IBM PC

个人计算机的使用

——最新的 XT 和 DOS 2.0——

[美] T.G.刘易斯 著

新书预订

· 用你的个人计算机做一千零一件事

（1985年3月出版） ¥3.80元

· Prolog 语言程序设计

（1985年3月出版） ¥3.50元

· 建立数学模型实例研究

（1985年5月出版） ¥3.50元

请订购者通过邮局汇款至

武汉市813信箱

3.60

IBM个人计算机的使用

Using the IBM Personal Computer

liu yi si
[美] T.G.刘易斯 著

袁由光 吴泰煦 等译校

中国南方电脑工程公司

1984

IBM个人计算机的使用

Using the IBM Personal Computer

[美] T.G.刘易斯 著

袁由光 吴泰煦 等译校

中国南方电脑工程公司

1984

译　者　序

IBM PC是国际商用机器(IBM公司)研制的个人计算机。由于其功能强、兼容性好、软件丰富，因而在国际上享有盛誉，处于领先地位。该产品市场日益扩大，用户日益增多。目前，IBM PC在我国刚刚开始引起人们的注意，大家都迫切要求了解和掌握其使用方法。正是在这样一种形势下，为满足国内四化建设的需要，我们谨向读者推荐本书。

本书是一本掌握和使用IBM PC的通俗读物。作者Lewis考虑到初学者的利益，把本书的重点放在“怎样使用”上。它没有象一般计算机书籍那样对基本原理作冗长的叙述，只是对有关的术语、概念作了简明扼要的说明。书中文字浅显易懂，内容简炼，配有大量实例及屏幕显示图，使读者对计算机的操作过程能有一个较直观的理解。

本书对想购买或手头已有IBM PC、Easywriter、Visicalc或Pascal系统的人特别适用。

本书由中国船舶工业总公司第七〇九研究所的部份专业技术人员及专职翻译共同翻译。参加本书翻译工作的有王仁华、刘胜厚、吴泰煦、张鸿海、付扬、贺良材、贾建平、马中、王勤、柳浩然、夏以信、李开轩等同志，参加校订工作的有刘日升、袁由光、贺良材、吴泰煦、陈护勋、余伟新、沈占鳌、李山林、夏以信、贾建平等同志、最后由袁由光、吴泰煦同志负责总校。

由于译者水平有限，书中可能有不少错误和不当之处，敬请读者批评指正。

目　录

序　言

本书是为那些想买或最近已买了IBM个人计算机、Easy Writer、VisiCalc或Pascal系统的人撰写的。如果你有了一台带Easy Writer或VisiCalc系统的微计算机，你仍然会对本书感兴趣，因为这些程序现在正在许多的微计算机上运行。

本书的主要目的是为了告诉读者：微计算机能做些什么工作？计算机是怎样操作的？文件（file）、RAM、位（bit）、BASIC、数据库（database），电子工作单（electronic worksheet）字处理，操作系统和软件这些名词的含义是什么？本书将对这些名词和其他一些名词作一简洁的解释。

更确切地说，本书的目的是要告诉读者怎样使用最流行的MDOS操作系统、EasyWriter字处理程序、称做VisiCalc的展开单（spreadsheet）计算程序以及两种程序设计语言；MBASIC和Pascal。本书也为那些已拥有或计划购买IBM个人计算机的人描述了这种功能很强的机器。

学习计算机的最好途径是实际使用一下。

因此，本书是从用户的角度来写的，也就是说，我是从如何使用机器这个角度来阐明每一问题的。事实上，许多例子都是直接从计算机显示屏幕翻拍下来的，这样你能确切地看到计算机在做什么。在有些例子中，照片与你从自己的计算机屏幕上看到的有所不同，这是因为计算机制造厂家可能

已修改了系统以容纳其他程序。但是在大多数地方，我将尽力使例子接近实际情况。

在第一章，我研究了计算机可以给你个人带来好处的几个领域。它们是：（1）字处理，（2）展开单计算，（3）一般问题解算，（4）数据库处理。这几个方面将在下面章节中进行深入的讨论。

在第二章，我简单介绍了计算机界使用的术语。当你遇见程序员、推销员或其他计算机拥有者时，有关“计算机术语”的介绍将使你感到方便。这也将有助于你理解下面的章节。如果要详细了解计算机是怎样工作的，请见第八章。

第三章开始接触正题，解释了怎样使用随你的计算机一道提供的MDOS 操作系统程序。这里重点放在“怎样使用”上，而不是怎样修改、增加或改造整个操作系统。如果你想对 MDOS 进行一些改造，那就需要再阅读一些关于MDOS的书。

第四章是关于字处理的。Easy Writer是一种程序，它能把你的计算机变成一架高级的电子打字机。但是有了Easy Writer程序以后，你还能做一些打字机所不能做的事情。这种令人振奋的程序是对人类的一大贡献，但要花很大的努力才能学会它。这也是第四章的要点之所在。学完第四章后，你应该成为中等水平的字处理专家。

也许，最近对计算影响最大的冲击要算为微计算机设计的许多展开单计算程序。这些程序把计算机变成了一种电子工作单。第五章说明了怎样使用VisiCalc来做大量的直观计算。这种程序是如此有趣，以至于你将舍不得往下看下一章。

第六章涉及程序设计。字处理程序和展开单计算程序不足以解决的一般问题将从本章中找到答案。本章讨论了两种语言：MBASIC和Pascal。前者是一种解释程序，后者是一种编译语言。它们的差别在哪里呢？看完本章便知。

第七章涉及的问题是小型计算机领域中的重大问题。数据库管理系统能降低日益上升的软件价格，是非常重要的程序。我们周围有许多数据库管理程序。哪一种最好？在本章中，我们将考察一下种种不同类型的数据库模型，然后学习使用一种简单的关系模型数据库系统。为了把事情做得更细一些，我们将研究一种关系数据库程序，该程序在随机器一起提供给你的BASIC中。你可以把这种程序输进计算机运行。在购买更昂贵的数据库系统前，它将给你一个机会试用一下这种非常简单的数据库系统。

