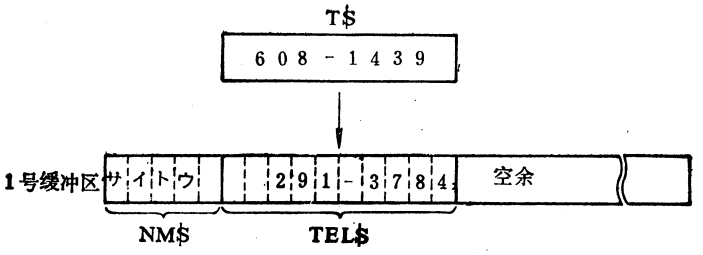



图 6.6.1　字段内容的修改

这样一来，文件rtel则发生如图6.6.2所示的变化。

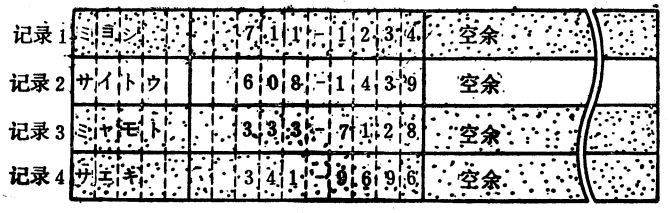



图 6.6.2　修改过的文件rtel

## 6.7　数值数据的读写

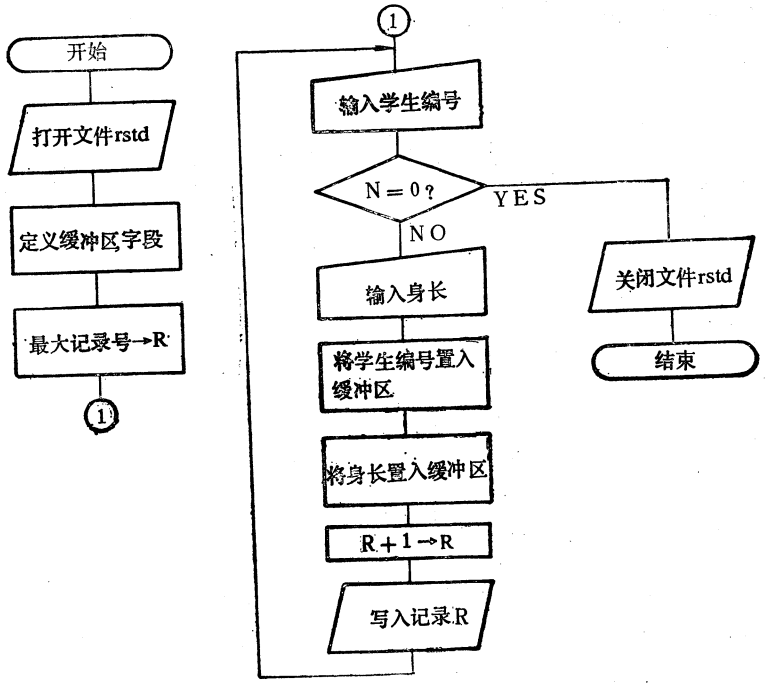
前面曾经介绍过，随机文件的数据全都是以字符型记录的，因此如果处理的是数值型的数据，在写入磁盘之前，一定要将其转换成字符型。另外，从磁盘读取数据之后，亦需要将这些数据恢复成数值型。

**（1）数值数据的写入**

程序表6.7.1为，从键盘输入学生编号和身高，建立随机文件rstd的程序。

程序表 6.7.1 随机文件数值数据写入程序

```
100 OPEN "rstd" AS #1
110 FIELD #1,2 AS N$,4 AS S$
120 R=LOF(1):DEFINT N
130 INPUT   学生编号     ;N
140 IF N=0 THEN 210
150 INPUT   身长   ;S
160 LSET N$=MKI$(N)
170 LSET S$=MKS$(S)
180 R=R+1
190 PUT #1,R
200 GOTO 130
210 CLOSE #1
```




程序表 6.7.1 的流程图

**将数值数据转换为字符串**

- **MKI$** (〈整数型数据〉)
- **MKS$** (〈单精度型数据〉)
- **MKD$** (〈双精度型数据〉)

随机文件备有三个用于将数值数据转换为字符串的函数。

MKI$：将整数型数值转换为双字节字符串。

MKS$：将单精度型数值转换为 4 字节字符串。

MKD$：将双精度型数值转换为 8 字节字符串。

表6.7.1的160和170行为：

160 LSET N$＝MKI$(N)

170 LSET S$＝MKS$(S)

160行将学生编号N（整数）转换为双字节字符串并置入缓冲区字段N$，170行将身高S（单精度）转换为4字节字符串并置入缓冲区字段S$。

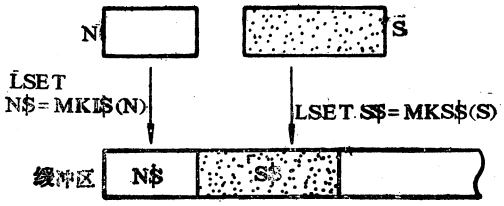



图 6.7.1 将数值转换为字符串

STR$也是一种将数值转换为字符串的函数，虽说都是将数值转换为字符串，但STR$和MKI$的转换方式完全不同。例如，将数值转换成字符串时，如图6.7.2所示，STR$和MRI$分别转换为不同形式。

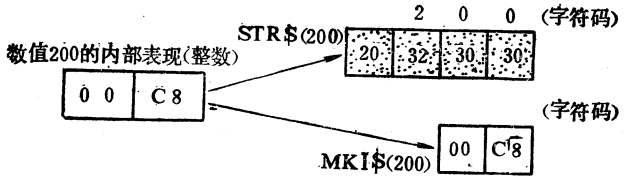



图 6.7.2 STR$和MKI$的差别

即STR$的内部表现完全变了，而MKI$的内部表现不变化，仅仅是从数值型转换为字符型。

执行该程序。

```
run↵
学生编号    ? 1↵
身长  ?  170.8↵
学生编号    ? 2↵
身长  ?  165.8↵
学生编号    ? 3↵
身长  ?  154.2↵
学生编号    ? 0↵
Ok
```

画面6.7.1 表 6.7.1 的执行实例

这样便建立了如图6.7.3所示的文件rstd。

| | | | | |
|---|---|---|---|---|
| 记录1 | (1) | (170. 8) | 空余 | |
| 记录2 | (2) | (165.8) | 空余 | |
| 记录3 | (3) | (154.2) | 空余 | |

( )表示数值的内部表现

图 6.7.3 文件rstd

（2）数值数据的读入

下面是一个从文件rstd 读入全部记录的程序。

程序表 6.7.2 读入随机文件数值的程序

```
100 OPEN "rstd" AS #1
110 FIELD #1,2 AS N$,4 AS S$
120 RMAX=LOF (1)
130 FOR R=1 TO RMAX
140   GET #1,R
150   N=CVI(N$)
160   S=CVS(S$)
170   PRINT N;S
180 NEXT R
190 CLOSE #1
```

**将字符串恢复成数值数据**

- **CVI**（〈双字节的字符串〉）
- **CVS**（〈4 字节的字符串〉）
- **CVD**（〈8 字节的字符串〉）

可分别使用这三个函数，将为 MKI$、MKS$、MKD$转换过的字符串，恢复成原来的数值。

CVI：将双字节的字符串变为整型数。

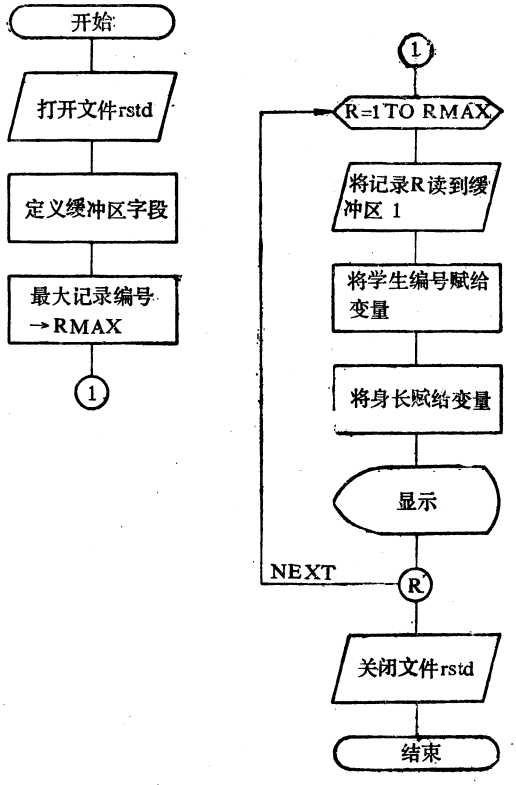
CVS：将 4 字节的字符串变为单精度型数。

CVD：将 8 字节的字符串变为双精度型数。

150 N=CVI(N$)

160 S=CVS(S$)

150行将学生编号从字符串N$转换为整型数值N；160行将身高S$由字符串转换为单精度型的数值S。




程序表 6.7.2 的流程图

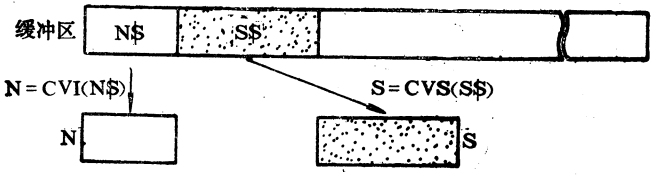



图 6.7.4 字符串转换成数值

执行此程序，则显示如6.7.2所示的画面。

```
run↵
 1  170.8
 2  165.8
 3  154.2
Ok
```

画面 6.7.2 表 6.7.2 的执行实例

现将数值和字符串之间进行转换的函数关系，归纳如下：

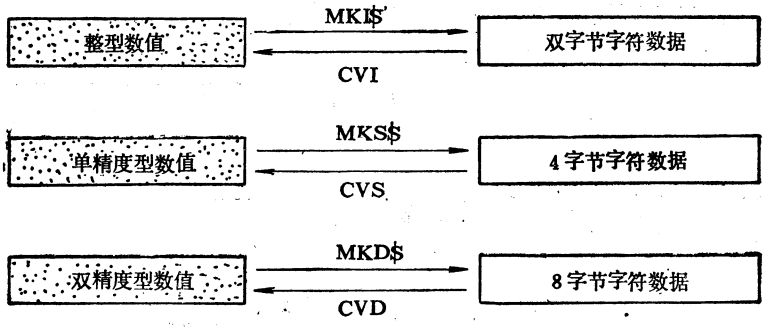



图 6.7.5 使用随机文件时，字符串和数值之间的转换

## 6.8 库存管理程序

5.8节的库存管理程序是将库存文件作成顺序文件，下面介绍的是将其建立成随机文件，并对其进行管理的程序。可以多和5.8节进行对比。随机文件 stock 的各个记录的格式如图6.8.1所示。

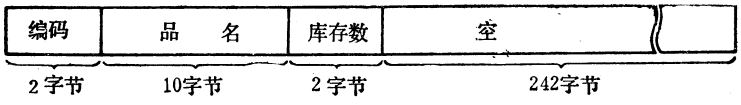



图 6.8.1 文件stock的记录

本程序约定在输入数据时，应使编码与记录号相一致。因此，键入编码，即可读入记录。另外还约定当记录内数据项中的编码与记录号不一致时，表示该记录为删除过的记录或空记录。

执行此程序，则在屏幕上出现如画面6.8.1所示的菜单。

```
****** menu ******
1 建立、追加 ……向库存文件追加记录，或建立文件
2 更新 …… 更新库存数
3 删除 …… 删除库存记录
4 读入 …… 读入记录
0 结束 …… 终止程序运行
?
```

画面 6.8.1 库存管理程序在画面上的菜单

下面对各处理程序加以说明

**库存记录的建立和追加**

输入编码，品名及库存数，则这些数据便写到键入编码的记录之中。最初建立文件时，也是利用建立追加程序来实现的。该位置若已有记录，则将该记录更新。当输入空编码时，（只打回车键），则终止建立、追加处理程序。

例如，照下述方式输入，则建立了3个库存记录。

```
****** 建立·追加 ******
编码  1↵
品名  pencil↵
数量  20↵

编码  2↵
品名  note↵
数量  25↵

编码  4↵
品名  clip↵
数量  50↵

编码  ↵
complete
hit any key
```

画面 6.8.2 库存管理程序的建立和追加

**库存数的更新**

输入编码和入、出库的数目。入库时，其值为＋（正），出库时其值为－（负）。输入空编码，（即只打回车键），则更新处理作业结束。

```
****** 更新 ******
编码  ? 1↵
编码 =1 品名 =pencil      库存数=20
入库(+) 出库(-)数? 10↵
编码 =1 品名= pencil      库存数 =30

编码  ? 2↵
编码 =2 品名=note         库存数=25
入库(+)出库 (-) 数? -3↵
编码 =2 品名=note         库存数=22

编码  ?↵
complete
hit any key
```

画面 6.8.3 库存管理程序的更新

**库存记录的删除**

输入编码，可立即将该记录删除，该记录删除后，其编码即转换为空字符。

```
****** 删除 ******
编码 ? 2↵
deleted
编码 ?↵
complete
hit any key
```

画面 6.8.4. 库存管理程序的删除

**库存记录的读入**

输入记录编码，读入该记录，并加以显示，若输入“all”作为编码，则读入全部记录，此时，若遇到删除或未使用过的记录，则跳过去。

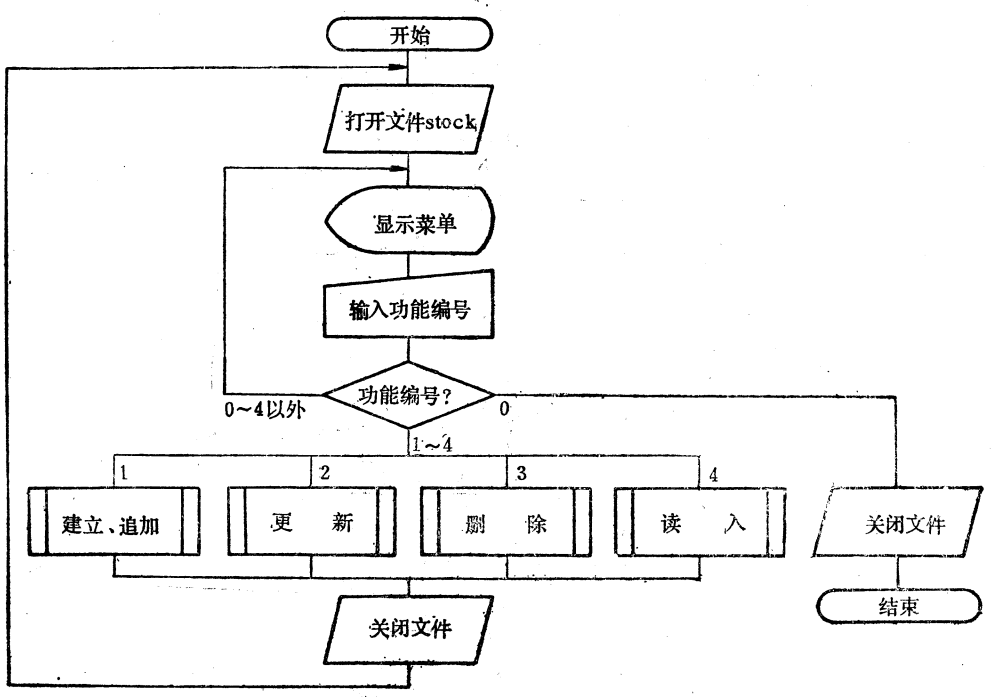
```
****** 读入 *******
code   no.? 1↵
编码          品名          库存数
1             pencil        30
code   no.? all↵
编码          品名          库存数
1             pencil        30
4             clip          50
code   no.?↵
complete
hit any key
```