第八章简单评述了计算机是如何工作的。它研究了计算机的内部工作情况。如果仅仅是使用计算机，那么你就不必了解本章中涉及的课题。然而，如果你有兴趣的话，也不妨一读。

T·G·**刘易斯**

第 一 章

计算机能做些什么？

也许，人们应该被这个武器，这个祸根．这个可怕的破坏性机器所震惊。但是，他们因为太忙而没有注意到。

也许，人们应该被这个工具．这个给人类赐福的强大的和平工具所震惊。但是，他们因为太忙而没有注意到。

直到人们不得不注意到这个威力无比的机器—计算机。

信 息 工 具

微计算机作为一种信息工具给人们处理信息的手段带来了革命性的变革。正象一世纪前蒸汽机的发明增强了人类肌肉的力量一样，计算机增强了人类的智力。一个证券经纪人借助微计算机这个工具可以记录委托人的帐单和某几种股票，能够进行以前从不可能做到的整个股票行情分析。若不用计算机的话，一个建筑师或施工员在估算建筑费用时要花很长时间，约为完成任务所需时间的十分之一。一个电气承包商在微计算机上不仅能进行通常的商业性数据处理，而且还能通过分析出价形势来操纵竞争。一旦知道某一个竞争者的出价，聪明的承包商总能够以低5美元的喊价来赢得合同。

使用微计算机的理由因人而异。下面是商业上使用微计算机的一些理由：

- 提高精确度，
- 提高生产率，
- 减少涌塞或混乱现象，
- 改善现金流通或税金情况，
- 或者简单地说改善你的状况。

如果用户能恰当选择、安装和使用计算机，那么、它就可以承担所有这些事情。然而，如果用得不恰当，计算机也可能带来不幸。在购买和使用计算机时避免出毛病的最好的方法是事先成为熟悉的用户。这就是本书的目的。

精确度

也许，关于计算机最闻名的是它们每秒钟可执行成百万次算术运算而不出错的能力。我们最喜爱的计算机的性能之一是它们的可靠性。例如，当人们打入成千上万的十进制数字时，微计算机就可在里面完成相加，求平均数。

典型的商业应用要求“元和分”的计算精度。为达此目的，在每次中间计算中有必要进行具有“元和分”精度的运算。例如，10.95美元变成10.950美元，这样在计算过程中，一个千分之一美元的零头就算进去了。即使这样谨慎的话，如本章以后所讨论的那样也有可能损失一些精度。

生产率

计算机制造商最近宣称计算机能提高工人的生产率。这种想法直接和计算机作为一种信息处理工具的概念有关。然而，我们必须小心，不要过多地渲染计算机的能力。例如，一种通常的谬误就是计算机取代了人的工作。这不是事实。然而，计算机可以干某些原来人干的工作。让我们举例说明。

约翰和玛利在一个大城市的郊区经营一个蔬菜农场。他们的农场生产各种蔬菜，约翰每天把它们运到城里去。

一天，约翰决定买一台微计算机来处理农场的帐目。但是，玛利有些犹豫，因为她意识到将需要人来操作计算机和输入数据，等等。根据玛利的想法，他们将要雇用一人来管理计算机，其结果是降低了整个生产率。但是，约翰讲计算机由于其速度和精度将提高生产率。谁对呢？

实际上，约翰和玛利两人的判断都只对了一部分。没有计算机，约翰和玛利要花费很多时间来准备他们的销售单、收据和日记帐，再把这些交给他们的会计去处理。会计反过来又要为政府、约翰和玛利把这些数据处理成他们各自所需的形式。会计的工资是一种流通费用，约翰和玛利每年（或每月）付他一次。

有了一台计算机，约翰和玛利就能取代一部分原属于会计的工作以及他们自己的部分工作。这些工作交给了效率更高的计算机和计算机操作员。这样，总的生产效率提高了，因为计算机代替了一名会计、约翰和玛利的工作。

在其寿命周期里，计算机的一次性的购买费是分期偿还的。一旦付清后，就不用付了（假定它不损坏）。

这个关于约翰和玛利的故事说明了计算机在商业界应用的情况。这就创立了一种新的业务，因为它们比所取代的旧的工作方式具有更高的生产率；旧的工作方式消亡了，因为它们缺乏竞争力。

涌塞和混乱

计算机常用来消除过分的延迟或改进信息流通工作。即

使小的商业部门也想在发薪前一天把所有雇员的工资单打印出来。律师事务所的接待员想检索出三个星期前已输入字处理计算机的一封信。计算机可以很快找出这封信，比人工在文件柜内查找要快。

现金流通

计算机是一种信息处理工具，它也可改变一个小企业的现金流通。让我们看一个这种辅助作用的例子。

假设约翰和玛利花10000美元买了一个计算机系统。这种购买可通过借贷来支付，以后再每月偿还。假设在5年以后付清欠款，因而约翰和玛利也就能“自由”地使用计算机。即，计算机干的工作在付清购买款以前只是一种“价值有限的工作”（marginal labor），然后在付清购买款以后还可使用许多年。

用原来支付给会计的钱来购买一台计算机，从现金流通方面来看还能从国内税务局得到进一步的好处。如果我们假设折旧率为五年，那么从约翰和玛利应纳税的收入中每年要减去2000美元。如果减税属于30%的一档，那么美国政府5年中每年就要为约翰和玛利的计算机付2000(美元)×30%＝600美元。总而言之，政府就要为约翰和玛利的10000美元的计算机支付3000美元。

另外，政府对于购买新设备经常给予高达10%的补贴鼓励。这意味着约翰和玛利的计算机在第一年还可获得1000美元的补贴。这样购买计算机的费用就可下降到6000美元。

下面我们假设约翰和玛利的会计每小时要60美元的薄记费。没有计算机的话，这一项一年总共要1200美元；但是有

了一台计算机，每年会计的工资就降到300美元。因此，买了计算机以后每年可节省900美元。5年中，约翰和玛利为计算机支付的整个费用实际已降到1500美元，即每年300美元。

如果约翰和玛利把他们的计算机保留10年，单这项收入有限的工作，就可为他们节省3000美元。这还没有包括购买计算机所带来的方便性、速度、准确性等方面的提高，也不包括最初采购时借款的利息。

地位

最后，许多人购买计算机纯粹作为自己地位的标志。计算机也是一种力量的标志。如果你用微计算机当场进行计算的话，究竟还有谁会提出疑问？

不止一个贷款员受到顾客的责骂，因为他们已经确切知道买一辆新车每月需付多少钱。另外，顾客可能也提前知道整个利息是多少，多少利息将从减税中得到补偿。

目前计算机不能解决的问题

计算机能思考吗？这是一个引起争论的问题，它刺激了心理学、医学、哲学以及计算机科学方面的研究工作。问题的答案是我们至今还不知道什么是思维或它在人脑中的工作过程。

完全有可能在某一天设计出至少能模拟人类思维的计算机。如能成功，全世界都将承认发明者是最聪明的人。然而也可能思维完全是人类活动的结果，因为有些问题从逻辑上讲

机器是根本不能解答的。下面给出一个简单的不可解答的问题，它能说明模拟人类思维中遇到的问题。

一个难解的问题

考虑下面这个不可解的问题：

“弗雷德常讲真话，”约翰讲。

“约翰撒谎，”弗雷德说。

在这个问题中，我们要根据上面的二项声明，判明是弗雷德还是约翰讲了真话。我们假定约翰撒了谎。这个假定在逻辑上就不能成立。假如约翰撒了谎，那么意味着弗雷德也说了谎。假如弗雷德撒了谎，那就意味着约翰没有撒谎，那就和原来的假定相矛盾。如果我们的假定相反，即假如约翰讲了真话，那么弗雷德讲的是真话。但是，假如弗雷德讲了真话，那么约翰撒了谎，这样和事实相矛盾。总之，此问题是不可解的。

这样，建造一台可思维的计算机究竟是可能的，还是象上面的例子那样，是不可能的？没有人知道答案，但我们知道，有一些事情人可以做得比机器更好。下面是一些人比机器做得好的事情。

校正功能

计算机能够校正在操作过程中可能发生的小的错误。例如，计算机能判明输入的一个数对于机器的操作来说是太大或太小了。然而，有时计算机本身不能判断它给出的答案是否“正确”。

只有计算机操作员才能判断计算机提供的回答是否正

确。例如一台计算机将错误地把1000美元的支票打印成1000000美元。然而，一个熟练的使用者能立即发现这类错误，因为1行6个零总能引起人们的注意。

输入错误

数据处理中最常见的错误类型是不正确的输入。只有计算机操作员才能判明一个输入值是否正确。

在输入过程中，计算机很容易检测到用户输入中的名字错误，不是数字错误。计算机不能检测出“元和分”输入中的错误，例如把14.55美元错成15.95美元。

检查输入错误的方法之一是使用校验和(Checksums)。在数据输入过程中可以计算出校验和，然后与以前算出的校验和进行比较。如果两个和不相符，操作员就知道出现了错误。这种检测错误的方法非常有用，但请注意这取决于人工干预。

理解能力

计算机能够模拟某些智能过程，因此我们也可以说这种模拟类似人类的思维，但是当今没有计算机能理解人类的想法。例如，计算机可用英语进行通信，但是计算机不知道英语单词的含义。

我们不能用日常英语叫计算机做事(至少目前不行)。相反，我们必须使用象BASIC或Pascal这样非常严密的语言作为中间语言来和机器通信。这些程序设计语言更象数学而不象英语。另外，它们不直接控制计算机，而必须首先转换成电信号。我们将在下一章讨论其转换的过程。

使用计算机中的问题是计算机不能理解自然语言。这意味着程序员必须把英语写的指令翻译成计算机语言，这样命令能依次再转换成电信号。这就是计算机程序设计的过程，这也是软件的来源(控制计算机的程序)。

难解的问题

有些问题的解算非常费时间，甚至非常高速的计算机也解算不出来。例如，有些密码非常复杂，一台计算机可能要花20年才能找出其中所有的组合。

在一盘国际象棋中棋子可以移动的地方非常多，以至于一台非常高速的计算机在十亿年中也算不出所有走子的方案。这类问题似乎是人为设计出来的，但是在许多现实的应用中，我们恰恰面临着类似的困难。

交通部门面临这类难解问题的一个例子是最佳调度问题。假设一家公司有几个仓库和遍布全国各地的许多商品零售批发点。另外假设公司想通过每次卡车集散货物的运输距离最小化来降低从仓库向零售商店供货的费用。这个问题可以通过研究比较几千条卡车路线来解决。然而这样做，一台好的计算机也要花很长时间才能算出其结果。

难解的问题常常可通过人的直觉结合计算机处理来解决。在运输问题中，某人提出一种调度的方案，而计算机计算其费用。因为有人发现一条“更好的路线”，因此进行改进，计算机的计算也指出费用真正下降了。这种过程继续下去，直到找出一种合理的解。

计算机不能发现新方法、使用直觉、进行革新、理解自然语言、自己编程序、发现输入错误、论证输出值的正确性

或判断问题是否能解决。计算机是非常高速、精确而又勤快的，但思维则是人类独一无二的功能。

计算机做价值有限的劳动

确切地说微机能干些什么呢？自从一台微机开始做价值有限的劳动以来，从某种意义上讲它们是一些干某些事情比人更快、更精确、更方便的奴隶。我们采用价值有限的劳动这种说法的意思是计算机将不会节省劳力，因为需要有人来操作、编程序、维护它。然而，对于相同的劳动费用来说，计算机将有更高的生产率。另外，在不增加拥有者附加费用的情况下，计算机还能增加处理的功能。一旦付清了购买费用以后，计算机就开始省钱—这就是价值有限的劳动的真正含义。

我们将在下列几个广阔的应用领域中来说明“电子奴隶”的用处。

- 字处理
- 展开单计算
- 通过程序设计解决一般问题
- 数据库处理

在本书以后几章中，我们将详细介绍怎样在这些应用领域中使用你的计算机。在本章的其余部分，我们将介绍这些领域，并讨论在每个领域中怎样使用计算机。

字处理

在计算中，最明显的想法之一是用一台微计算机来取代

你的机械式打字机，从而能节约时间和劳力，因为与打字机不同，计算机具有下列功能：

1.不必重新打印就能擦除、插入、删去、改变、移动、复制文件以及在任何书写文件中查找文本的特性。

2.通过记录在软盘上面而“记住”任何书写文件，以备以后参考。

第一个优点意味着当编辑商业信件、法律合同等等的文本时，不用重新打印整个文件。一封商业信件可以输进微计算机并保存在软盘上。然后信件可以在微机显示器屏幕上显示出来，进行修改。如果任何字符、字、句子或段落需要修改，可以在不改变周围文本的情况下进行修改。一旦修改满意以后，不用人的干预，它就可以打印出一个或多个相同的复制件。

字处理最常见的应用之一是保存邮寄单。假设你要给一个组织的每个成员写一封信。你可把信输进字处理系统，在里面编辑好，然后打印出原文的许多复制件。而且，你还可用以前存在机器里面的名字和地址单，叫字处理机在每封信的信头打上不同的名字和地址。

一个字处理系统包括一台带一台或多台磁带机的微计算机、一台或多台显示屏／键盘和一台高质量打印机。通常，字处理程序是和硬件分开购买的。字处理程序有许多种，然而在本章这个标题下我将只讨论其中一种。