画面 6.8.5 库存管理程序的显示

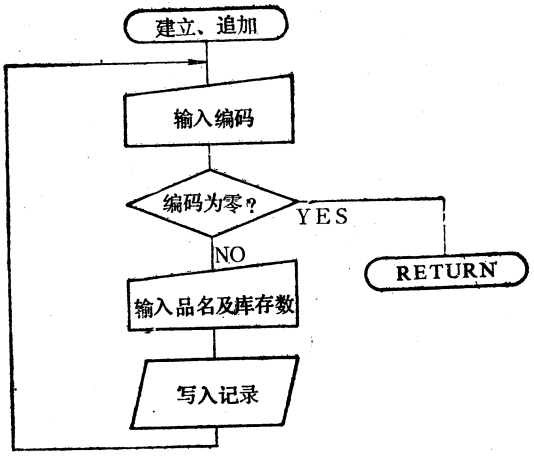
**程序表 6.8.1 随机文件的应用实例库存管理程序**

```
100 '*********************************
110 '*          库存管理程序表          *
120 '*********************************
130  OPEN "stock" AS #1
140  FIELD #1,2 AS CD$,10 AS NM$,2 AS Q$
150  CLS
160  PRINT "****** menu ******
170  PRINT "1 建立、追加
180  PRINT "2 更新
190  PRINT "3 删除
200  PRINT "4 读入
210  PRINT "0 结束
220  INPUT F
230  IF F=0 THEN CLOSE #1:END
240  IF F<0 OR F>4 THEN 160
250  CLS
260  ON  F GOSUB 320,420,800,590
270  PRINT "complete"
280  PRINT "hit any key"
290  IF INKEY$= "" THEN 290
300  CLOSE #1
310  GOTO 130
320
330  PRINT "****** 建立、追加  ******"
340  C$="":INPUT "编码  ;C$
350  IF C$="" THEN RETURN
360  INPUT "品名  ;NX$
370  INPUT "数量  ";Q
380  LSET CD$=C$:LSET NM$=NX$:LSET Q$=MKI$(Q)
390  PUT #1,VAL(C$)
400  PRINT
410  GOTO 340
420
430  PRINT "****** 更新  ******"
440  RMAX=LOF(1)
450  C$="":INPUT "编码  ";C$
460  IF C$="" THEN RETURN
470  IF VAL(C$)>RMAX OR VAL(C$)<1 THEN 450
480  GET #1,VAL(C$)
490  IF VAL(C$)<>VAL(CD$) THEN 450
500  PRINT "编码 =";C$;" 品名 =";NM$;" 库存数 =";CVI(Q$)
510  INPUT "入库(+) 出库(-)数  ;QN
520  Q=CVI(Q$)
530  Q=Q+QN
540  LSET Q$=MKI$(Q)
550  PRINT "编码 =";C$;"  品名 =";NM$;" 库存数 =";CVI(Q$)
560  PUT #1,VAL(C$)
570  PRINT
580  GOTO 450
590
600  PRINT "****** 读入  *******"
```

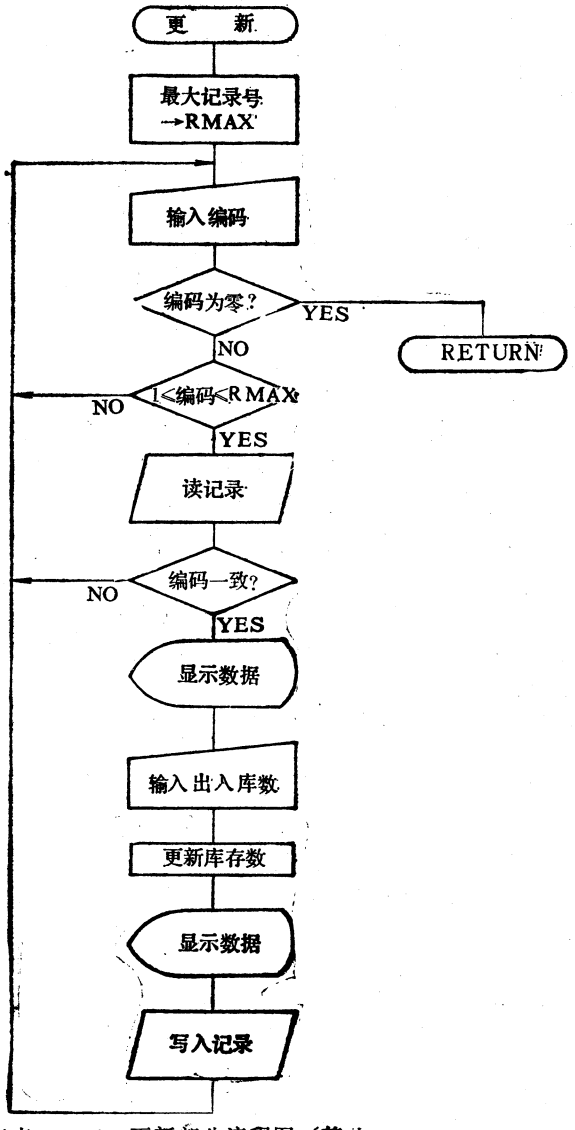
```
610  RMAX=LOF(1)
620  C$="":INPUT "code   no.";C$
630  IF C$="" THEN RETURN
640  IF C$="all" THEN GOSUB 710:GOTO 620.
650  IF VAL(C$)>RMAX THEN 620
660  GET #1,VAL(C$)
670  IF VAL(CD$)<>VAL(C$)  THEN 620
680  PRINT "编码  ","品名  ","  库存数"
690  PRINT CD$,NM$,CVI(Q$)
700  GOTO 620
710
720  PRINT "编码"  ","品名  ","库存数
730  RMAX=LOF(1)
740  FOR I=1 TO RMAX
750    GET #1,I
760    IF VAL(CD$)<>I    THEN 780
770    PRINT CD$,NM$,CVI(Q$)
780  NEXT I
790  RETURN
800
810  PRINT "****** 删除  ******"
820  RMAX=LOF(1)
830  C$="":INPUT "编码  ";C$
840  IF C$ ="" THEN RETURN
850  IF VAL(C$)>RMAX OR VAL(C$)<1 THEN 830
860  GET #1,VAL(C$)
870  LSET CD$="  "
880  PUT #1,VAL(C$)
890  PRINT "deleted"
900  GOTO 830
```



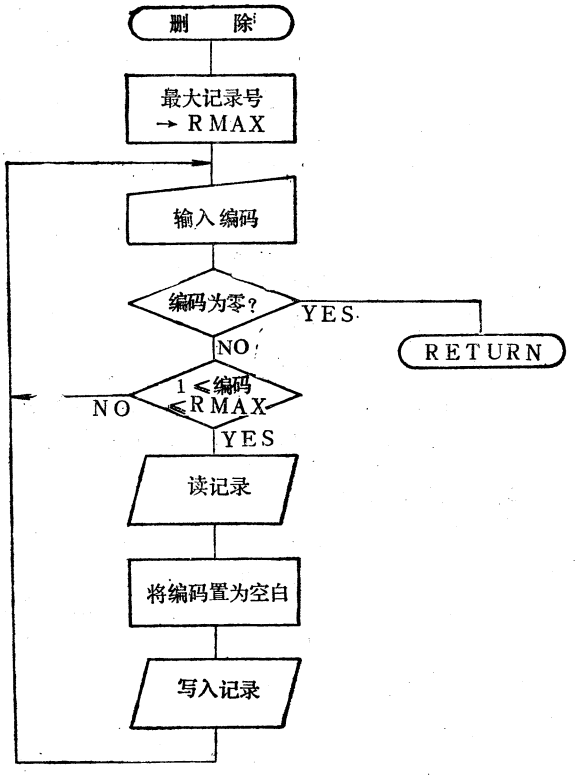

程序表 6.8.1　主控程序流程图（其一）



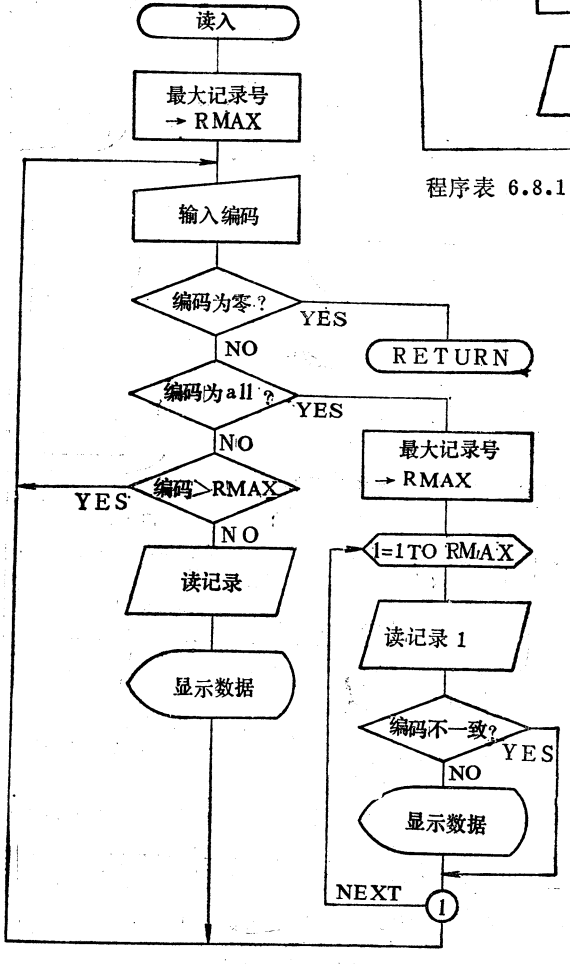

程序表 6.8.1 建立、追加部分流程图（其二）




程序表 6.8.1 更新部分流程图（其二）




程序表 6.8.1　删除部分流程图（其四）




程序表 6.8.1　显示部分流程图（其五）

# 6.9 随机文件汇总

通过以上学习，大概能够对随机文件有所了解了吧，这一章出现了许多新的命令，头脑当中可能有些混乱。下面，我们把随机文件的特征和所使用的命令加以汇总：

**随机文件的特点**

- 记录长度固定为256个字节（定长记录方式）。
- 数据的读写顺序是任意的。
- 通过指定记录编号，可迅速访问所要查寻的记录。
- 便于对记录进行更新。
- 由于记录当中存在有空白区，因而造成浪费，使得文件变大。
- 与顺序文件相比较，读写步骤比较复杂。

**随机文件的命令**

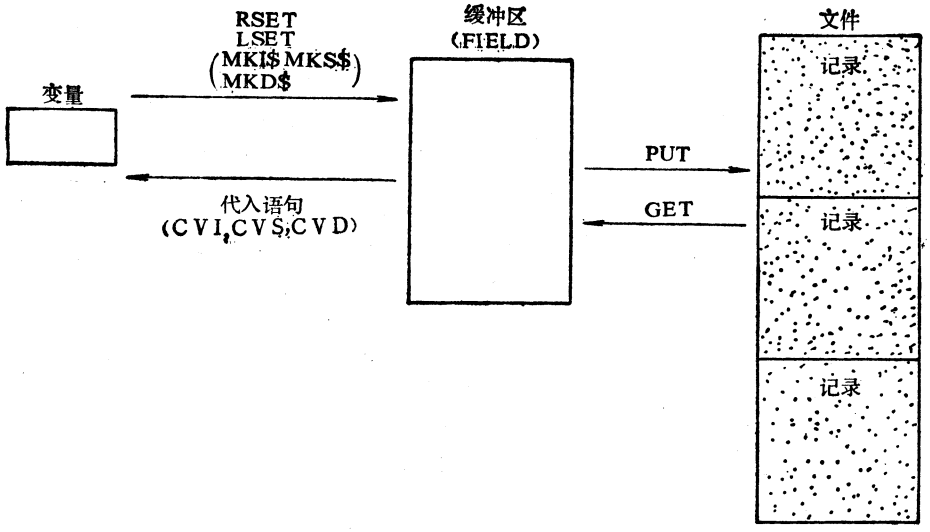



图 6.9.1 随机文件的读写

| | | |
|---|---|---|
| 打开随机文件 | PC | OPEN 〈文件指定〉AS#〈文件编号〉 |
| | FM | OPEN"R",#〈文件编号〉,〈文件指定〉 |
| 关闭文件 | PC | CLOSE[#〈文件编号〉,…] |
| | FM | CLOSE[#〈文件编号〉,…] |
| 定义缓冲区字段 | PC | FIELD#〈文件编号〉,〈字段长度〉AS〈字符变量〉[,〈字符长度〉AS〈字符变量〉,…] |
| | FM | FIELD#〈文件编号〉,〈字段长度〉AS〈字符变量〉[,〈字段长度〉AS〈字符变量〉,…] |
| 将缓冲区内的数据写入文件 | PC | PUT#〈文件编号〉,[〈记录编号〉] |
| | FM | PUT#〈文件编号〉,[〈记录编号〉] |
| 将文件里的数据读到缓冲区 | PC | GET#〈文件编号〉,[〈记录编号〉] |
| | FM | GET#〈文件编号〉,[〈记录编号〉] |

| | | |
|---|---|---|
| 将字符数据从左边起置入缓冲区字段 | PC | LSET〈字符变量〉=〈字符串〉 |
| | FM | LSET〈字符变量〉=〈字符串〉 |
| 将字符数据从右边起置入缓冲区字段 | PC | RSET〈字符变量〉=〈字符串〉 |
| | FM | RSET〈字符变量〉=〈字符串〉 |
| 将2字节的字符串变为整数值 | PC | CVI(〈2字节的字符串〉) |
| | FM | CVI(〈2字节的字符串〉) |
| 将4字节的字符串变为单精度实数值 | PC | CVS(〈4字节的字符串〉) |
| | FM | CVS(〈4字节的字符串〉) |
| 将8字节的字符串变为双精度实数值 | PC | CVD(〈8字节的字符串〉) |
| | FM | CVD(〈8字节的字符串〉) |
| 将整数值转换为2字节的字符串 | | MKI$(〈整数数据表示符号〉) |
| | | MKI$(〈整数数据表示符号〉) |
| 将单精度实数值转换为4字节的字符串 | | MKS$(〈单精度数据表示符号〉) |
| | | MKS$(〈单精度数据表示符号〉) |
| 将双精度实数值转换为8字节的字符串 | | MKD$(〈双精度数据表示符号〉) |
| | | MKD$(〈双精度数据表示符号〉) |
| 得到随机文件的最大记录号 | | LOF(〈文件编号〉) |
| | | LOF(〈文件编号〉) |

# 7. 程序的结合

## 7.1 程序结合的优点

假使程序很短，比如只有几行或者100行，那么直接通过键盘输入该程序并不是什么复杂的事情。然而，当程序很长时，比如有1000行以上，再来做这件事情就不那么简单了。

通常在编写一个大程序时，都是先将该程序分成若干个部分，各个部分独立进行编写、调试。待各部分分别通过以后，再把它们连接到一起，从而构成一个程序。采用这种方法，程序总流程清晰易懂，便于调试，并且可以由几个程序员一起共同编制，比起一个人单独编写，无疑提高了工作效率，节省了时间。

另外，还要考虑到程序过大，内存容纳不下的情况。这时，只有把程序分割成若干个模块，执行时，按顺序将有关模块调入内存加以执行。

程序结合的方法有两种，一种是执行前在内存中的静结合，另一种是执行时，按顺序进行的动结合，对于这两种结合分别使用以下命令：

**静结合**：①使用RENUM和MERGE命令。

**动结合**：①使用RUN和LOAD命令。

②使用CHAIN命令。

## 7.2 静结合（RENUM，MERGE）

我们已经学习过MERGE命令，并且对 RENUM 命令也有所了解，究竟应该怎样做，大体上是能够想象出来的。现在让我们再来复习一下前面的内容。

（1）**先执行RENUM，然后执行MERGE**

设如图7.2.1所示，有两个程序A，B。现准备在程序A后面接上程序B，构成一个新的程序C。

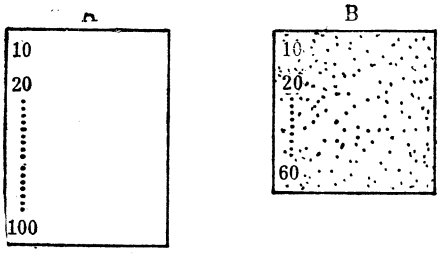



图 7.2.1 程序A和B

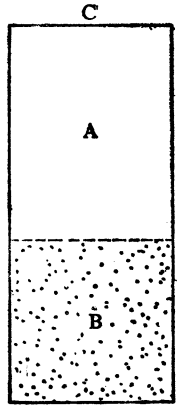



图 7.2.2 结合后形成的程序C

首先，采用ASCII代码方式，将A存储到磁盘上，再将B装入内存。然后，用RENUM命令对B进行重新编号。

**renum 110**

即将B的语句号重新分配为110，120……，接着执行命令

**merge "A"**

这样，A和B就在内存中结合到一起，从而产生了新的程序C。

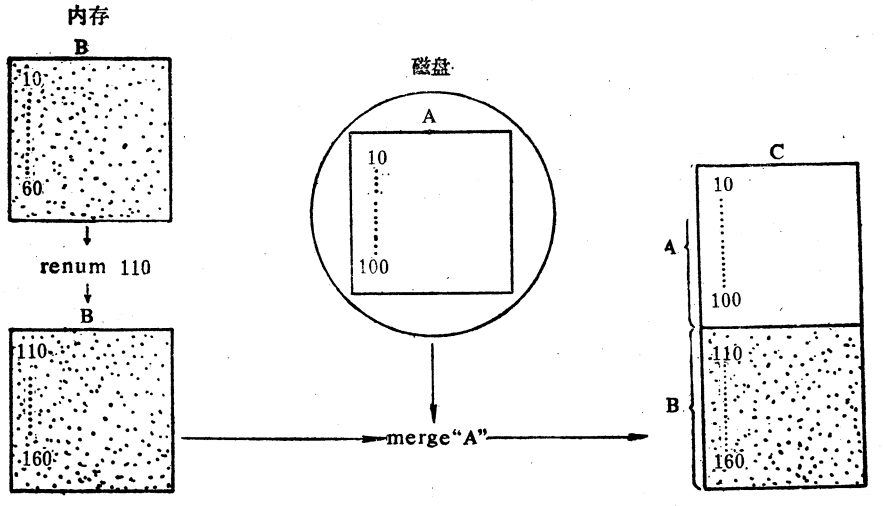



图 7.2.3 程序A和B的结合

采用这种方法建立的程序，其大小受到内存容量的限制，也就是说，用这种方法不可能建立超过内存容量的大程序。

对于这个问题，可以采用下述的动态结合法加以解决。

## 7.3 动结合 (LOAD, RUN)

在第三章曾经介绍过LOAD命令和RUN命令，并指出这两个命令主要是用于装入和执行程序。这里我们将介绍有关这两个命令的另一种使用方法，即对LOAD命令和RUN命令做R选择指定的方法。