下面列出字处理擅长的应用领域：

·法律合同、房产经纪人表格、商业（秘书事务的）信件；

·邮寄单，照相排版；

• 新闻信札、报纸、杂志、书籍；
• 俱乐部、协会、团体、学校；
• 作者、编辑、记者；
• 电子邮件、信息中心、社团通信

实际上，一切需准备重要文本文件的人使用字处理机都能提高工作效率。

展开单计算

一台微计算机比一台高质量的计算器要好得多，因为它可以“记住”大量的信息，并能“记住”处理这些信息的指令。事实上是存储的程序使得计算机优于计算器。

计算机的程序必须有人编写和维护。这是一个艰巨的任务，除非你依靠象这里讨论的自动展开单程序之类的软件工具。

展开单程序是一种帮助用户设计和应用计算机的程序，用户不必掌握许多关于计算机程序设计的知识。这种可能性如何？

一种典型的展开单程序把屏幕划分成行和列的矩阵。例如图1—1示出的一个简单的3×3的矩阵，它包括编成1.2.3的三行和标上A.B.C的三列。见图1—1。

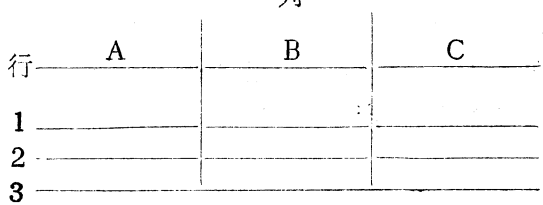


图1—1

我们标出该矩阵中标有列与行标号的每个单元。我们把左上的单元称为A_1；中间的单元称为B_2。

现在，假设我们设计了一个使用9单元矩阵的车辆运行的展开单程序。让下列单元表示已知的量：

A_1：总的汽油消耗

A_2：总的燃料费用

A_3：每加仑汽油的价格

B_1：总的行车距离

B_2：花费的总的时间

下列单元表示算出的数量：

C_1：平均里程（消耗每加仑汽油所行驶的里程）

C_2：每里的价格

C_3：平均速度

没有规定的单元B_3没有使用。

已知A_1到B_3，我们要算出C_1到C_3。由于计算机需要知道怎样算出C_1到C_3，所以我们必须提供一个公式。记住各参量间具有下列关系：

每加仑平均哩程＝总的距离除以总加仑数

每哩费用＝总费用除以总距离

平均速度＝总距离除以总时间

通过上面指定的单元标志，再把公式变成单元符号。因此，为了得到每个计算量的数字结果，我们通过计算机把存贮在一个单元里的量除以存贮在另一个单元里的量。至于C_1.C_2和C_3：

$C_1=B_1/A_1$

$C_2=A_2/B_1$

$C_3=B_1/B_2$

由上面这些公式和单元标志，我们可以得出汽车／行程展开单的结果。例如，假定我们已知（已存在矩阵里）

A_1：总加仑数＝12.8

A_2：总费用＝21.50

A_3：每加仑价格＝1.679

B_1：总距离＝512

B_2：总时间＝9.5

展开单中各个量及计算结果如下：

行 \ 列	A	B	C
1	12.8	512.0	40.0
2	21.50	9.5	0.042
3	1.679		53.9

图 1—2

这种展开单是汽车／行程系统的财务报表模型。在许多应用场合，这种模型是非常有用的。如果我们改变存储在一个单元里的其中一个值，那么，与改变了值的单元有关的其它单元的值也要改变。例如我们把A_1的值变为10.5加仑，C_1单元里的值也要变。因此，该模型可以用来研究汽车或行程系统的“如果……那么”这类问题。

在许多应用中，展开单的计算是有用的。下面我们列出其中一些最明显的应用。

·抵押、金融、实际估算

·工程、设计、施工估算
·统计、投资、科学计算
·零售、批发、仓库应用

常见问题解算

展开单计算程序简化了计算机的使用，但是对于新的或更复杂的问题，我们必须借助程序设计来解决。对于专门问题现在有两种途径来得到程序：（1）从软件卖主那里购买程序，（2）自己编写程序。

如果你选择从卖主那里购买软件，这里有三种方法可供使用：

1·购买已编好的程序，如应收帐目、工资单一般分类帐。

2.购买已编好的程序，然后根据你的要求进行修改。

3.根据你的设计要求和售主签订合同，叫他为你编写新程序。

如果你决定自己编写程序，那么，你就有必要学习更多的关于计算机方面的知识。特别是有业余爱好而自学的程序设计员必须首先学习一种正式的程序设计语言如BASIC、Pascal、FORTRAN等等，然后再学习程序设计这门技术。

计算机程序员是常见问题的解决者。程序员解决问题有其一定的规律。对于大多数程序员来说，它包括下列各点：

要求确定。对问题的解决有什么要求？例如什么是输入和输出？

设计说明。问题怎样来解决？解决的方法是什么？数据怎样组织？

编码说明。设计怎样转换成程序？使用什么样的程序设计语言？采用什么样的编码规则？

实现。用程序设计语言写的实际的程序必须是正确的和可懂的。

测试。选择的测试必须使别人确信程序是可行的。

文件编制。系统必须包括使用手册、程序清单、问题说明、程序设计、实现和测试结果。

我们可以用一个非常简单的问题来说明这些步骤。例如，已知查帐后有余额B和一张总值为A的支票，那么支付后最终的余额EB是多少呢？在这个简单的问题上，一个老练的程序员必须提防许多容易发生错误的地方。首先，问题的要求是什么呢？

要求

输入：B＝余额

A＝支票的总数

输出：EB＝最终余额

注意：要求应规定B必须总保持正，A必须大于0。

设计

方法：检查并确认B和A是正数并大于0。然后使用减法得到EB。注意：数据被编成简单的实数值数字。然而，我们必须假定它们是小到可以存放在存储器里。

编码

我们将使用BASIC作为程序设计语言以及下列文件标准：

1.所有程序将包含说明程序做什么的注释标题。

2.程序将使用“结构”控制，这样阅读程序象阅读书

本一样。

3.为变量起有意义的名字，例如，BALANCE,AMOUNT和ENDBAL。

实现

在编码过程中使用“分离层次”以提高程序设计速度和减少错误。在第六章“怎样写你自己的程序”里，我们将作更详细的说明。

本例的编码如下：

```
10  REM ----------------------------
20  REM   CHECK BOOK BALANCE PROGRAM
            INPUTS  :  BALANCE=current
                                balance
                       AMOUNT=check amount
           OUTPUTS : ENDBAL=ending balacne
60  REM ----------------------------
100  PRINT "Enter balance :" ;
110  INPUT BALANCE
120  PRINT "Enter amount :" ;
130  INPUT AMOUNT
140  IF AMOUNT <= 0 THEN 120
150  IF BALANCE - AMOUNT < = 0 THEN 200
160  ENDBAL = BALANCE-AMOUNT
170  PRINT "Ending balance $ ", ENDBAL
180  STOP
190  REM ---- ERROR CONDITION -----
```

测试

我们可以通过仔细地选择输入数据来测试本程序。数据必须用每个if Statement（条件语句）进行测试，这样每个程序语句至少执行一次。例如

AMOUNT<=0

BALANCE−AMOUNT<0

表示测试AMOUNT是否为负，BALANCE是否为负，BALANCE是否小于AMOUNT。该分析提出下列几组测试值。

1.AMOUNT=10.00
  BALANCE=50.00
2.AMOUNT=50.00
  BALANCE=10.00
3.AMOUNT=−50.00
  BALANCE=5.00
4.AMOUNT=5.00
  BALANCE=5.00
5.AMOUNT=5.00
  BALANCE=−5.00

上述这些和其他测试情况是用来检验程序是否工作。当然，这种方法并不能保证你的程序完全正确，但是这是一种保证程序在大多数时间里正常运行的好方法。

文件编制

程序为下面两组人编制成文件的形式：用户和程序员。用户只需要知道怎样使用程序，而不需了解实现的细节。

程序员必须能维护、也许要改变一条程序，因此需要了

解大量的细节。最重要的细节应该以简单的、直接的句子和图象来解释。

在下面章节中，我们将更详细地讨论程序设计。这个简明的介绍将说服你或是从一个软件售主那里购买程序或阅读下面的章节以得到更多的了解。

数据库处理

把数据表存在软盘上，这样计算机就能用来“记住”大量的信息。这些表以磁盘文件的形式来组织并存放在磁盘里（详细情况见第八章）。无论什么时候需要这些信息，只要使用一个控制键就能实现从磁盘文件里识别和检索数据。

在许多应用中，存储在磁盘文件里的表数目很大而且难以管理。由于这个原因，人们使用了一种叫做数据库管理程序的专门程序。

数据库管理系统是用软件在磁盘上输入、检索、修改及输出信息的计算机。数据库管理系 统（DBMS)主要目标之一是简化用户存取数据库信息的过程。这是怎样实现的？

实现方案：用户视图

数据库根据图解或用户视图来编排信息，从而简化了处理大量信息的工作。在DBMS 领域里，有三种基本图解(视图)：

1.网络—数据库被看作是交互式信息片的网络 。例 如美国的电话是通过电话线网络连在一起的。任何电话线可以看作和任何其他电话线相连。

2.层次结构—数据库被看作是信息片 的层次系统。例如，美国的电话可以看成是分层连结的：第一层规定为地区

码，第二层是事先规定的，等等。虽然这不是电话线的连接方法，却是用户观察系统的一种方法，即一种可接受的用户视图。

部　件

数量	部件编号	价格	日期
5	NC—875	5.95	81.9.1
3	NC—501	0.35	81.9.30
0	NC—450	1.95	81.9.9

供　应　商

姓名	街道		城市	邮区号码
Acme	Broadway	1100	Billings	87550
Jones	Olivara	3506	Los Angeles	97730

用　户

姓名	街道		城市	邮区号码	余额
Smith	Oak	3653	Tulsa	65401	100.95
Jones	Main	1335	Tewes	08350	98.50

图 1—3

3.关系—数据库被看作是表的集合。每个表是“简明”的，即每个表只是由通常的行和列构成。例如，美国的电话可以看作为出现在电话簿上。簿上每行的内容相当于一个行，每个登记项相当于一个列。因此，姓名、地址和电话号

码构成簿上的列，例如簿上的某一行中包括“Lewis”，“2812 Monterey，P_1”和“755—0853”。

关于用户视图的争论仍然在继续，但是大多数数据库系统或是层次型的，或是关系型的。作为一个关系数据的例子，我们可以研究一下怎样用关系数据来存储商品目录数据库。

三张关系表，即部件表、供应商表和用户表组成了一个目录表数据库，每张表中包括了列成表的信息（见图1－3）。

这些关系表的列叫做**域**，行叫做**元组**。“Part No.”域可以作为从PARTS里检索一个部件的控制键，“Name”可以用来从另外二张关系表中的一张检索一个元组。

我们可以想象一个处理程序能从PARTS表上取走三个NC—875部件，并把它们送给CUSTOMERS关系表中的“Smith”。CUSTOMERS的元用来查找采购的发票。

理想中的处理程序还将修改对应于这个部件编号的“Qty”域。如果“Qty”低于某一值，那么将从合适的供应商那里订更多的货物。

DBMS有许多应用。关于这种计算方法的其他情况，见专门讲数据库处理的第七章。下面是数据库处理应用的其他一些地方：

- 保存记录单、体育统计表、杂志表格。
- 商业数据处理、例如商品单、记帐。
- 能源储备情况、情报检索服务。
- 不动产一览表、人员一览表、计时服务。
- 银行帐目、火车时刻表、俱乐部、图书馆、预订、生日、事件日程表。

常见问题

问：为什么微计算机被称为信息工具？

答：因为它通过提供价值有限的劳动提高了劳动生产率。

问：如果我的个人所得税率是35%，我花5000美元买一台计算机，政府能为我支付多少？

答：假定残值为0，直线式折旧率，时间为5年，政府将支付5000美元的35%。然而，信贷、物质刺激和双重折扣，随着时间推移它们都将进一步改变这个估计。

问：一台微计算机可用多少时间？

答：如果维护和使用得当，一台微机使用时间将和一台汽车使用的时间一样长或更长。大多数情况是在一台微机“死亡”以前，你就不想再用它了，而想再买一台“更大”的机器。

问：计算机能思维吗？

答：不能。

问：计算机能解下列问题吗？

“弗雷德常撒谎”，约翰说。

“约翰常撒谎”，弗雷德说。

弗雷德和约翰在讲真话吗？

答：假设约翰讲的是真话。那么，弗雷德一定在撒谎，这样真理即是约翰在讲真话。现在，假设弗雷德讲的是真话。那么，约翰就在撒谎，于是我们提出的弗雷德在讲真话的假设成立。但是不可能两人讲的都是真话。难道是两人都在撒谎吗？