**LOAD〈文件指定〉, R**

**RUN〈文件指定〉, R**

这两种命令在功能上完全相同，它们的作用是将程序从磁盘装入内存后立即加以执行。但假若仅仅如此的话，则和不带R选择的RUN命令完全一样了，事实上它们之间是有着很重要的差别的。不作R选择指定，装入程序时，该时刻之前已打开的文件将全部关闭。然而，若进行了R选择指定，则已打开的数据文件将不关闭，而是继续保持打开状态，表7.3.1列出了这些命令的不同之处。

下面，让我们在程序中来实际使用一下这些命令。要想将存储在磁盘中的程序文件X装入内存并加以执行，可以有下面几种不同的方法：

ⓐ run "X"

ⓑ run "X", r

表 7.3.1

| | 处　理 | 处于打开状态的数据文件 |
|---|---|---|
| LOAD 〈文件指定〉 | 装入 | 关闭 |
| LOAD 〈文件指定〉,R<br>RUN 〈文件指定〉,R | 装入并执行 | 继续保持打开状态 |
| RUN 〈文件指定〉 | 装入并执行 | 关闭 |

ⓒ LOAD "X", r

使用ⓐ法，则已打开的数据文件将全部关闭。ⓑ和ⓒ的功能完全一样，即处于打开状态的数据文件继续保持打开状态，同时将X调入。

此外，不管是采用ⓐ，ⓑ，ⓒ三种方法中的哪一种方法，调入程序时，内存中原有的内容均被清除。因此，若需要传递数据的话，就需要事先将数据存放到数据文件之中。特别是使用随机文件传递数据时，若采用ⓑ和ⓒ法，那么在调入新程序以后，就不必再重新打开数据文件。

**（1）使用顺序文件的数据传递**

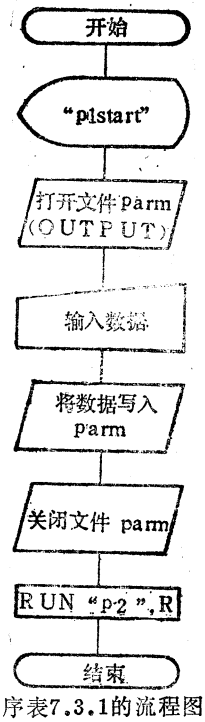
下面的一段程序是在执行p1的过程中调用 p2。即，通过程序 p1 接收键盘输入的数据，然后调用程序p2来显示这些数据。

**程序表 7.3.1　使用顺序文件传递数据的示范程序a**

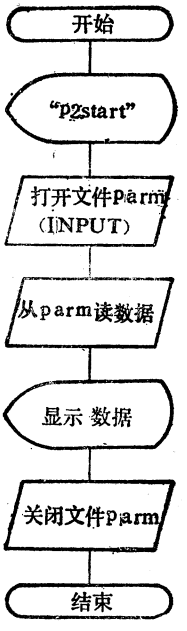
```
10 REM p1
20 PRINT "p1 start
30 OPEN "parm" FOR OUTPUT AS #1
40 INPUT A$,B$
50 PRINT #1,A$;",";B$
60 CLOSE #1
70 RUN "p2",R
```

**程序表 7.3.2　使用顺序文件传递数据的示范程序b**

```
10 REM p2
20 PRINT "p2 start"
30 OPEN "parm" FOR INPUT AS #1
40 INPUT #1,A$,B$
50 PRINT A$,B$
60 CLOSE #1
70 END
```




程序表7.3.1的流程图




程序表7.3.2流程图

程序p1是将键入的数据写入顺序文件parm，程序p2是将写入文件parm中的数据读出，并在屏幕上显示出来。

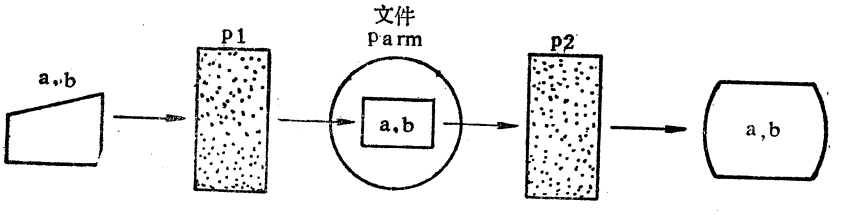



图 7.3.1 经由文件传递数据

对表7.3.1所列程序的70行作如下改动，效果一样

**70 LOAD "p2", R**

程序p1和p2都是以顺序文件的方式打开文件parm的，由于在程序p1的最后关闭了文件parm，所以在程序p2中又用INPUT方式打开该文件。

由于必须在程序p1中将数据文件关闭掉，所以也可以对p1程序稍加改动，改后的程序如表7.3.3所示。

上述改动即省略了70行RUN命令的R选择，这样一来，文件将会自动关闭，因而也就不再需要使用60行的CLOSE命令了。

下面，先将程序p2存入磁盘，然后执行程序p1。

**程序表 7.3.3 顺序文件数据传递示范程序c**

```
10 REM p1
20 PRINT "p1 start"
30 OPEN "parm" FOR OUTPUT AS #1
40 INPUT A$,B$
50 PRINT #1,A$;",";B$
70 RUN "p2"
```

修正

```
p1 start
? a,b↵
p2 start
a              b
Ok
```

画面 7.3.1 顺序文件的数据传递

程序执行结果，表明键入的a和b，可以显示在屏幕上。此时，内存中只保留有程序p2。

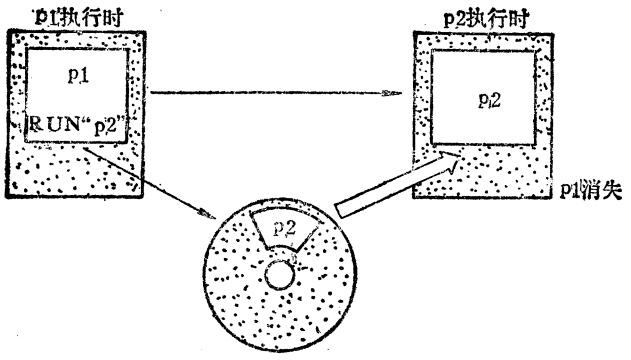



图 7.3.2 原来的程序消失

（2）使用随机文件的数据传递

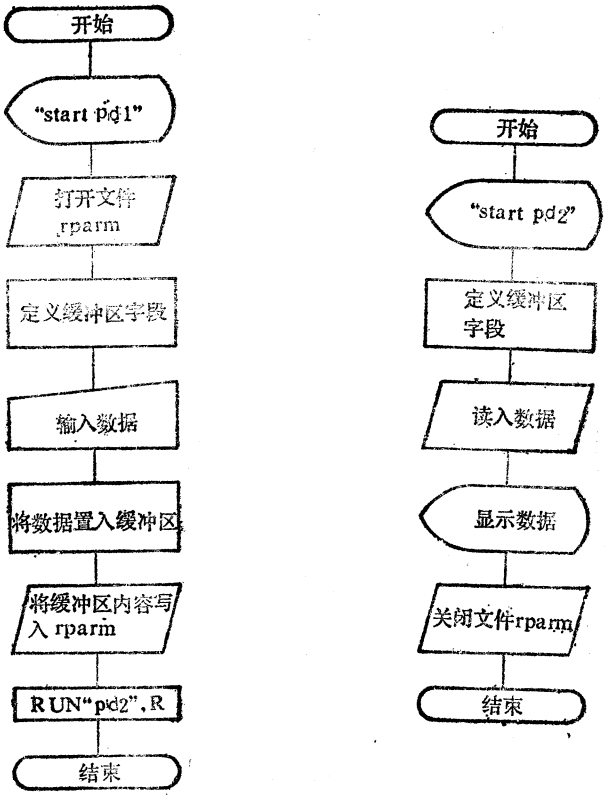
让我们再来编写一个通过随机文件传递数据的程序。先利用程序pd1输入任意数据，然后用程序pd2来显示这些数据。

程序表 7.3.4 利用随机文件数据传递的示范程序a

```
10 REM pd1
20 PRINT "start pd1"
30 OPEN "rparm" AS #1
40 FIELD #1,4 AS A$,4 AS B$
50 INPUT X$,Y$
60 LSET A$=X$
70 LSET B$=Y$
80 PUT #1,1
90 RUN "pd2",R
```

程序表 7.3.5 利用随机文件数据传递的示范程序b

```
10 REM pd2
20 PRINT "start pd2"
30 FIELD #1,4 AS A$,4 AS B$
40 GET #1,1
50 PRINT A$,B$
60 CLOSE #1
70 END
```




程序表7.3.4和程序表7.3.5的流程图

随机文件的OPEN命令，不象顺序文件那样还要指定打开方式，所以我们可以在 pd1 程序中使用带R选择的RUN命令，以保持文件不关闭，而在程序pd2中也就不必再去打开文件rparm了。

执行程序pd1后，屏幕显示的结果，如画面7.3.2所示。

```
start pd1
? a,b↙
start pd2
a              b
Ok
```

画面7.3.2

画面 7.3.2　利用随机文件传递数据的实例

（3）使用缓冲区的数据传递

也可以不将数据写到文件之中，而是仅仅将数据写入缓冲区。

参看下面两个程序pd1x和pd2x

程序表 7.3.6　利用缓冲区传递数据的示范程序a

```
10 OPEN pd1x
20 PRINT "start pd1x"
40 FIELD #1,4 AS A$,4 AS B$
50 INPUT X$,Y$
60 LSET A$=X$
70 LSET B$=Y$
90 RUN "pd2x"
```

程序表 7.3.7 利用缓冲区传递数据的示范程序b

```
10 REM pd2x
20 PRINT "start pd2x"
30 FIELD #1,4 AS A$,4 AS B$
50 PRINT A$,B$
```

我们看到，在程序pd1x中没有pd1中30行的OPEN命令和80行的PUT 命令。同时，在程序pd2x中没有pd2中40行的GET命令和60行的CLOSE 命令。即数据是通过 FIELD 语句定义的 1 号缓冲区来传递的，因此也就可以不使用文件了。

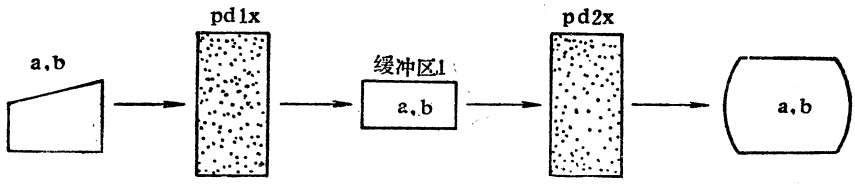



图 7.3.3 经由缓冲区的数据传递

由于这种方法，没有使用文件，所以也就不需要对RUN命令作R选择了。而且由于该方法不对文件进行读写，所以其执行速度较快。

## 7.4 动结合 (CHAIN)

执行LOAD命令和RUN命令，将使当前内存中的程序和变量全部消失。但执行 CHAIN 命令，则可保留当前内存中的程序和变量，同时完成程序的结合。因此，可以使用变量来完成数据传递。另外，即使执行CHAIN命令，文件仍将处于打开状态。

**（1）传递全部变量**

**· CHAIN**〈文件指定〉，〔〈语句行编号〉〕，**ALL**

CHAIN 命令将指定的程序文件装入内存，并使其从指定的行编号开始执行。例如，欲装入程序xyz，并令其由100行开始执行，则打命令

**chain "xyz", 100, all**

若省略行号，则从程序开头处执行。

指定ALL选择，表示当前程序中的每个变量都将传递给程序 "xyz"。另外不指定ALL选择，而是使用COMMON命令，也可以完成仅对特定变量的数据传递。

程序表 7.4.1 CHAIN命令示范程序1a

```
10 REM pc1
20 PRINT "pc1 start"
30 INPUT A$,B$
40 CHAIN "pc2",,ALL
```

程序表 7.4.2 CHAIN命令示范程序1b

```
10 REM pc2
20 PRINT "pc2 start
30 PRINT A$,B$
40 END
```

程序pc1和pc2结合，并实现经由变量A $ 和B $ 的数据传递。

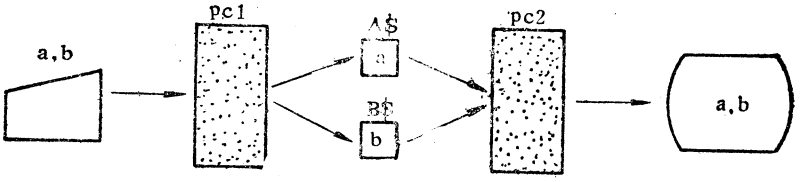



图 7.4.1 经由变量的数据传递

由于没有经由缓冲区和文件传递数据，所以程序相当简单，执行该程序，则屏幕出现7.4.1所示的画面。

```
run↵
pc1 start
? a,b↵
pc2 start
a              b
Ok
```

画面 7.4.1 CHAIN命令，示范程序 1 的执行实例

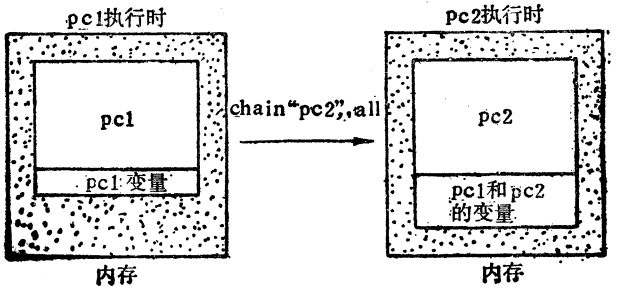



图 7.4.2 保留存在的变量

**（2）传递特定变量**

• **CHAIN**〈文件指定〉[〈行编号〉]

当仅仅想要传递特定的变量时，则可省略掉ALL选择，而用 COMMON 命令把想要传递的变量指定出来。

• **COMMON**〈变量名〉[〈变量名〉……]

例如，想传递变量a $，c，x (100)，d#时，可做如下指定：

**COMMON a $，c，x ( )，d#**

其中x（ ）表示x为数组变量名。另外COMMON命令可以出现在程序中的任何地方，但在执行CHAIN命令之前，必须执行过COMMON命令。否则，COMMON命令不起作用。

参看下面的程序

程序表 7.4.3 CHAIN命令示范程序2a

```
10 REM pc1x
20 PRINT "pc1x start"
30 COMMON A$
40 INPUT A$,B$
50 CHAIN "pc2x"
```

程序表 7.4.4 CHAIN命令示范程序2b

```
10 REM pc2x
20 PRINT "pc2x start"
30 PRINT A$,B$
40 END
```

在程序pc1x中只定义3A $作为被传递的公用变量，因此，B $未能被传递到程序pc2x。执行上述程序的结果，如画面7.4.2所示。

```
run↵
pc1x start
? a,b↵
pc2x start
a
Ok
```

画面 7.4.2 CHAIN命令示范程序 2 的执行实例

显然是由于B $未被传递到程序pc2x中去，所以屏幕上没有显示出B $的内容，因而是一个空字符串。

（3）合并被结合的程序

• **CHAIN MERGE**〈文件指定〉[,〈行编号〉][,**ALL**][,**DELETE**〈范围〉]

方法（1）、（2）虽可传递变量，但当前内存中的程序却消失了。下面介绍一种使原先的程序不消失的连接方法，即在CHAIN命令中进行MERGE选择项指定，于是，就可将内存中的程序和磁盘文件中的程序相连接，从而变成一个程序，但要注意，此时这个被连接的程序必须是以ASCII代码的形式存储在磁盘文件上的。

DELETE选择项的作用，只是在执行MERGE之前，将内存程序中指定范围的语句删除。例如**chain merge"X", 100,all, delete 1000—2000**

这样便可先将内存程序的1000行到2000行的内容删除，然后和程序文件X结合，最后从100行开始执行。由于指定了ALL选择项，所以当前内存中的全部变量都被传递过来了。

带MERGE选择项的CHAIN命令和MERGE命令的不同之处在于，MERGE 命令是在程序执行之前进行程序结合的；而带MERGE选择项的CHAIN命令却是在程序执行的过程中进

行程序结合的。

参看下面的程序

程序表 7.4.5 CHAIN命令示范程序3a

```
10 REM pc1m
20 PRINT "pc1m start"
30 INPUT A$,B$
40 CHAIN MERGE"pc2m",100,ALL
```