问：使用微计算机过程中最常见的错误是什么呢？

答：输入中产生的错误偶然会进入系统。

问：检查输入错误的例子或方法是什么？

答：校验和方法。

问：为什么使用程序设计语言来代替英语与计算机通信？

答：对于计算机来说，英语太含糊和不规则，难于理解。

问：什么是软件？

答：控制硬件工作的程序或指令。

问：我们所说的“难解问题”指什么？

答：难解问题是指这样一种问题，虽然它们是有解的，但即使是一台非常高速的计算机也要花很长时间才能解决。

问：什么是字处理？

答；就其最简单的方式来说，字处理就是使用计算机来代替打字机。更复杂一点，一台计算机通过一个字处理系统被用于电子邮件分发。

问：我们说的编辑是指什么？

答：对内容进行重新编排、修改，以及通常进行的编辑处理。常见的编辑操作有删除、插入、置换、查找、移动、复制等等。

问：展开单程序是什么？

答：展开单程序是一种计算机程序，它通过把执行的计算限制在单元网络或矩阵里，从而简化了计算机的程序设计。这些单元用来存储与用户规定的公式相关的值。因此，用户程序就由展开单单元间的相互关系来规定。

问：展开单程序为什么这样有用？

答：因为它们减少了为一台微计算机编程序的时间和工作量，它们可用作实际生活中系统的模型。

问：使用一台计算机解决一般问题的步骤是什么？

答：要求、设计、编码说明书、实现方式、测试和文件编制。

问：给出一个程序设计语言的例子。

答：BASIC。

问：什么是DBMS？

答：一个数据库管理系统。这是一种处理数据的程序的集合，按用户的观点存贮在文件里，称作图解。

问：DBMS有哪三种图解？

答：网络式、层次式和关系式。

问：为什么有人要使用DBMS？

答：DBMS减少了完成处理大量数据的程序所需的时间和工作量。

问：为什么有人避免使用DBMS？

答：由于DBMS的通用性，DBMS 很可能不符合你的需要，或因DBMS速度太慢，另外，占有太多的磁盘空间。

第 二 章

计 算 机 词 汇 入 门

对他们从山说起，他们不懂。但是这些话却象雨露洒在山谷一样，使知识的种子扎根成长。

春来秋往，年复一年，理想的果园变成了理想的森林。纵有那荆棘杂草，果园处处仍旧生机勃勃，郁郁葱葱，到处爬满了葡萄藤。所有听过讲的人都获得了丰收、

果园更新的日子到了，又有人要对新来者讲话。对他们从山说起，他们不懂。

计算机的目的

对家庭主妇、卡车司机、独立经营的商人、银行家、音乐家、艺术家、教育工作者、政治家、电影工作者、以及其它各行各业富有创造精神的人们来说，微型计算机的卓越才能已经把数字电子方面的控制了带到家门口。简而言之，由于大规模集成电路(LSI)的发展，微型计算机已经渗透到了各个领域。然而，哪些是对计算技术有着深远影响的事件呢？

在这一章里，我们给读者一条了解计算机词汇的捷径，其目的是介绍计算技术方面的最基本概念，并为生活在信息社会的你作些准备。至于计算机如何工作这方面的详细情

况请参阅第八章。

全部计算技术中决定性的基本设计思想有三条：

1.自动控制——机器在没有人的干予下能够完成某些“智能”操作的概念。

2.存贮器—创造的信息“银行”或贮藏室，它能保存数据，更重要的是，它能保存自动控制机器的其它部份（即存贮器以外的部份）的程序。

3.软件——一层层建立起来的程序，这些程序提高了机器的“智能”水平，使计算机能执行越来越复杂的功能。

自动控制

这一概念是指在没有人的干予下，机器能作出判定并且执行（简单的）任务。或许，这种设想受到时钟的启发，时钟并不需要人来操纵它。同样，计算机如果没有操作员的干予也可以由程序无休止地运行下去。用这种简单的设想人们发明了卡片控制编织的织布机，后来又发明了自动投弹瞄准器（第二次世界大战期间）。

现代的计算机是一些**数字控制器**。早期的电子控制器在连续信号（电压、电流等）状态中工作，所以被称作**模拟**控制器。大多数汽车中的速度指示器就是以每小时多少英里的连续信号装置来测量速度的。但是，由于大规模集成电路的迅速发展，数字电子计算机以它的廉价而优于模拟计算机。数字计算机运行在不连续的或离散的状态下，信息的单位称作**二进制位**。一个二进制位不是赋予某些连续值而是要么是“接通”要么是断开。

全部信息能划分小的“开—关”状态的设想是现代技术中最有深远意义的见解之一。因此，二进制位是信息的基本单

位。由于一个二进制位仅仅可以表示二种可能值中的一种（0或1），所以，全部现代计算机的运算都是用**二进制**数字控制器。一个二进制数的计算机计数是从0到1而不是从0到9。

大量数字和信息可以由一组二进制数**编码**成许多较大的信息单位存贮在计算机里，如在第八章中所讨论的那样。在二进制计算机中所有的信息都能够被编码，并由此实现对大量信息环境的控制，这是此处要记住的要点之所在。

存贮程序装置(存贮器)

计算技术中的第二个突破是电子存贮器的发明。在第八章中我们将会看到如何用电子存贮器来存贮二进制编码信息。这些存贮器的最直接推论是存贮程序的概念。**存贮的程序**是管理计算机操作的一系列指令。

有这么一个概念：人类发明了一种用一系列简单的操作来做某些事情的方法，这就称作一种**算法**。算法被转换成一系列的**机器指令**（控制计算机电路的二进制编码）。当一个程序员用二进制数编码他的算法时，我们就说该算法是**机器语言编码的**，最后，在机器一次检索其中一些机器语言指令机器语言指令的同时也**加载**（复制）到存贮它的电子存贮器中。每一条机器语言指令控制计算机一瞬间，这个过程的细节将在第八章中谈到。

自动控制和计算机存贮器结合在一起给予了计算机足够的能力和灵活性，使得计算机能够计算一切可以被计算的东西。这是一项意义非常深远的成就，三十年来一直促使人类用一些称作“软件”的东西“把握”它。的确，目前我们正处在软件开发时代之中。