程序表 7.4.6 CHAIN命令示范程序3b

```
100 REM pc2m
110 PRINT "pc2m start"
120 PRINT A$,B$
130 END
```

若程序pc2m以ASCII代码形式存储在磁盘上，则在执行程序pc1m的40语句CHAIN命令以后，即在内存中形成表7.4.7所列出的程序，并从100行开始执行，而且所有的变量都进行了传递。

程序表 7.4.7 CHAIN命令示范程序3执行后，内存中驻留的程序

```
10 REM pc1m
20 PRINT "pc1m start"
30 INPUT A$,B$
40 CHAIN MERGE"pc2m",100,ALL
100 REM pc2m
110 PRINT "pc2m start"
120 PRINT A$,B$
130 END
```

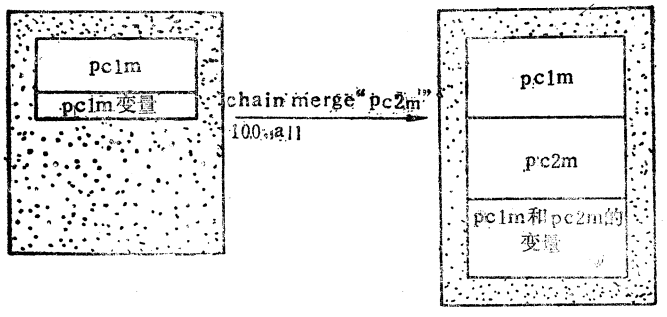



图 7.4.3 保留原来的程序和变量

注意，若省略程序pc1m 40行中CHAIN命令所带的选择项ALL，则变量将不被传递。另外，需要注意的是，在CHAIN语句中，当即不指定MERGE选择，亦不指定ALL选择时DETINT, DEFSNG, DEFDBL, DEFSTR, DEFFN, OPTINT, BASE, ON ERROR, GOTO等语句均不起作用。

现将前面学习过的有关程序结合的各种命令之间的差别汇总于表7.4.1。

表 7.4.1

| | 结合时 | 打开的数据文件 | 原来的变量 | 原来的程序 |
|---|---|---|---|---|
| MERGE | 执行前 | — | — | — |
| RUN(R选择无) | 执行中 | 关闭 | 消失 | 消失 |
| RUN(R选择有) | 执行中 | 打开 | 消失 | 消失 |
| LDAD(R选择有) | 执行中 | 打开 | 消失 | 消失 |
| CHAIN(ALL选择无) | 执行中 | 打开 | 消失 | 注2 |
| CHAIN(ALL选择无) &COMMON | 执行中 | 打开 | 部分保留 | 注2 |
| CHAIN(ALL选择有) | 执行中 | 打开 | 全部保留 | 注2 |
| CHAIN(MERGE选择有) | 执行中 | 打开 | 注1 | 保留 |

注1：取决于ALL, COMMON的有无。
注2：取决于MERGE的有无。

## 7.5 程序结合用命令汇总

| | | |
|---|---|---|
| 合并程序 | PC | MERGE 〈文件指定〉 |
| | FM | MERGE 〈文件指定〉 |
| 装入程序并执行程序，但不关闭文件 | PC | LOAD 〈文件指定〉,R<br>RUN 〈文件指定〉,R |
| | FM | LOAD 〈文件指定〉,R<br>RUN 〈文件指定〉,R |
| 将程序连接起来并加以执行 | PC | CHAIN[MERGE] 〈文件指定〉[,〈行编号〉<br>[,ALL][,DELETE 〈范围〉] |
| | FM | CHAIN[MERGE] 〈文件指定〉[,〈行编号〉]<br>[,ALL][,DELETE 〈范围〉] |
| 执行CHAIN语句时，赋给变量 | PC | COMMON 〈变量名〉[,〈变量名〉,…] |
| | FM | COMMON 〈变量名〉[,〈变量名〉,…] |

# 8. 处理机器语言文件

## 8.1 什么是机器语言文件

所谓机器语言文件，系指以机器语言形式存放的由程序或数据所组成的文件，信息在机器语言文件中的存储格式和它们在内存中的存储格式是完全相同的，也就是将内存中的内容原封不动地照原样记录到磁盘文件中去。

## 8.2 机器语言文件的存储 (BSAEV)

使用BSAVE命令即可将内存中的一部分内容以机器语言的形式转储到磁盘文件上。BSAVE命令对存储的数据到底属于哪一种类型一概不加以检验。也就是说使用BSAVE命令处理文件时，该命令将不对程序和数据加以区分。这样，用户就有责任自己事先对程序和数据加以区分。

**（1）存储机器语言数据**

**·BSAVE**〈文件指定〉〈存储起始地址〉，〈长度〉

将内存中的机器语言程序或数据转储到磁盘文件中去的命令格式如上所示。例如，欲从内存FOOOH开始，将40H字节的内容存放到文件bxfile上，其命令为：

**bsave"bxfile", &he000, &h40**

此时，若磁盘上不存在bxfile文件，则建立新的文件，否则，原bxfile文件的内容将被新存入的内容所更新。这一点和SAVE命令相同。

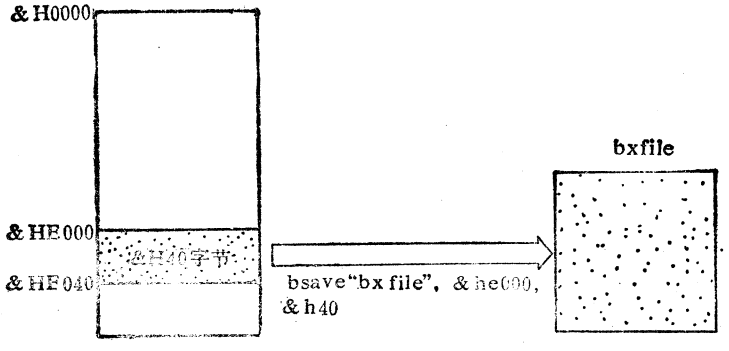



图 8.2.1 机器语言文件的存储

注意，在某些系统中，对一部分内存区不能使用BSAVE命令。例如，PC8801的N88 DISK-BASIC在&H8000～&H83FF区域就不能使用BSAVE。

下面，利用FILE8命令来检查一下用BSAVE所建立的机器语言文件。

```
bsave "bxfile",&he000,&h40↵
Ok
files↵
bxfile*
Ok
```

画面 8.2.1 机器语言文件的存储

检查结果表明，bxfile*已经存储到磁盘上去了。这里的*用来表示机器语言文件。

**BSAVE实例**

**BSAVE"file. bin", &H5000, &H100**

即从5000（十六进制）地址开始，将100（十六进制）字节长的内容转存到1号驱动器的file. bin文件中。

**BSAVE"2:file. bin",8000,100**

表示从8000(十进制)地址开始,将100(十进制)字节长的内容转存到2*驱动器的file. bin文件中。

## 8.3 机器语言文件的装入 (BLOAD)

使用BLOAD命令可将机器语言文件装入内存。BLOAD命令有两种功能，一是仅将文件内容装入内存，二则是在装入文件内容之后，从指定的装入地址开始执行。

**（1）将机器语言文件装入内存**

**·BLOAD**〈文件指定〉[,〈装入地址〉]

比如，若想要将bxfile文件中的机器语言数据从磁盘上装入到D000H开始的一片内存单元，打命令

**bload"bxfile", &Hd000**

如果BLOAD命令中省略了装入地址这一项的话，那就意味着，文件的装入地址和原来用BSAVE命令时的存储地址一样，即

**bload"bxfile"**

此外，装入内容的长度和用BSAVE命令存储时指定的长度相同。

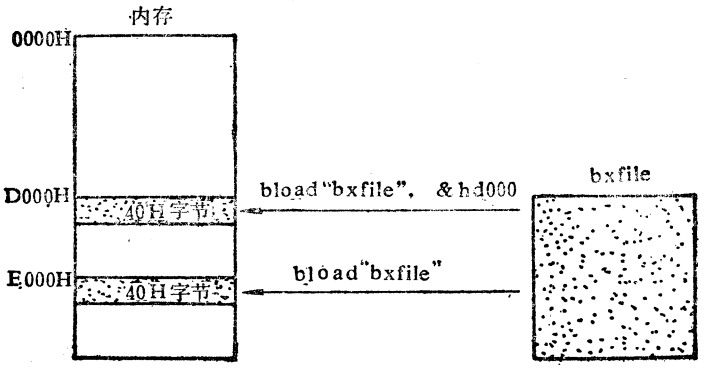



图 8.3.1 机器语言文件的装入

如果想把一个机器语言程序装入到不同于用BSAVE命令存储时所指定的地址的其它一片内存单元的话，那么该程序必须是可再定位的。

装入机器语言程序之后，可用CALL命令执行该程序，表8.3.1为其一例。

上述程序中第10行的CLEAR命令，是将BASIC程序的上限确定在DFFFH，用以防止机

程序表 8.3.1 使用BLOAD命令的程序

```
10 CLEAR,&HDFFF
20 BLOAD "bxfile"
30 GO=&HE000
40 ARG%=10
50 CALL GO(ARG%)
```

器语言程序被BASIC程序所破坏。

50行为利用CALL命令，从头执行所装入的机器语言程序。

**BLOAD实例**

**BLOAD"file. bin"**

**从1号驱动器将机器语言文件"file. bin"装入到由BSAVE所指定的存储地址中去。**

**BLOAD"2:file. bin", &HD000**

**从2号驱动器将机器语言文件"file. bin"装入到D000H地址中去。**

**（2）装入并执行机器语言程序**

**·BLOAD〈文件指定〉[,装入地址>]，R**

为了在将机器语言程序装入内存之后立即加以执行，可在BLOAD命令的最后加上R选择项，例如，为了装入并执行bxfile，则命令格式为

**bLOAD"bxfile", r**

这里省略了装入地址，则取在BSAVE命令中指定的地址，也就是说，文件的装入地址和它在被存储时的地址相同（对这里的bxfile文件来说装入地址是EOOOH）。如果想把bxfile装入到从DOOOH地址开始的内存单元，并同时加以执行，则可输入命令：

**bload"bxfile", &hd000,r**

但此时，该程序应是可再定位的。另外，在这种情况下，程序总是从装入地址开始执行的。如果想要从其它地址开始执行程序的话，那就只能选用不带R选择项的BLOAD命令来装入程序，程序装入之后，再用CALL命令来执行该程序。

显然这种带R选择项的BLOAD命令只适用于机器语言程序。如果对机器语言数据文件使用此命令，则会出现意想不到的情况，千万要注意这一点。

**BLOAD（带R选择）的实例：**

**BLOAD"file. bin", R**

**从1号驱动器将file. bin机器语言程序装入内存，并执行。**

**BLOAD"2:file. bin", &HD000, R**

**从2号驱动器将file. bin机器语言程序装入到内存为D000H的地址，并加以执行。**

# 8.4 机器语言文件使用命令的汇总

| | | |
|---|---|---|
| 存储机器语言程序 | PC | BSAVE〈文件指定〉,〈存储起始地址〉,〈长度〉 |
| | FM | SAVEM〈文件指定〉,〈存储起始地址〉,〈存储结束地址〉,〈执行起始地址〉* |
| 装入机器语言程序 | PC | BLOAD〈文件指定〉[,〈装入地址〉] |
| | FM | LOADM〈文件指定〉[,〈偏移量〉]** |
| 装入机器语言程序，并执行 | PC | BLOAD〈文件指定〉[,〈装入地址〉],R |
| | FM | LOADM〈文件指定〉[,〈偏移地址〉]**,R |

* 使用LOAD命令（带R选择）时，开始执行的地址。

** 实际装入地址和SAVEM命令的存储起始地址之差。

# 9. 磁盘的记录格式

## 9.1 磁盘的内部构造

前面已经介绍过，未进行格式化之前，是不能使用磁盘的，那么格式化后的磁盘究竟变成了什么样子呢？本书将对格式化后，磁盘的内部构造加以说明。

**（1）磁道和扇区**

格式化后的磁盘被分割成了一个个的磁道（track）和扇区。磁道是一些从外向里排列的同心圆，就象田径场上的跑道一样。从外面开始按顺序分别叫做0磁道，1磁道，2磁道……。

Sector是扇区的意思。每个磁道都被割成许多个扇区，扇区是从1开始编号的。

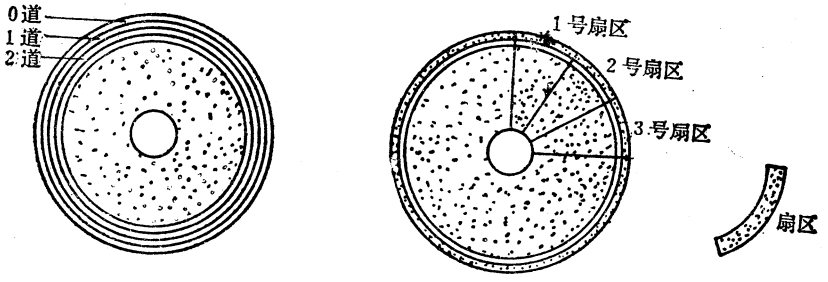



图 9.1.1 磁道和扇区

磁盘的种类不同，磁道和扇区的数目也不一样，五英寸双面双密度磁盘有40个磁道，每一个磁道又分为16个扇区，如图9.1.2所示。表9.1.1列出了五英寸双面双密度、八英寸双面双密度及五英寸单面双密度磁盘的盘面，磁道和扇区的划分情况，仅供参考。

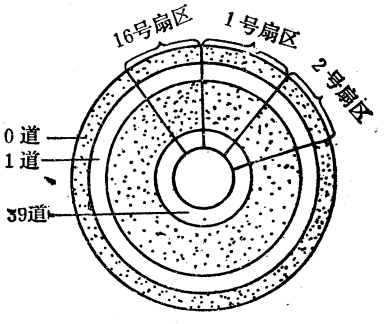



图 9.1.2 五英寸双面双密度盘，磁道和扇区的划分情况

我们把磁盘的盘面称为surface，双面磁盘正面叫做0面，背面叫1面。

系统将数据写入磁盘，或者是从磁盘读取数据时，需要指定磁盘的物理位置。磁盘是以扇区为单位进行物理划分的，所以用磁道号和扇区号，可在众多的扇区中，指定某一扇区。系统是以扇区为单位和磁盘进行数据交换的，一个扇区为256个字节，若以一个字节为单位与磁盘进行数据交换效率太低。

**（2）盘束**

除去磁道和扇区之外，还有另外一种单位叫盘束（cluster）。盘束不同于磁道和扇区，与其说是物理单位，倒不如把它看做是磁盘BASIC管理文件所使用的单位。

盘束这个词是群和堆的意思，具体点说，几个扇区的集合称作磁盘的盘束，例如五英寸

表 9.1.1 磁盘构成（摘自PC-8801用户手册）

| | 盘　面 | 磁道号 | 扇区号 | 扇区字节数 |
|---|---|---|---|---|
| 8英寸双面双密盘 | 0或1 | 0—76 | 1—26 | 256字节 |
| 5英寸双面双密盘 | 0或1 | 0—39 | 1—16 | 256字节 |
| 5英寸单面双密盘 | — | 0—35 | 1—16 | 256字节 |

磁盘8个扇区叫做一个盘束。表9.1.2列出了盘束号和盘面、磁道、扇区的关系。

表 9.1.2 盘束号（摘自PC-8801用户使用手册）

| ① 8英寸双面双密度软磁盘 | | | | ② 5英寸双面双密度软磁盘 | | | | ③ 5英寸单面双密度软磁盘 | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 盘束 | 面 | 道 | 扇区 | 盘束 | 面 | 道 | 扇区 | 盘束 | 道 | 扇区 |
| 0 | 0 | 0 | 1～26 | 0 | 0 | 0 | 1～8 | 0 | 0 | 1～8 |
| 1 | 1 | 0 | 1～26 | 1 | 0 | 0 | 9～16 | 1 | 0 | 9～16 |
| 2 | 0 | 1 | 1～26 | 2 | 1 | 0 | 1～8 | 2 | 1 | 1～8 |
| 3 | 1 | 1 | 1～26 | 3 | 1 | 0 | 9～16 | 3 | 1 | 9～16 |
| ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ |
| 152 | 0 | 76 | 1～26 | 158 | 1 | 39 | 1～8 | 68 | 34 | 1～8 |
| 159 | 1 | 76 | 1～26 | 159 | 1 | 39 | 9～16 | 69 | 34 | 9～16 |

盘束是存储文件的最小单位，对磁盘BASIC来说，文件是以盘束为单位进行分配的。


图 9.1.3 文件和束

换句话说，一个盘束不可能记录两个文件，即，一个磁盘上建立的文件数目不会超过该盘的盘束数。例如五英寸双面双密度磁盘的盘束数为160，那么最多可以建立160个文件*。