软　件

给计算机带来深远意义的第三个概念是软件的概念。**软件**是在计算机上“执行”的程序的总称。软件对计算技术的作用正如蒸气发动在对工业革命的作用一样。

如果人们被迫将算法编码成机器语言程序(二进制的)，那么软件的缺陷会极大地表现出来。幸而，我们有软件这个工具。

现代软件的层次结构很象一个莴苣头。实际上，各层间可以互相作用，因此不可能绝对分开，但是这里我们将把它们作为独立部分来研究。

当然，在里层是机器自身，这由我们称作机器语言的二进制编码指令来表示。由于机器语言对我们来说非常繁琐和乏味，我们几乎总是用一种称作**汇编程序**的软件工具把**符号**转换成二进制机器语言机器语言程序。汇编程序是这样的一种程序，它把用符号形式写的程序翻译成用二进制编码形式写的程序，其中，用符号形式写的程序称作源程序，用二进制写的程序称作目标程序。因此，输入到翻译器里面去的是一串源程序语句，从翻译器里输出的是一串目标程序语句。

汇编语言源程序一一对应地翻译成机器语言目标程序。在这种意义上，它们是低级程序。为了得到较高级的软件我们必须进入到下一层。

程序设计语言是一种高级语言，一个程序设计语言语句可翻译成多个机器语言语句。BASIC，FORTRAN，COBOL 等就是这种程序设计语言。

例如，一个用BASIC语言写的源程序，可以将每一个

BASIC语句翻译成5个或更多的目标指令。这就是为什么程序设计语言提高了程序员的生产率的道理。

有二种（或更多的）方法把高级语言的语句翻译成若干个目标语句：

编译

解释

编译程序是这样的一种程序，它把源程序翻译成目标程序，目标程序当时并不执行，而是存贮起来，在以后某一时刻再执行。Pαscαl，COBOL和FORTRAN就是这种编译语言。

解释程序是这样的一种程序，它不但把源程序翻译成目标程序，而且立即执行该目标程序语句。实际上，解释程序并不产生输出的目标程序，因为没有这种必要。因此，解释程序是一种**直接执行**其它程序的程序。BASIC，LISP和APL就是解释程序的例子。

解释语言与编译语言相比其优点是：

1.开发时间有所缩短，因为新程序能够立即执行，不用耗费翻译阶段的时间。

2.解释程序通常比编译程序要小些（占用的存贮器少些）。

缺点是：

1.解释语言运行时除需要源程序外，还需要有占据存贮空间的解释程序。

2.由于解释是一个很慢的过程，因此解释程序运行比目标代程序慢，甚至要慢10倍。

微计算机的大量程序是用BASIC或Pαscαl写的。然而，

即使是这些源语言对许多人来说也是太低级了。目前对非常高级语言的探索正在把计算机程序设计者引入语言设计的新领域。下面的综述将揭示其中的某些途径。

询问语言处理程序是在处理数据库系统时用来解释非常高级的语句的一些程序。我们可以设想非常高级的询问语言中如下一个程序段

```
FOR ALL NAMES IN MARSTER FILE
        DO;
    BALANCE IS SUM OF ALL
    TRANSACTIONS;
        PRINT BALANCE;
    END;
```

这个询问程序告示了一个记帐程序的帐目新余额。

为了减少程序设计的麻烦，程序员探索的另一种方法是程序编辑程序。**程序编辑程序**是一种能自动写其它程序的程序。例如该编辑程序（生成程序）要求用户输入信息，然后用这个信息来产生一个BASIC，Pascal或COBOL程序。

穿过程序编辑程序的范围，我们就进入了人工智能的领域。**人工智能程序**是一种能用自然语言跟用户通讯的程序。假如有这样一个系统，那么到那时程序设计将会象训练一个助手做一件工作那样容易。

现代电子二进制计算机在其技术的历程中是独特的。因为它能自动地控制它的环境，能通过在一个电子存贮器里存贮二进制编码信号来“记忆”信息，並且通过软件层次日益“智能”化。计算机的发展全部来自于这三个功能。

在下一节里，我们将举步漫游计算机硬件世界，以便学一些计算机硬件专业方面的术语，然后在最后一节，我们

再回过头来看看软件术语。

硬件方面的术语

第八章将讨论硬件的内部工作状况。这儿我们的目的是向你展现计算机界日常使用的词汇，它是计算行家们的“家常语言”。

假如我们讨论汽车的话，根据舒适、安全性和性能来分类它们或许是有用的。同样，一台计算机也是用易用性、可靠性和性能这些术语来衡量的。遗憾的是，易用性、可靠性和性能的估量对新手来说非常陌生，让我们来看看是否能澄清这些陌生的估量方法

CPU

CPU（中央处理部件）由运算逻辑部件（ALU）及主主存贮器或RAM（随机存取存贮器）组成，运算逻辑部件执行ADD（加）、COMPARE（比较）和COPY（复制）一类机器语言操作。

通常，CRT（阴极射线管）屏幕和键盘被连接或**对接**到CPU，所以用户可以与CPU通讯。CRT有一个**光标**或闪烁的光点，它指示计算机已接通，並正在等待你的输入。

在评价CRT的屏幕时我们一般用的是它的大小和显示速度(易用性)这些术语。一个24×80的CRT屏幕可以显示24行每行80列的字符。9600**波特**的屏幕每秒可以写入960个字。因此，不到2秒钟就可写完整个屏幕的24×80＝1920个字符。

计算机的速度通常用**带宽**这个单位来度量。例如，9600

再回过头来看看软件术语。

硬件方面的术语

第八章将讨论硬件的内部工作状况。这儿我们的目的是向你展现计算机界日常使用的词汇，它是计算行家们的“家常语言”。

假如我们讨论汽车的话，根据舒适、安全性和性能来分类它们或许是有用的。同样，一台计算机也是用易用性、可靠性和性能这些术语来衡量的。遗憾的是，易用性、可靠性和性能的估量对新手来说非常陌生。让我们来看看是否能澄清这些陌生的估量方法

CPU

CPU（中央处理部件）由运算逻辑部件（ALU）及主主存贮器或RAM（随机存取存贮器）组成，运算逻辑部件执行ADD（加）、COMPARE（比较）和COPY（复制）一类机器语言操作。

通常，CRT（阴极射线管）屏幕和键盘被连接或**对接**到CPU，所以用户可以与CPU通讯。CRT有一个**光标**或闪烁的光点，它指示计算机已接通，並正在等待你的输入。

在评价CRT的屏幕时我们一般用的是它的大小和显示速度(易用性)这些术语。一个24×80的CRT屏幕可以显示24行每行80列的字符。9600**波特**的屏幕每秒可以写入960个字。因此，不到2秒钟就可写完整个屏幕的24×80＝1920个字符。