（3）管理表

进行系统格式化时，要建立扇区和盘束使用状态管理表，共有以下三种：

① DIR（磁盘文件目录表）

② FAT（文件分配表）

③ ID（盘标志）

**DIR**

DIR为管理磁盘文件用表。磁盘上的每个文件在该表上都占有16个字节，这16个字节记录着，该文件的有关信息，参看下表：

* 实际上，由于存在一些文件管理用表DIR、FAT、ID（下一节将要介绍）等，所以磁盘允许的最大文件数还要小一些。

表 9.1.3 DIR的形式（摘自PC-8801用户手册）

| 字　节 | 内　容 |
|---|---|
| 0～5 | 文件名 |
| 6～8 | 扩展名 |
| 9 | 属性 |
| 10 | 文件起始盘束号 |
| 11～15 | 未使用 |

| 属性值(16进制) | |
|---|---|
| 00 | ASCⅡ码形式 |
| 10 | ASCⅡ码形式，写保护 |
| 40 | ASCⅡ码形式，写检验 |
| 80 | 中间语形式 |
| 90 | 中间语形式，写保护 |
| CO | 中间语形式，写检验 |

若文件名的头一个字节为FF(16进制)，则表示该目录尚未使用，若为0，则表示该文件已被删除。

用SET命令可更改文件的写保护或写检验属性，用ATTR $ 函数可读取文件的属性。

**FAT**

FAT用来表示每个盘束的分配使用情况，以及盘束间的连结关系。

FAT的一个字节表示一个盘束目前的使用状况，例如FAT表中的第5个字节就表示盘束5的使用状况。

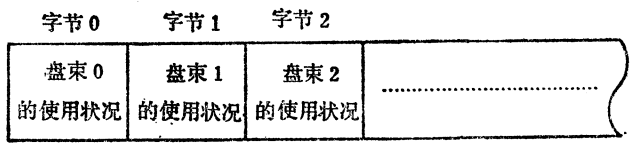



图 9.1.4 FAT的形式

写入各字节中的数值，反映了该盘束的使用状况，具体意义如表9.1.4所示。

表 9.1.4 FAT的内容

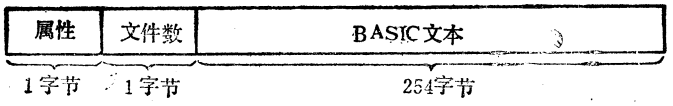
| 字节值（十六进制） | 盘束使用状况 |
|---|---|
| 00—99　8英寸双面<br>00—9F　5英寸双面<br>00—45　5英寸单面 | 使用该束的全部扇区<br>有后续盘束的话，字节值表示后续盘束号 |
| C1—DA　8英寸双面<br>C1—C8　5英寸双面<br>C1—C8　5英寸单面 | 某文件的最后盘束。从该字节值中扣除192(10进制)<br>则表示该束内实际使用的扇区个数 |
| FE | 系统使用的束 |
| FF | 空束 |

FAT占用了三个扇区，内容是完全一样的，只不过是为了安全起见，才采用了这种三重记录的方式*。

* F-BASIC的FAT只用一个扇区记录。

ID

ID是磁盘管理用表，占用一个扇区。如图9.1.5所示。




• 属性
10（16进制）：写保护
40（16进制）：写检验
• 文件数
FF（16进制）：表示没有进行设定
0～F（16进制），能同时打开的文件数
• BASIC文本
电源接通时，自动执行写在BASIC文本内的命令

图 9.1.5 ID的形式

这里的属性为磁盘属性，可用SET命令更改其属性，用ATTR $ 函数读取该属性，文件数初始值为FF(16进制)，若将文件数设定为0～15的数值，则进入磁盘BASIC时，就不再显示下述信息：

**How many files**(0～15)?

而是按设定值来执行。

BASIC文本区存放着BASIC命令，这里写入的若不是00（16进制），则接通电源时，会自动从BASIC文本区取出命令加以执行。

可用厂家提供的系统盘中的服务程序Setinf. $n_{88}$来设置文件数和BASIC文本。也可用第十章介绍的DSKO $ 命令来设置，注意，使用不当，会破坏磁盘中的信息。

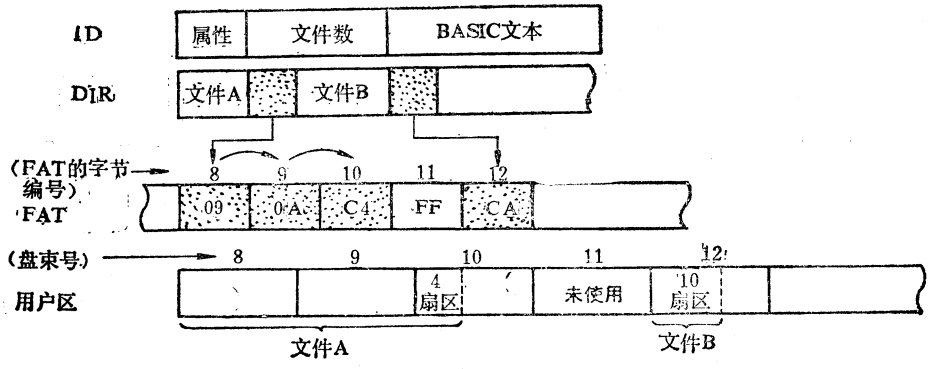



图 9.1.6 管理表的结构

**（4）磁盘BASIC系统**

格式化时，如果制备的是系统盘，则同时拷贝了磁盘BASIC系统和IPL（Initial Program loader）IPL又叫做初始程序的装入程序。在调入磁盘BASIC之前，先要将IPL读入内存，并加以执行。IPL的作用是将磁盘BASIC引导到内存，然后再把控制权交给磁盘BASIC。

IPL和磁盘BASIC系统以磁盘的0面0道开始顺序存放。

表 9.1.5 管理表在磁盘上的位置（摘自PC-8801用户手册）

| | | DIR | ID | FAT | IPL & DISK BASIC | |
|---|---|---|---|---|---|---|
| 8英寸 双面双密度 | 面 | 0 | | | 0 | 1 |
| | 道 | 35 | | | 1～2 | 1 |
| | 扇 区 | 1～22 | 23 | 24～26 | 1～26 | |
| 5英寸 双面双密度 | 面 | 1 | | | 0 | 1 |
| | 道 | 18 | | | 0～1 | |
| | 扇 区 | 1～12 | 13 | 14～16 | 1～16 | |
| 5英寸 单面双密度 | 道 | 18 | | | 0～3 | |
| | 扇 区 | 1～12 | 13 | 14～16 | 1～16 | |

## 9.2 获取磁盘信息（DSKF）

利用函数DSKF可以得到有关磁盘内部构造及空闲区域大小的情报。

**（1）获得磁盘空闲区域大小的信息**

**·DSKF**（〈驱动器号〉）

使用函数DSKF，可获得以盘束为单位的，有关某个驱动器的磁盘空闲区的信息。例如，想了解1号驱动器中磁盘空闲区域的大小，则可键入：

**dskf**(1)

下面检验一下完全未使用过的5英寸单面双密度数据盘的情况。

PRINT dskf(1)↵

```
print dskf(1)↵
68
Ok
```

画面 9.2.1 检查磁盘空闲区

5英寸单面双密度盘共有70个盘束，其中DIR等占用两个，故空闲区为68个盘束。

**DSKF的实例**

**DSKF(1) 可得到有关1号驱动器中磁盘空闲区的信息**

**DSKF(2) 可得到有关2号驱动器中磁盘空闲区的信息**

**（2）获得其它磁盘信息**

**·DSKF**(〈驱动器号〉,〈功能编号〉)

要想获得有关磁盘的其它信息，则需要指定功能编号，表9.21列出了功能编号和函数所带回的信息之间的对应关系。

表 9.2.1

| 功能编号 | 功　　能 | 功能编号 | 功　　能 |
|---|---|---|---|
| 0 | 各面的最大磁道号 | 6 | 每束的扇区数 |
| 1 | 各道的扇区数 | 7 | FAT的起始扇区号 |
| 2 | 单面盘为0，双面盘为1 | 8 | FAT的终止扇区号 |
| 3 | 每一道的盘束数 | 9 | FAT的个数 |
| 4 | 总盘束数 | 10 | 存放ID的扇区号 |
| 5 | 存放目录的磁道号 | | |

例如，键入

**dskf**(1,6)

则可得到1号驱动器磁盘盘束的扇区数。

参看程序表9.2.1所示的程序。

**程序表 9.2.1　应用DSKF（扩展）函数获取磁盘信息的程序**

```
100 INPUT "drive no.";D
110 PRINT
120 PRINT "**** disk information ***"
130 PRINT "sector/track=";DSKF(D,1)
140 IF DSKF(D,2)=0 THEN PRINT " 单面" ELSE PRINT"双面"
150 PRINT "cluster/track=";DSKF(D,3)
160 PRINT "cluster/volume=";DSKF(D,4)
170 PRINT "DIR track no.=";DSKF(D,5)
180 PRINT "sector/cluster=";DSKF(D,6)
190 PRINT "FAT sector no.=";DSKF(D,7);"-";DSKF(D,8)
200 PRINT "the no. of FATs=";DSKF(D,9)
210 PRINT "ID sector no.=";DSKF(D,10)
```

将5英寸双面双密度盘放入1号驱动器，然后执行上述程序则可获得画面 9.2.2 所示的内容。

```
run↵
drive no.? 1↵

**** disk information ***
sector/track= 16 ·····每道扇区数
リョウメン ·····单面或双面
cluster/track= 2 ·····每道的盘束数
cluster/volume= 160 ·····1张盘的盘束数
DIR track no.= 18 ·····DIR所在扇区号
sector/cluster= 8 ·····每一盘束的扇区数
FAT sector no.= 14 - 15 ·····FAT所在的扇区号(道号和DIR相同)
the no. of FATs= 3 ·····FAT扇区数
ID sector no.= 13 ·····ID所在的扇区号(道号和DIR相同)
Ok
```

画面 9.2.2　表9.2.1的执行结果

利用DSKF函数的扩展功能，可以在程序中判别磁盘种类，还可对盘进行相应的处理。下面举例介绍将盘束号转换为物理地址的程序。此程序在计算物理地址时，引用了附录2的公式。

将磁盘放入1号驱动器，而后执行下述程序，输入盘束号。利用DSKF函数来调查磁盘种类，计算物理地址，并将结果显示出来。

```
100  '***** cluster no. to physical addr *****
110  INPUT "cluster no."; C
120  FF=DSKF(1,2)
130  SN=DSKF(1,6)
140  IF SN=26 THEN 260
150  IF FF=1 THEN 210
160  'mini 1 side
170  T=C 2
180  S=(C MOD 2)*8+1
190  F=-1
200  GOTO 300
210  'mini 2 sides
220  F=(C 2)MOD 2
230  T=C4
240  S=(C MOD 2)*8+1
250  GOTO300
260  'standard 2 sides
270  F=C MOD 2
280  T=C 2
290  S=1
300  'output
310  PRINT
320  IF F=-1 THEN 340
330  PRINT "surface no. ="; F
340  PRINT "track no. ="; T
350  PRINT "sector no. ="; S; "—"; S+SN-1
```

例如，将5英寸双面双密度盘装入1号驱动器，执行上述程序，则显示如下结果。

```
run↵
cluster no.? 4↵

surface no.= 0
track no.= 1
sector no.= 1 - 8
Ok
```

**DSKF（扩展）的实例**

**DSKF(1,1) 得到的是1号驱动器中，磁道的扇区数。**

**DSKF(2,3) 得到的是2号驱动器中磁道的盘束数。**

## 9.3 顺序文件的记录状态

用函数LOC、FPOS、LOF可以获得有关磁盘上顺序文件记录状态的信息。

- **LOC**(〈文件编号〉)
- **FPOS**(〈文件编号〉)
- **LOF**(〈文件编号〉)

函数LOC带回的是，到目前为止读写过的扇区数。FPOS带回的是连续扇区号。LOF带回的是以扇区数表示的文件大小。

连续扇区号，是以0面0道1扇区为0号，依次排列下去的。

例如，5英寸双面、单面磁盘的分配情况如表9.3.1和表9.3.2所示：

表 9.3.1

| | |
|---|---|
| 磁面 | 0 0 0 …… 0 0 1 1 …… 1 0 0 …… |
| 磁道 | 0 0 0 …… 0 0 0 0 …… 0 1 1 …… |
| 扇区 | 1 2 3 …… 15 16 1 2 …… 16 1 2 …… |
| 连续扇区号 | 0 1 2 …… 14 15 16 17 …… 31 32 33 …… |

表 9.3.2

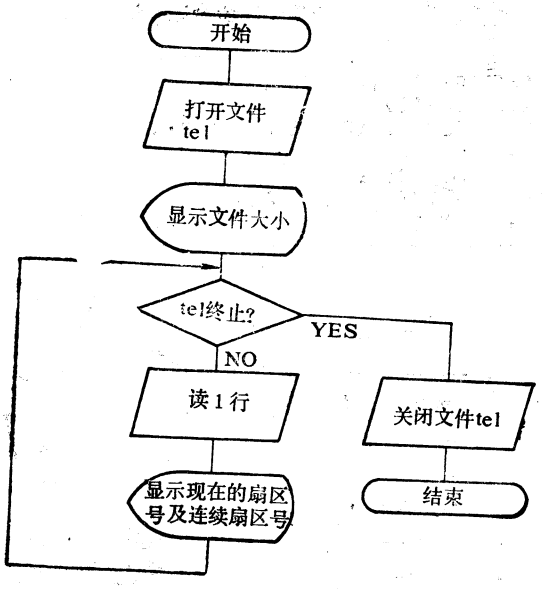
| | |
|---|---|
| 磁道 | 0 0 0 …… 0 0 1 1 …… 1 2 2 …… |
| 扇区 | 1 2 3 …… 15 16 1 2 …… 16 1 2 …… |
| 连续扇区号 | 1 2 …… 14 15 16 17 …… 31 32 33 …… |

**（1）检查电话薄文件的记录状态**

程序表9.3.1所列程序是用来检验电话簿文件tel记录状态的。

**程序表 9.3.1 显示读写过的扇区数和当前连续扇区号的程序**

```
10 OPEN "tel" FOR INPUT AS #1
20 PRINT "total sector no.";LOF(1)
30 IF EOF(1) THEN 80
40 LINE INPUT #1,A$
50 N=N+1
60 PRINT N;LOC(1);FPOS(1)
70 GOTO 30
80 CLOSE #1
```




程序表 9.3.1 的流程图

20行用LOF函数来获得文件tel的大小，并且显示结果。60行用LOC函数来获取读写过的总扇区数，用FPOS函数求得当前的连续扇区号，并且显示出结果。

执行上述程序后，屏幕显示如画面9.3.1所示。

```
total sector no. 1
 1   1   585
 2   1   585
 3   1   585
 4   1   585
 5   1   585
```

画面 9.3.1 程序表9.3.1执行的结果

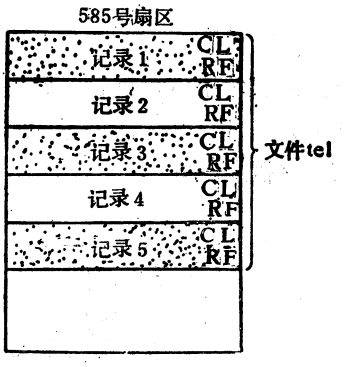



图 9.3.1 文件tel的记录位置

由上述结果可以看出，文件tel有5个记录，全部写在一个扇区（585扇区）内。

## 9.4 随机文件的记录状态

- **LOC(〈文件编号〉)**
- **FPOS(〈文件编号〉)**
- **LOF(〈文件编号〉)**

同样也可以用LOC、FPOS、LOF这三个函数，来检查随机文件的状态。但带回值与顺序文件所代表的意义多少有些不同。但函数的功能并没有变化。LOC带回的是当前的记录号，FPOS带回值是当前的连续扇区号，LOF则带回文件的最大记录号。

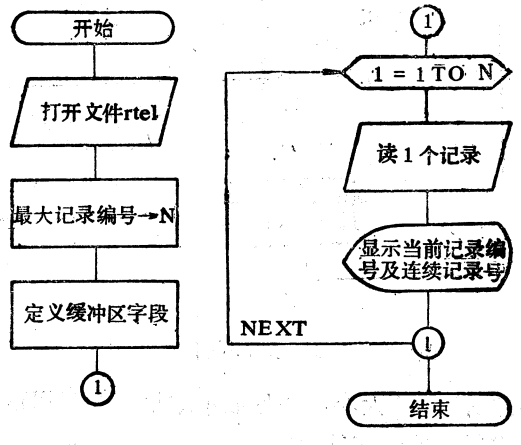
函数LOF已经出现过多次了，这里只不过是为了进行系统的说明，而再次引用。

**（1）检查电话薄文件的记录状态**

下面检查一下电话簿文件rtel在磁盘上的记录位置。具体程序如表9.4.1所示。

程序表 9.41 显示用GET命令读入记录号和连续扇区号的程序

```
10 OPEN "rtel" AS #1
20 N=LOF(1)
30 FIELD #1,128 AS NM$,128 AS TEL$
40 FOR I=1 TO N
50   GET #1,I
60   PRINT LOC(1);FPOS(1)
70 NEXT I
```




程序表 9.4.1 的流程图

程序的20行用LOF函数，将文件rtel的最大记录号赋给变量N。60行利用LOC和FPOS来求得刚刚为GET命令读入的记录号和连续扇区号。

执行该程序，则显现出画面9.4.1所示的结果。

```
 1   537
 2   538
 3   539
 4   540
```

画面 9.4.1 程序表9.4.1执行结果

下图文件rtel的记录位置。

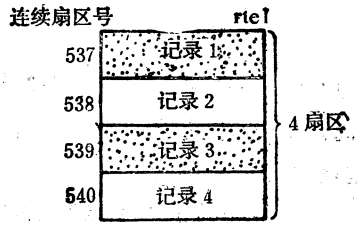



图 9.4.1 文件rtel的记录位置

# 9.5 本章的命令汇总

| | | |
|---|---|---|
| 取得磁盘未使用区域的盘束数 | PC | DSKF(〈驱动器号〉) |
| | FM | DSKF(〈驱动器号〉) |
| 取得磁盘的信息 | PC | DSKF(〈驱动器号〉,〈功能编号〉) |
| | FM | — |
| 取得顺序文件中到目前为止已读写的扇区数 | PC | LOC(〈文件编号〉) |
| | FM | — |
| 取得随机文件中目前的记录号 | PC | LOC(〈文件编号〉) |
| | FM | — |
| 取得随机文件中目前的记录号的下一个编号 | PC | — |
| | FM | LOC(〈文件编号〉) |
| 取得目前的连续扇区号 | PC | FPOS(〈文件编号〉) |
| | FM | — |
| 取得以扇区数表示的顺序文件的大小 | PC | LOF(〈文件编号〉) |
| | FM | — |
| 取得随机文件的最大记录号 | PC | LOF(〈文件编号〉) |
| | FM | LOF(〈文件编号〉) |
| 取得输入到用文件编号表示的输入文件的缓冲区中的字符数 | PC | — |
| | FM | LOF(〈文件编号〉) |

# 10。按物理地址进行读写

## 10.1 按物理地址进行读写

在此之前，我们在读写磁盘上的数据或程序时，并不需要了解这些数据或程序究竟存储在磁盘上的哪一部位，而只需要指定出文件名来就可以了。就是因为在磁盘上存有各种用于管理文件和磁盘记录状态的表，如DIR、FAT、ID等等。磁盘BASIC系统根据这些表，就可以确定对磁盘的哪一部分进行读写操作。但有时由于某种事故会破坏这些管理表，再者，为了进行调试，有时也需要对文件和磁盘进行转储（DUMP）。在上述情况下，倘若能不使用文件名，而是直接指定出磁道号、扇区号就可以对磁盘进行读写的话，那将是很方便的。磁盘BASIC系统为此提供了相应的函数和命令。

## 10.2 从扇区读（DSKI$）

DSKI$是对指定的扇区（256字节）读取数据的函数。所读入的数据，经0号缓冲区赋值给盘定的变量，因此，不必使用OPEN、CLOSE命令。在进入BASIC时，对于提问How many files(0～15)？仅仅回答0也是可以的。

**（1）读取任意扇区**

- **①DSKI$**（〈驱动器号〉，〈磁道号〉，〈扇区号〉）
- **②DSKI$**（〈驱动器号〉，〈盘面号〉，〈磁道号〉，〈扇区号〉）

①　适用于单面磁盘。②　适用于双面磁盘，差别在于对双面盘来说必须指定盘面号。比如，若要读入1号驱动器中的双面盘的0面5道10扇区的内容，则可作如下指定：

**dski$**(1,0,5,10)

**将扇区内容读到变量中去**

下面编制一段读取双面盘任意扇区的程序。

**程序表 10.2.1　利用DSKI$对扇区进行读入的程序**

```
10 INPUT "drive,surface,track,sector";D,F,T,S
20 S$=DSKI$(D,F,T,S)
30 PRINT S$
```

上述程序可将任意双面盘中指定扇区的内容读取到内存并赋给变量S$，然后再将其显示在屏幕上。由于字符串变量的长度最大为255字节，因而磁盘扇区的最后一个字节无法被读入变量。

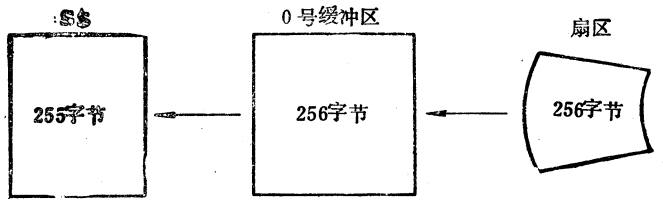



图 10.2.1　将扇区内容读到变量中

现在让我们利用该程序来读取1号驱动器中系统盘上存放DIR表的起始扇区(1 面，18道， 1扇区)，并加以显示，执行结果如画面10.2.1所示。

```
run↵
drive,surface,track,sector? 1,1,18,1↵
backupn88_N     formatn88_I     setinfn88_E     xfilesn88_D     Quartz   _L
Cpaint   _C     DEMO     _M     DEMOUT   _B

Ok
```

画面 10.2.1 读取DIR表的起始扇区

**FIELD语句的使用**

上述程序不可能将扇区的最后一个字节取出。为能取出一个扇区的全部内容 (256 个字节)，我们可以利用FIELD语句将 0 号缓冲区划分成几个区段，相应地，将数据读到几个变量中去。例如，可将 0 号缓冲区划分为两个 128 字节的区段，来读写数据。参见下面的程序。

程序表 10.2.2 使用FIELD语句读入扇区内容的程序

```
10 FIELD #0,128 AS A$,128 AS B$
20 INPUT "drive,surface,track,sector";D,F,T,S
30 S$=DSKI$(D,F,T,S)
40 PRINT A$,B$
```

注意，FIELD语句的缓冲区号必须为 0 。另外，上述程序中的S $ 仍是读入 255 个字节的数据，但在这里仅仅是作为哑变量供 DSKI $ 函数使用而已。执行程序后，即可将被读入扇区的前128字节赋给 0 号缓冲区的A $ 区段，而将后128字节赋给 0 号缓冲区的B $ 区段。

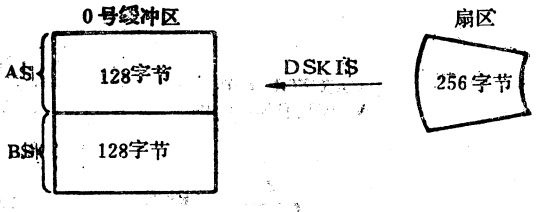



图 10.2.2 将扇区内容分成两部分读入缓冲区

**VARPTR函数的使用**

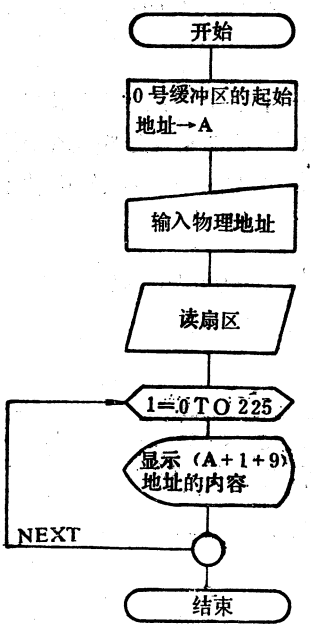
**·VARPTR(#<缓冲区号>)**

使用VARPTR函数可以得到缓冲区的起始地址*。若把该函数与 PEEK 函数结合使用，便可读出 0 号缓冲区的内容。请参看下述程序。

* F-BASIC的VARPTR函数只能获得变量的存储地址。

程序表 10.2.3 利用VARPTR函数读入扇区内容的程序

```
10 A=VARPTR(#0)·····取0号缓冲区的起始地址
20 INPUT "drive,surface,track,sector";D,F,T,S
30 S$=DSKI$(D,F,T,S)
40 FOR I=0 TO 255
50   B=PEEK(A+I+9)·····缓冲区头9个字节为作业区
60   PRINT CHR$(B);
70 NEXT I
80 PRINT
```




程序表 10.2.3 所列程序的流程图

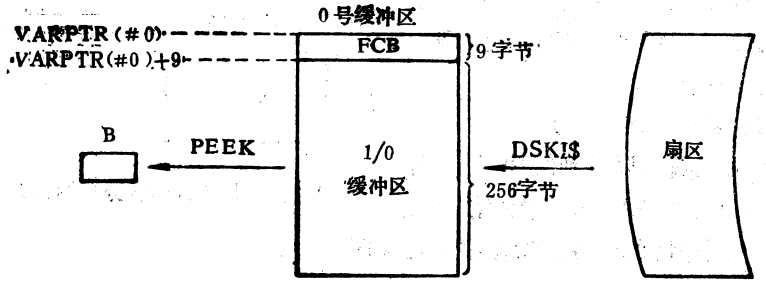



图 10.2.3 利用VARPTR和PEEK取出0号缓冲区内容

在上述程序的第10行中，先用VARPTR函数求出了0号缓冲区的起始地址，然后在50行中用PEEK函数从头开始一个字节一个字节地将0号缓冲区内容取出赋给B。由于缓冲区开头的9个字节是控制信息（FCB），所以跳过不读。

在这里，S$仅仅是作为哑变量使用。

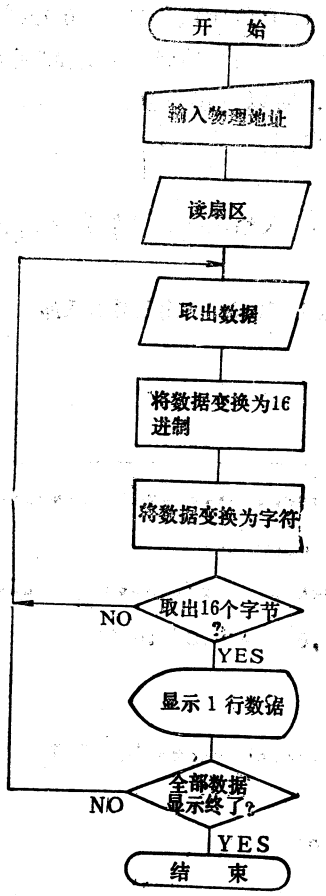
**扇区的转储**

表10.2.4.列出了利用函数VARPTR和PEEK转储扇区内容(字符及16进制数)的程序。

程序表 10.2.4 扇区转储程序

```
100 '*************************************************
110 '*                                               *
120 '*              dump  program                    *
130 '*                                               *
140 '*************************************************
150  DIM A$(20),H$(40)
160  DATA 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F
170  FOR I=0 TO 15
180    READ A$(I)
190  NEXT I
200  A=VARPTR(#0)
210  INPUT "drive,surface,track,sector";D,F,T,S
220  S$=DSKI$(D,F,T,S)
230  FOR I=0 TO 15
240    FOR J=0 TO 15
250      N=PEEK(A+I*16+J+9)
260      GOSUB 410
270      H$(J)=H$
280      IF N<32 OR N>&HF7 THEN H$(J+17)="." ELSE H$(J+17)=CHR$(N)
290    NEXT J
300    H$(16)="  "
310    N=I*16
320    GOSUB 410
330    PRINT H$;" :";
340    FOR J=0 TO 32
350      IF J<16 THEN PRINT H$(J);" "; ELSE PRINT H$(J);
360    NEXT J
370    PRINT
380  NEXT I
390  END
400
410  REM DEC TO HEX
420  N2=INT(N/16)
430  N1=N-N2*16
440  H$=A$(N2)+A$(N1)
450  RETURN
```

利用上述程序，可转储1号驱动器中系统盘上目录（DIR)的起始扇区的内容(即1面、18道、1扇区的内容),并显示出结果。




程序表 10.2.4　所列程序的流程图

```
run↵
drive,surface,track,sector? 1,1,18,1↵
00 :62 61 63 6B 75 70 6E 38 38 80 48 FF FF FF FF FF   backupn88.H....
10 :66 6F 72 6D 61 74 6E 38 38 80 49 FF FF FF FF FF   formatn88.I....
20 :73 65 74 69 6E 66 6E 38 38 80 45 FF FF FF FF FF   setinfn88.E....
30 :78 66 69 6C 65 73 6E 38 38 80 44 FF FF FF FF FF   xfilesn88.D....
40 :51 75 61 72 74 7A 20 20 20 80 4C FF FF FF FF FF   Quartz   .L....
50 :43 70 61 69 6E 74 20 20 20 80 43 FF FF FF FF FF   Cpaint   .C....
60 :44 45 4D 4F 20 20 20 20 20 80 4D FF FF FF FF FF   DEMO     .M....
70 :44 45 4D 4F 55 54 20 20 20 80 42 FF FF FF FF FF   DEMOUT   .B....
80 :FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF   ................
90 :FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF   ................
A0 :FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF   ................
B0 :FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF   ................
C0 :FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF   ................
D0 :FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF   ................
E0 :FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF   ................
F0 :FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF   ................
```

画面 10.2.2　转储DIR起始扇区的内容

**DSKI$　的实例**

**DSKI$(1,5,10)　读入1号驱动器中单面盘的5道10扇区的内容**

**DSKI$(2,1,6,3)　读入2号驱动器中双面盘的1面6道3扇区的内容**

## 10.3 向扇区写入内容（DSKO$）

DSKO $ 的作用与DSKI $ 相反，它可将 0 号缓冲区的内容写到指定的磁盘扇区中去。但要注意，如果一旦把物理地址搞错了，就会破坏磁盘内容。因此，使用前最好先拷贝一块盘作为副本。

（1）向任意扇区写入内容

·①DSKO $ 〈驱动器号〉,〈磁道号〉,〈扇区号〉

·②DSKO $ 〈驱动器号〉,〈盘面号〉,〈磁道号〉,〈扇区号〉

与DSKI $ 函数一样，上面的①适用于单面盘②适用于双面盘。但 DSKO $ 并不是函数

例如，要把 0 号缓冲区的内容写入 1 号驱动器中双面磁盘的 1 面 4 道 2 扇区，即可键入

dsko $ 1,1,4,2

**FIELD语句的使用**

和向随机文件中写入内容时一样，在利用DSKO $ 向磁盘扇区内写入内容之前，要用FIELD语句将 0 号缓冲区划分为几个区段，然后再用LSET或RSET 命令将数据送入 0 号缓冲区。

下面是一段对任意扇区进行清 0 的程序。

程序表 10.3.1 利用DSKO $ 对任意扇区进行清0的程序

```
10 FIELD #0,128 AS A$,128 AS B$
20 LSET A$=STRING$(128,0)  } 缓冲区的内容全部清零
30 LSET B$=STRING$(128,0)  }
40 INPUT "drive,surface,track,sector";D,F,T,S
50 DSKO$ D,F,T,S
```

在上述程序中，第10行将 0 号缓冲区划分为A $ 和B $ 两个区段。20行与30行，将缓冲区的内容全部清零。40行提示用户输入磁盘的物理地址。50行将 0 号缓冲区的内容写入指定的磁盘扇区

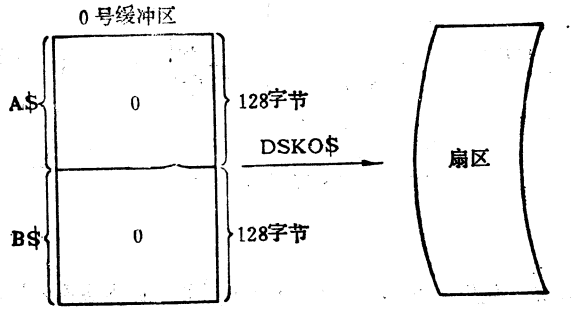



图 10.3.1 将缓冲区数据写入扇区

下面，利用上述程序，将 1 号驱动器内的磁盘的 0 面 6 道 7 扇区清 0 。具体操作如下：

```
run↵
drive,surface,track,sector? 1,0,6,7↵
Ok
```

画面 10.3.1　表10.3.1所列程序的执行实例

注意，上面那段程序仅仅是示范性的，不要随便执行，弄不好会把磁盘中有用的部分冲掉。

**VARPTR函数的使用**

除了使用FIELD语句以外，还可以利用VARPTR函数和POKE语句来将数据写到0号缓冲区，下面对表10.3.1所列程序做修改如下：

**程序表 10.3.2　使用VARPTR函数对任意扇区进行清0的程序**

```
10 A=VARPTR(#0)·····取0号缓冲区的起始地址
20 FOR I=0 TO 255
30   POKE A+I+9,0·····缓冲区头9个字节为作业区
40 NEXT I
50 INPUT "drive,surface,track,sector";D,F,T,S
60 DSKO$ D,F,T,S
```

在上述程序的第10行中，先利用VARPTR函数得到了0号缓冲区的起始地址，然后在30行中利用POKE语句将0号缓冲区一个字节一个字节地全部清0。

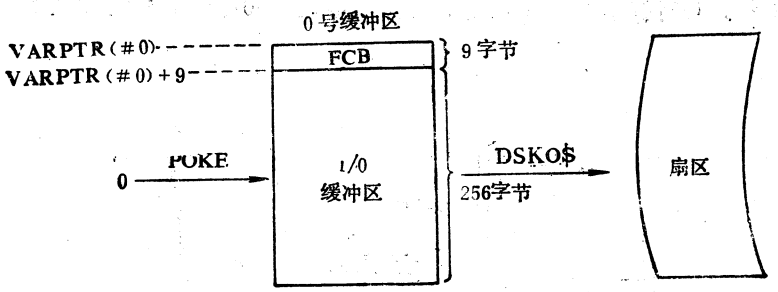
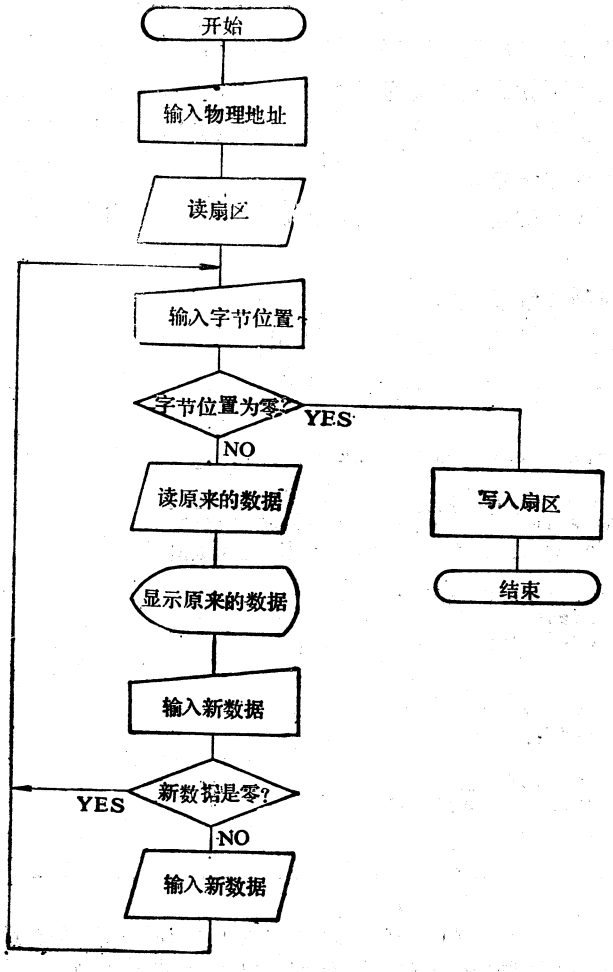



图 10.3.2　使用VARPTR和POKE将0号缓冲区内容写入扇区

**扇区的更新**

下面介绍一个更新扇区内容的程序，该程序的流程图如下：




程序表 10.3.3　所列程序的流程图

**程序表10.3.3　更新任意扇区的程序**

```
100 '*************************************************
110 '*                                               *
120 '*              zap   program                    *
130 '*                                               *
140 '*************************************************
150  A=VARPTR(#0)+9
160  INPUT "drive,surface,track,sector";D,F,T,S
170  S$=DSKI$(D,F,T,S)
180  PRINT
190  B$="";INPUT "byte position";B$
200  IF B$="" THEN 310
210  B=VAL(B$)
220  IF B>255 OR B<0 THEN 180
230  BA=PEEK(A+B)
240  PRINT "old data:";HEX$(BA)
250  ND$="";INPUT "new data";ND$
260  IF ND$="" THEN 180
270  ND=VAL(ND$)
280  IF ND<0 OR ND>255 THEN 250
290  POKE A+B,ND
300  GOTO 180
310  DSKO$ D,F,T,S
320  PRINT "***** zap   end ******"
330  END
```

使用上述程序更新扇区内容，其步骤如下：

① 输入被更新扇区的物理地址。

② 输入该扇区内任意字节的位置（0～255）。若只打入回车键，则执行对扇区的写入，而后程序终止执行。

③ 显示②所指定字节的内容，然后可输入新的内容。若只按回车键，则表明不进行更新。

④ 返回到②继续执行。

该程序的执行过程如画面10.3.2所示。

```
run↙
drive,surface,track,sector? 1,1,2,3↙          扇区物理地址

byte position? 45↙ ..... 输入字节位置
old data:FF ..... 显示原来的数据
neu data? 0↙ ..... 输入新的数据

byte position? &h10↙
old data:FF
neu data? &h23↙

byte position? 13↙
old data:FF
neu data?↙

byte position?↙
******* zap    end ********
Ok
```

画面 10.3.2 程序表10.3.3所列程序的执行实例

上述程序也可以用来对第 9 章中介绍过的ID进行改写。

**DSKO $ 实例**

**DSKO $ 1,6,2 将0号缓冲区的内容写入1号驱动器单面盘的6道2扇区。**

**DSKO $ 2,1,5,3 将0号缓冲区的内容写入2号驱动器双面盘的1面5道3扇区。**

## 10.4 本章的命令汇总

| | | |
|---|---|---|
| 进行对单面磁盘的直接读入 | PC | DSKI $(〈驱动区号〉,〈磁道号〉,〈扇区号〉) |
| | FM | DSKI $(〈驱动器号〉,〈磁道号〉,〈扇区号〉) |
| 进行对双面磁盘的直接读入 | PC | DSKI $(〈驱动器号〉, 〈盘面号〉, 〈磁道号〉, 〈扇区号〉) |
| | FM | DSKI $(〈驱动器号〉, 〈磁道号〉, 〈扇区号〉*) |
| 进行对单面磁盘的直接写入 | PC | DSKO $〈驱动器号〉, 〈磁道号〉, 〈扇区号〉 |
| | FM | DSKO $〈驱动器号〉, 〈磁道号〉, 〈扇区号〉 |
| 进行对双面磁盘的直接写入 | PC | DSKO $〈驱动器号〉, 〈盘面号〉, 〈磁道号〉, 〈扇区号〉 |
| | FM | DSKO $〈驱动器号〉, 〈磁道号〉, 〈扇区号〉* |
| 取得缓冲区的起始地址 | PC | VARPTR(#〈缓冲区号〉) |
| | | — |

* 小型磁盘的扇区数正面为1～16，反面为17～32，标准磁盘的扇区数正面为1～26，反面为27～52。

# 附　　录

## 1．从连续扇区号寻找物理地址

作为取得文件最后的读写位置的函数，虽然有函数FPOS，但用该函数得出的是连续扇区号。要从连续扇区号寻求物理地址，需作如下的运算。由于N60m，N60，N66-BASIC及F-BASIC没有FPOS函数，所以这里就省略了。

**（1）$N_{88}$，N，$N_{80}$磁盘BASIC的场合**

1）小型单面磁盘

```
T＝N￥16
S=(N MOD16)＋1
```

2）小型双面磁盘

```
F＝(N￥16)MOD 2
T＝N￥32
S=(N MOD 16)＋1
```

3）标准双面磁盘

```
F＝(N￥26)MOD 2
T＝N￥52
S=(N MOD 26)＋1
```

**（2）T-BASIC7**

```
F＝(N￥8)MOD 2
T＝N￥32
S=N MOD 8＋((N￥16)MOD 2)*8＋1
```

## 2．从盘束号寻找物理地址

由于DSKO $是直接将0号缓冲区的内容写入磁盘，如不注意就会破坏磁盘的内容，所以让人担心。为了避免磁盘的内容被破坏，可参照DIR和FAT。虽然可以将0号缓冲区的内容写入系统和文件没有使用的扇区，但是由于DIR和FAT是用盘束号表示文件位置的，所以从盘束号来寻找物理地址是非常方便的。从盘束号寻找物理地址其运算如下。

盘束号：C　　盘面号：F

磁道号：T　　起始扇区号：S

**（1）$N_{88}$，N，$N_{80}$磁盘BASIC的场合**

1）小型单面磁盘

```
T＝C￥2
S=(C MOD 2)*8＋1
```

2）小型双面磁盘

```
F=(C￥2)MOD2
```

T＝C￥4
S＝(C MOD2)*8＋1

3）标准双面磁盘

F＝C MOD 2
T＝C￥2
S＝1

（2）$N_{60}$m，$N_{60}$扩展，$N_{66}$-BASIC的场合

T＝C￥2
S＝(C MOD 2)*8＋1

（3）T-BASIC7的场合

F＝CMOD 2
T＝C￥4
S＝((C￥2) MOD 2)*8＋1

（4）F-BASIC的场合

F-BASIC在考虑磁盘的物理地址时，不使用盘面号，正反两面的全部扇区作成连续编号，就用其连续编号来区别正面和反面。例如，小型双面磁盘扇区 1～16号为正面，17～32号就表示为反面，标准双面磁盘扇区1～26号为正面，27～52号就表示为反面。另外，小型双面磁盘把 2 号磁道的扇区1～8号作为0号盘束，标准双面磁盘把1号磁道的 1～26号扇区作为 0 号盘束。

1）小型双面磁盘

T＝(C￥4)＋2
S＝(C MOD 4)*8＋1

2）标准双面磁盘

T＝(C￥2)＋1
S＝(C MOD 2)*26＋1

## 3．不同机种的磁盘BASIC命令表

本书主要是用PL-8801的$N_{88}$磁盘BASIC 来进行说明的，据了解BASIC 由于机种不同存在着“方言”，磁盘BASIC也不例外。这里列有不同机种的磁盘BASIC命令表。当移植程序时供参考（但是，该命令表列出的仅仅是关于磁盘的命令）。表的左边栏中列出的是本书中出现的命令形式，括号中的命令表示是该机种所特有的命令。此外，各命令表中没有说明的命令由于与$N_{88}$磁盘BASIC完全相同，因此，请参照$N_{88}$磁盘BASIC命令表的说明。

下面列出命令表及其对应的机种。

$N_{88}$磁盘BASIC命令表……PC-8801，PC-8801mkII，PC-9801，PC-9801F
$N_{80}$磁盘BASIC命令表……PC-8001mkII
N磁盘BASIC命令表……PC-8001
$N_{60}$-扩展BASIC命令表……PC-6001，PC-6001mkII
F-BASIC命令表……FM-8，FM-7，FM-11
T-BASIC7命令表……PASOPIA，PASOPIA7

## （1）N88磁盘BASIC

| 命令 | |
|---|---|
| BLOAD | BLOAD〈文件指定〉，〈装入地址〉，R<br>装入机器语言程序。指定装入地址并从该地址装入。指定R并从BSAVE执行时的存储起始地址执行 |
| BSAVE | BSAVE〈文件指定〉，〈存储起始地址〉，〈长度〉<br>存储机器语言程序。从存储起始地址开始以指定的长度存储内存中的机器语言程序 |
| FILES/LFILES | FILES〈驱动器号〉/LFILES〈驱动器号〉<br>显示已存入磁盘的文件的名字、种类以及大小。用LFILES在打印机上打印 |
| LOAD | LOAD〈文件指定〉，R<br>从文件装入程序。若指定R装入后立即执行。已经打开的文件仍保持打开状态 |
| MERGE | MERGE〈文件指定〉<br>合并程序。合并内存里的程序和文件指定的程序。文件指定的程序必须用ASCII码形式存储。相同的行号则文件指定的程序优先 |
| SAVE | SAVE〈文件指定〉，A<br>SAVE〈文件指定〉，P<br>将内存中的BASIC程序存储到文件中。指定A时用ASCII码形式存储。如指定P则是用代码化的二进制形式存储 |
| 磁盘控制命令/语句 | |
| CHAIN | CHAIN MERGE〈文件指定〉，〈行编号〉，ALL,DELETE〈范围〉<br>连接程序并执行。详见本书 |
| COMMON | COMMON〈变量名〉<br>执行CHAIN语句时传递变量。数组变量要加上“(　　)”来表示 |
| DSKO $ | DSKO$〈驱动器号〉,〈盘面号〉,〈磁道号〉,〈扇区号〉<br>将0号缓冲区的内容写入指定的扇区中。单面磁盘省略盘面号 |
| KILL | KILL〈文件指定〉<br>从磁盘中删除文件 |
| NAME | NAME〈旧文件指定〉AS〈新文件指定〉<br>更改文件名 |
| SET | SET〈驱动器号〉,〈属性字〉<br>SET〈文件指定〉,〈属性字〉<br>SET#〈文件编号〉,〈属性字〉<br>文件属性的设定和解除。属性字用P表示写保护，用R表示写检验 |
| 磁盘控制命令/函数 | |
| ATTR $ | ATTR$(〈驱动器号〉)<br>ATTR$(〈文件指定〉)<br>ATTR$(#〈文件编号〉)<br>获得文件的属性。用“　　”表示没有属性，用“P”表示写保护，“R”表示写检验 |
| DSKF | DSKF(〈驱动器号〉,〈功能号〉)<br>获得有关磁盘的情报。如省略功能号可获得空余的盘束数。有关功能号可参照本书 |
| DSKI$ | DSKI$(〈驱动器号〉,〈盘面号〉,〈磁道号〉,〈扇区号〉)<br>将指定的扇区的内容读到0号缓冲区中。单面磁盘省略盘面号 |

| 输入输出命令/语句 | |
|---|---|
| CLOSE | CLOSE#〈文件编号〉<br>关闭文件 |
| FIELD | FIELD#〈文件编号〉,〈字段宽度〉AS〈字符变量〉<br>进行缓冲区字段的定义。字段宽度的总计达256 |
| GET | GET#〈文件编号〉,〈记录编号〉<br>将文件里的数据读到缓冲区中 |
| INPUT# | INPUT#〈文件编号〉,〈输入名表〉<br>从顺序文件读入数据 |
| LINEINPUT | LINE INPUT#〈文件编号〉,〈字符变量〉<br>将整个1行不加分隔地由顺序文件读入字符变量 |
| LSET/RSET | LSET〈字符变量〉=〈字符串〉/RSET〈字符变量〉=〈字符串〉<br>将数据置入缓冲区字段。字符串比字段长度短时，LSET左起置入数据，RSET右起置入数据。字符串比字段长度长时，LSET,RSET都失去右侧部分 |
| OPEN | OPEN〈文件指定〉FOR〈形式〉AS#〈文件编号〉<br>打开文件。关于形式可参照本书。如省略FOR〈形式〉则为打开随机文件 |
| PRINT # | PRINT#〈文件编号〉,〈输出名表〉<br>向顺序文件输出数据 |
| PRINT#USING | PRINT#〈文件编号〉，USING〈写方式控制字符串〉；〈输出名表〉<br>用指定的写方式向顺序文件输出数据。写方式控制字符串与PRINT USING字符相同 |
| PUT | PUT#〈文件编号〉，〈记录编号〉<br>将缓冲区内的数据写入文件 |
| (WRITE#) | WRITE#〈文件编号〉,〈输出名表〉<br>将数据写入顺序文件。虽然WRITE#语句和PRINT语句一样将数据写入文件，但该语句能自动地删除多余的空格，并在数据之间写入逗号。另外，字符串用双引号括起来输入 |

| 输入输出命令/函数 | |
|---|---|
| EOF | EOF(〈文件编号〉)<br>检查文件是否结束。若文件结束为真，否则为假 |
| FPOS | FPOS(〈文件编号〉)<br>获得目前的连续扇区号 |
| INPUT$ | INPUT$(〈字符数〉，#〈文件编号〉)<br>从文件读取任意字符数的字符 |
| LOC | LOC(〈文件编号〉)<br>获得文件中逻辑上的目前位置。顺序文件时获得到目前为止已读写过的记录数。随机文件时获得目前的记录号 |
| LOF | LOF(〈文件编号〉)<br>获得文件的大小 |

续表

| 输入输出命令/函数 | |
|---|---|
| CUI/CVS/CVD | CVI(〈2字节的字符串〉)<br>CVS(〈4字节的字符串〉)<br>CVD(〈8字节的字符串〉)<br>将从随机文件读入的字符数据转换为数值数据 |
| MKI$/MKS$/MKD$ | MKI$(〈整数表示符号〉)<br>MKS$(〈单精度表示符号〉)<br>MKD$(〈双精度表示符号〉)<br>将各数值转换为与内部表现相对应的字符码 |

## (2) $N_{80}$磁盘BASIC

| 命令 | |
|---|---|
| BLOAD | CMD BLOAD〈文件指定〉,〈装入地址〉,R<br>装入机器语言程序。指定装入地址并从该地址进行装入。指定R并从执行CMD BSAVE时的存储起始地址开始执行 |
| BSAVE | CMD BSAVE〈文件指定〉,〈存储起始地址〉,〈长度〉<br>存储机器语言程序。从存储的起始地址开始以指定的长度来存储内存中的机器语言程序 |
| FILES/LFILES | FILES〈驱动器号〉/LFILES〈驱动器号〉 |
| LOAD | LOAD〈文件指定〉,R |
| MERGE | MERGE〈文件指定〉 |
| SAVE | SAVE〈文件指定〉,A<br>将内存中的BASIC程序存储到文件中。指定A时用ASCII码形式存储 |
| (FORMAT) | FORMAT〈驱动器号〉<br>将磁盘进行物理格式化 |

| 磁盘控制命令/语句 | |
|---|---|
| CHAIN | 无 |
| COMMON | 无 |
| DSKO$ | DSKO$〈驱动器号〉,〈盘面号〉,〈磁道号〉,〈扇区号〉 |
| KILL | KILL〈文件指定〉 |
| NAME | NAME〈旧文件指定〉AS〈新文件指定〉 |
| SET | SET〈驱动器号〉,〈属性字〉<br>SET〈文件指定〉,〈属性字〉<br>SET#〈文件编号〉,〈属性字〉 |

| 磁盘控制命令/函数 | |
|---|---|
| ATTR$ | ATTR$(〈驱动器编号〉)<br>ATTR$(〈文件指定〉)<br>ATTR$(#〈文件编号〉) |
| DSKF | DSKF(〈驱动器号〉,〈功能号〉) |
| DSKI$ | DSKI$(〈驱动器号〉,〈盘面号〉,〈磁道号〉,〈扇区号〉) |

| 输入输出命令/语句 | |
|---|---|
| CLOSE | CLOSE #〈文件编号〉 |
| FIELD | FIELD#〈文件编号〉,〈字段长度〉AS〈字符变量〉 |
| GET | GET#〈文件编号〉,〈记录号〉 |
| INPUT# | INPUT#〈文件编号〉,〈输入名表〉 |
| LINE INPUT# | LINE INPUT#〈文件编号〉,〈字符变量〉 |
| LSET/RSET | LSET〈字符变量〉=〈字符串〉/RSET〈字符变量〉=字符串 |
| OPEN | OPEN〈文件指定〉FOR〈方式〉AS#〈文件编号〉 |
| PRINT# | PRINT#〈文件编号〉,〈输出名表〉 |
| PRINT#,USING | PRINT#〈文件编号〉,USING〈写方式控制字符串〉;〈输出名表〉 |
| PUT | PUT#〈文件编号〉,〈记录号〉 |

| 输入输出命令/函数 | |
|---|---|
| EOF | EOF(〈文件编号〉) |
| FPOS | FPOS(〈文件编号〉) |
| INPUT$ | INPUT$(〈字符数〉,#〈文件编号〉) |
| LOC | LOC(〈文件编号〉) |
| LOF | LOF(〈文件编号〉) |
| CVI/CVS/CVD | CVI(〈2字节的字符串〉)<br>CVS(〈4字节的字符串〉)<br>CVD(〈8字节的字符串〉) |
| MKI$/MKS$/MKD$ | MKI$(〈整数表示符号〉)<br>MKS$(〈单精度表示符号〉)<br>MKD$(〈双精度表示符号〉) |

## (3) N磁盘BASIC

| 命令 | |
|---|---|
| BLOAD | 无 |
| BSAVE | 无 |
| FILES/LFILES | FILES〈驱动器号〉/LFILES〈驱动器号〉 |
| LOAD | LOAD〈文件指定〉,R |
| MERGE | MERGE〈文件指定〉 |
| SAVE | SAVE〈文件指定〉,A<br>将内存里的BASIC程序存储到文件中,如指定A时则用ASCII码形式存储 |
| (MOUNT) | MOUNT〈驱动器号〉<br>能够进行读磁盘的FAT以后的磁盘输入输出 |

续表

| 命令 | |
|---|---|
| (REMOVE) | REMDVE〈驱动区号〉<br>写入磁盘的FAT后结束磁盘的输入输出 |
| (FORMAT) | FORMAT〈驱动器号〉<br>进行磁盘的物理格式化 |

| 磁盘控制命令/语句 | |
|---|---|
| CHAIN | 无 |
| COMMON | 无 |
| DSKO$ | DSKO$〈驱动区号〉，〈盘面号〉，〈磁道号〉，〈扇区号〉 |
| KILL | KILL〈文件指定〉 |
| NAME | NAME〈旧文件指定〉AS〈新文件指定〉 |
| SET | SET〈驱动器号〉，〈属性字〉<br>SET〈文件指定〉，〈属性字〉<br>SET#〈文件编号〉，〈属性字〉 |

| 磁盘控制命令/函数 | |
|---|---|
| ATTR$ | ATTR$(〈驱动器号〉)<br>ATTR$(〈文件指定〉)<br>ATTR$(#〈文件编号〉) |
| DSKF | DSKF(〈驱动区号〉)<br>获得磁盘的空余盘束数 |
| DSKI$ | DSKI$(〈驱动器号〉，〈盘面号〉，〈磁道号〉，〈扇区号〉 |

| 输入输出命令/语句 | |
|---|---|
| CLOSE | CLOSE#〈文件编号〉 |
| FIELD | FIELD#〈文件编号〉，〈字段宽度〉AS〈字符变量〉 |
| GET | GET#〈文件编号〉，〈记录号〉 |
| INPUT# | INPUT#〈文件编号〉，〈输入名表〉 |
| LINE INPUT# | LINT INPUT#〈文件编号〉，〈字符变量〉 |
| LSET/RSET | LSET〈字符变量〉=〈字符串〉/RSET〈字符变量〉=〈字符串〉 |
| OPEN | OPEN〈文件指定〉FOR〈方式〉AS#〈文件编号〉 |
| PRINT# | PRINT#〈文件编号〉，〈输出名表〉 |
| PRINT#USING | PRINT#〈文件编号〉，USING〈写方式控制字符串〉；〈输出名表〉 |
| PUT | PUT#〈文件编号〉，〈记录号〉 |

| 输入输出命令/函数 | |
|---|---|
| EOF | EOF(〈文件编号〉) |
| FPOS | FPOS(〈文件编号〉) |

续表

| 输入输出命令/函数 | |
|---|---|
| INPUT$ | INPUT$(〈字符数〉,#〈文件编号〉) |
| LOC | LOC(〈文件编号〉) |
| LOF | LOF(〈文件编号〉) |
| CUI/CVS/CVD | CVI(〈2字节的字符串〉)<br>CVS(〈4字节的字符串〉)<br>CVD(〈8字节的字符串〉) |
| MKI$/MKS$/MKD$ | MKI$(〈整数表示符号〉)<br>MKS$(〈单精度表示符号〉)<br>MKD$(〈双精度表示符号〉) |

## (4) $N_{60}$-扩展BASIC

| 命令 | |
|---|---|
| BLOAD | BLOAD〈文件指定〉,〈装入地址〉,R |
| BSAVE | BSAVE〈文件指定〉,〈存储起始地址〉,〈长度〉 |
| FILES/LFILES | FILES〈驱动器号〉/LFILES〈驱动器号〉 |
| LOAD | LOAD〈文件指定〉,R |
| MERGE | MERGE〈文件指定〉 |
| SAVE | SAVE〈文件指定〉,A<br>将内存中的BASIC程序存储到文件中。若指定A时则用ASCII码形式存储 |

| 磁盘控制命令/语句 | |
|---|---|
| CHAIN | 无 |
| COMMON | 无 |
| DSKO$ | DSKO$〈驱动器号〉,〈磁道号〉,〈扇区号〉<br>在指定的扇区内写入缓冲器的内容 |
| KILL | RILL〈文件指定〉 |
| NAME | NAME〈旧文件指定〉AS〈新文件指定〉 |
| SET | 无 |

| 磁盘控制命令/函数 | |
|---|---|
| ATTR$ | 无 |
| DSKF | DSKF(〈驱动器号〉)<br>获得磁盘的空余盘束数 |
| DSKI$ | DSKI$(〈驱动器号〉,〈磁道号〉,〈扇区数〉)<br>将指定扇区的内容读入0号缓冲区 |

续表

| 输入输出命令/语句 | |
|---|---|
| CLOSE | CLOSE#〈文件编号〉 |
| FIELD | FIELD#〈文件编号〉，〈字段宽度〉AS〈字符变量〉 |
| GET | GET#〈文件编号〉,〈记录号〉 |
| INPUT# | INPUT#〈文件编号〉,〈输入名表〉 |
| LINE INPUT# | 无 |
| LSET/RSET | LSET〈字符变量〉=〈字符串〉/RSET〈字符变量〉=〈字符串〉 |
| OPEN | OPEN〈文件指定〉FOR〈方式〉AS#〈文件编号〉 |
| PRINT# | PRINT#〈文件编号〉，〈输出名表〉 |
| PRINT#,OSING | 无 |
| PUT | PUT#〈文件编号〉，〈记录号〉 |

| 输入输出命令/函数 | |
|---|---|
| EOF | EOF(〈文件编号〉) |
| FPOS | 无 |
| INPUT$ | 无 |
| LOC | LOC(〈文件编号〉) |
| LOF | LOF(〈文件编号〉) |
| CVI/CVS/CVD | CVS(〈4字节的字符串〉<br>将从随机文件读入的字符数据转换为数值数据。CVI,CVD 无 |
| MKI$/MKS$/MKD$ | MKS$(〈单精度表示符号〉)<br>将数值转换为与内部表现相对应的字符码。MKI$,MKD$无 |

## （5）F-BASIC

| 命令 | |
|---|---|
| BLOAD | LOADM〈文件指定〉，〈偏移量〉，R<br>装入机器语言的程序。若指定偏移量时，则从SAVEM执行时的存储起始地址加上偏移量后的地址装入。若指定R时则从SAVEM的执行起始地址开始执行 |
| BSAVE | SAVEM〈文件指定〉,〈存储起始地址〉，〈存储结束地址〉，〈执行起始地址〉<br>存储机器语言程序。存储内存里的机器语言程序，从存储起始地址到存储结束地址 |
| FILES/LFILES | FILES"〈驱动器号〉:",L<br>显示已存入磁盘的文件名字、种类、形式、编制、大小。若指定L时也在打印机上打印 |
| LOAD | LOAD〈文件指定〉，R |
| MERGE | MERGE〈文件指定〉，R<br>合并程序。合并内存里的程序和文件指定的程序。文件指定的程序必须用ASCII码形式存储。相同的行号，则文件指定的程序优先。若指定R，则从合并后的程序开头开始执行 |
| SAVE | SAVE〈文件指定〉,A |

续表

| 命令 | |
|---|---|
| (DSKINI) | DSKINI〈驱动器号〉<br>进行磁盘目录的初始化 |

磁盘控制命令/语句

| | |
|---|---|
| CHAIN | CHAIN MERGE〈文件指定〉,〈行编号〉,ALL,DELETE〈范围〉 |
| COMMON | COMMON〈变量名〉 |
| DSKO$ | DSKO$〈驱动区号〉,〈磁道号〉,〈扇区号〉<br>在指定的扇区内写入0号缓冲区的内容。小型磁盘扇区号1-16表示为正面，17～32则表示为反面，标准磁盘扇区号1～26表示为正面，27～52则表示为反面 |
| KILL | KILL 〈文件指定〉 |
| NAME | NAME〈旧文件指定〉AS〈新文件指定〉 |
| SET | 无 |

磁盘控制命令/函数

| | |
|---|---|
| ATTR$ | 无 |
| DSKF | DSKF(〈驱动器号〉)<br>获得磁盘的空余盘束数 |
| DSKI$ | DSKI$(〈驱动器号〉,〈磁道号〉,〈扇区号〉)<br>将指定的扇区内容读入0号缓冲区内。磁道号的指定与DSKO$相同 |

输入输出命令/语句

| | |
|---|---|
| CLOSE | CLOSE#〈文件编号〉 |
| FIELD | FIELD#〈文件编号〉,〈字段宽〉AS〈字符变量〉 |
| GET | GET#〈文件编号〉,〈记录号〉 |
| INPUT# | INPUT#〈文件编号〉,〈输入名表〉 |
| LINE INPUT# | LINE INPUT#〈文件编号〉,〈字符变量〉<br>将整个一行不加分隔地由顺序文件读入字符变量 |
| LSET/RSET | LSET〈字符变量〉=〈字符串〉/RSET〈字符变量〉=〈字符串〉 |
| OPEN | OPEN"〈方式〉",#〈文件编号〉,〈文件指定〉<br>打开文件。方式用I为顺序文件的INPUT方式，用O为顺序文件的OUTPUT方式，用A为顺序文件的APPEND方式，用R则打开随机文件 |
| PRINT# | PRINT#〈文件编号〉,〈输出名表〉 |
| PRINT#,USING | PRINT#〈文件编号〉,USING〈写方式控制字符串〉;〈输出名表〉 |
| PUT | PUT#〈文件编号〉,〈记录编号〉 |
| (WRITE#) | WRITE#〈文件编号〉,〈输出名表〉<br>将数据写入顺序文件。虽然WRITE#语句和PRINT语句一样将数据写入文件，但该语句能自动地删除多余的空白，并在数据之间写入逗号。另外，字符串用双引号括起来输入，但用Ver4.0作支持 |

续表

| 输入输出命令/函数 | |
|---|---|
| EOF | EOF(〈文件编号〉) |
| FPOS | 无 |
| INPUT$ | INPUT$(〈字符数〉#〈文件编号〉) |
| LOC | LOC(〈文件编号〉)<br>对于随机文件获得下一个读写的记录号 |
| LOF | LOF(〈文件编号〉)<br>获得随机文件的最大记录号 |
| CVI/CVS/CVD | CVI(〈2字节的字符串〉)<br>CVS(〈4字节的字符串〉)<br>CVD(〈8字节的字符串〉) |
| MKI$/MKS$/MKD$ | MKI$(〈整数表示符号〉)<br>MKS$(〈单精度表示符号〉)<br>MKD$(〈双精度表示符号〉) |

## (6) T-BASIO7

| 命令 | |
|---|---|
| BLOAD | BLOAD〈文件指定〉,〈装入地址〉<br>装入机器语言程序。指定装入地址并从该地址装入 |
| BSAVE | BSAVE〈文件指定〉,〈存储起始地址〉,〈长度〉 |
| FILES/LFILES | EILES〈驱动器号〉/LFILES〈驱动器号〉 |
| LOAD | LOAD〈文件指定〉,R |
| MERGE | MERGE〈文件指定〉 |
| SAVE | SAVE〈文件指定〉,A<br>SAVE〈文件指定〉,P |

| 磁盘控制命令/语句 | |
|---|---|
| CHAIN | CHAIN MERGE〈文件指定〉,〈行编号〉, ALL, DELETE〈范围〉 |
| COMMON | COMMON〈变量名〉 |
| DSKO$ | DSKO$〈驱动器号〉,〈盘面号〉,〈磁道号〉,〈扇区号〉 |
| KILL | KILL〈文件指定〉 |
| NAME | NAME〈旧文件指定〉AS〈新文件指定〉 |
| SET | SET〈驱动器号〉,〈属性字〉<br>SET〈文件指定〉,〈属性字〉<br>SET#〈文件编号〉,〈属性字〉 |

续表

| 磁盘控制命令/函数 | |
|---|---|
| ATTR$ | ATTR(〈驱动器号〉)<br>ATTR$(〈文件指定〉)<br>ATTR$(#〈文件编号〉) |
| DSKF | DSKF(〈驱动器号〉)<br>获得磁盘的空余盘束数 |
| DSKI$ | DSKI$(〈驱动器号〉, 〈盘面号〉, 〈磁道号〉, 〈扇区号〉 |

| 输入输出命令/语句 | |
|---|---|
| CLOSE | CLOSE#〈文件编号〉 |
| FIELD | FIELD#〈文件编号〉, 〈字段宽〉AS〈字符变量〉 |
| GET | GET#〈文件编号〉, 〈记录号〉 |
| INPUT# | INPUT#〈文件编号〉, 〈输入名表〉 |
| LINE INPUT# | LINE INPUT#〈文件编号〉, 〈字符变量〉 |
| LSET/RSET | LSET〈字符变量〉=〈字符串〉/RSET〈字符变量〉=〈字符串〉 |
| OPEN | OPEN〈文件指定〉FOR〈方式〉AS#〈文件编号〉 |
| PRINT# | PRINT#〈文件编号〉, 〈输出名表〉 |
| PRINT#, USING | PRINT#〈文件编号〉, USING〈写方式控制字符串〉; 〈输出名表〉 |
| PUT | PUT#〈文件编号〉, 〈记录号〉 |
| (WRITE#) | WRITE#〈文件编号〉, 〈输出名表〉<br>将数据写入顺序文件。虽然WRITE#语句和PRINT语句一样将数据写入文件，但该语句能自动地删除多余的空格，并在数据之间写入逗号。另外，字符串用双引号括起来输入 |

| 输入输出命令/函数 | |
|---|---|
| EOF | EOF(〈文件编号〉) |
| FPOS | FPOS(〈文件编号〉 |
| INPUT$ | INPUT$(〈字符数〉,#〈文件编号〉) |
| LOC | LOC(〈文件编号〉) |
| LOF | LOF(〈文件编号〉) |
| CVI/CVS/CVD | CVI(〈2字节的字符串〉)<br>CVS(〈4字节的字符串〉)<br>CVD(〈8字节的字符串〉) |
| MKI$/MKS$/MKD$ | MKI$(〈整数表示符号〉)<br>MKS$(〈单精度表示符号〉)<br>MKD$(〈双精度表示符号〉) |

# 索　引