计算机的速度通常用**带宽**这个单位来度量。例如，9600

波特是表示CRT速度特征的一种方法。由于每个字符要求8个二进位作为它的编码（参考第八章），以及2个二进制位作为它的传送标志，因此9600**波特**的通讯线每秒可以传送960个字符。其它带宽单位有lpm（行／分）CPS（字符／秒），bps（位／秒）以及波特（二进制位数／秒）。

当程序正在进行时，CPU的存贮器被用来存贮程序和它的数据，这种存贮器称作RAM（随机存取存贮器），这是因为访问这种存贮器的各存贮单元都是一样地容易。有些计算机用一种专门称作ROM（只读存贮器）的非破坏性存贮器来存贮固定的程序和数据。例如，BASIC解释程序常常存贮在ROM里，所以对用户来说它们总是已装好的，只要一开启电源就可立即运行

存贮器的每一个单元都有一个号码，称作它的**地址**。我们把访问一个存贮单元並检索信息所用的时间称作存贮器的**周期**，这个时间是对CPU总速度的度量。因为访问存贮器是一种频繁执行的操作，

存贮器的周期与称为CPU**时钟**的定时器一致。由于CPU的所有操作都是用它的时钟作为时间控制的，所以我们能够用已知它的时钟速率来估算CPU的性能。一个4MHZ（4兆赫兹）的时钟每秒“滴答”4百万次。一说般来，访问存贮器和做某些操作需要4或5个“滴答”声、所以一个4MHZ的CPU每秒大约能处理1百万个存贮单位。

打印机

打印机是用来从计算机那里获取“硬拷贝”的设备。

通常打印机在80或132列的纸上输出文本，打印的字符质量与打印速度有关。

一个点阵铅字是一种矩阵点、用于近似地模仿固体铅字。例如在质量上5×7的矩阵要比9×12的矩阵差些，打印机的价格也是随着点阵密度的增加而增加。

打印机必须通过一个**接口**连在CPU上、串行接口一次只传递一个二进位信息，而并行接口一次传递8个二进位字符。RS—232接口是一个典型的串／并接口。

限定打印机双向打印可以提高打印机的性能，即随着打印机械扫瞄页面来两个方向打印

打印机速度范围一般从300波特（30cps）到1200波特（120cps）。如果输出的是一短行而不是一满行的话，打印机作回车动作的时间将降低有效打印速度。双向打印机减少了打印时回车的这种损失。

全ASCII（读ask—key音）打印机是有全套ASCII（美国信息交换标准码）字符的打印机。有些打印机不是全套的，如不分大小写。

磁盘机

磁盘机是从一个旋转的磁盘上读入读出或写入写出的一种装置。**软盘**是柔软的塑料磁盘，记录速度约为45转／分。**硬盘**是不易弯曲的盘，记录速度约为45转／分或33转／分，取决于容量大小。

磁盘存贮信息的容量通常是用百万个字节（字符）即MByte来度量的。例如，一个20兆字节的盘能够存贮二千万个字符信息。

磁盘的速度取决于它的旋转速度和传递速率。一个盘通常每分钟约旋转360转，传送的速率约为每秒 1 兆 字节（变化很大）。软盘的组织在第八章进行详细讨论

由于磁盘机价格低廉且可移动（温盘除外），因此它们以较低的价格扩充了RAM。不幸的是，访问存贮在磁盘上信息所需要时间比访问RAM所需的时 间要长得 多。因此，在设计许多计算机系统时我们不得不考虑存贮器 的 交 换 速度。磁盘存贮器—正如在第八章中所述，仅是存贮器层次结构中的另一种存贮器。

通　讯

为了发送／接收计算机系统外的信息，可以把计算机**接到**电话**或**其它通迅网络上。把计算机**连接到调制—解调制**一类其它的设备上就可做到这一点。调制—解调器把电子信号转换成声音信号来传送，然后为了接收又把声音信号转换成电子信号。

调制解调器和电话的传送速率一般为300或12 00波特。实际上，电话在没有额外“条件”设备的情况下也可支持高达2400波特左右（240字符／秒）的通迅。

通迅软件与调制解调器及调制解调器接口一道可以把信息从一个计算机系统复制到另一个计算机系统。例如，发送的计算机可发送一个程序给接收的计算机，然后接收的计算机可以用这个程序来处理它的信息。处理结果还可以送回到原来的发送计算机，等等。

调制解调器可以是简单的，也可以是复杂的。一个简单的调制解调器要求人拨电话，一个复杂的调制解调器可不要

人的干予自动地拔电话和自动地回答电话。

总之，计算机硬件非常象一辆小汽车：显示或打印点阵字符时的清晰度，CRT更新或显示整个屏幕的速率，新设备与CPμ接口的容易程度，这些是就衡量易用性的尺度。可靠性用MTBF（平均无故障时间）来量度；性能用兆赫、每秒字符数等等带宽单位来量度。存贮器装置用它们能存贮多少字符（字节）和它们的存取周期这些项目作为它的指标。速度、容量和兼容性是计算机术语中非常重要的关键词。

软件方面的术语

软件是计算机系统的控制部位。没有软件，计算机就毫无用处。软件中最重要的概念是：

- 算法
- 数据结构
- 语言

概括说来，用计算机解题的步骤是，选择（或发现）一种算法（解题的方法），设计一种数据结构——这种数据结构组织数据以适应算法的使用，然后，采用适用于此问题的程序设计语言，实现算法和数据结构。

算　法

算法是解决问题的一种明确的、严谨的和有效的方法。一种算法的构件是下面三种**控制结构**：

1．简单顺序——算法的一个操作执行完后依次接着执行下一个操作

2.分支——要么执行一个操作或操作序列，否则就执行另外的一些操作或操作序列，两者不能同时执行。

3.循环——一遍又一遍地反复执行一个操作或操作序列，直到满足某些条件为止。

最简单的算法是采用深藏于计算机系统最里层的计算程序来实现的。例如，启动一个计算机的算法要做下面几步：

1.一开始就把一个程序从磁盘机上复制到RAM里并且开始执行这个新程序——它被称为**引导装入程序**。

2.新装入的程序把另一种称为**操作系统**的程序加载到RAM里，然后操作系统等待用户的命令。

3.用户经操作系统程序调用其它的程序。例如，用户可以选择调用BASIC解释程序来输入并且执行一个新程序。

一个操作系统象DOS（磁盘操作系统），简单地说是一个帮助用户使用计算机系统的程序。操作系统帮助复制存贮在磁盘上的信息，编辑新的源程序文本，驱动打印机、调制解调器和CRT屏幕。

通过操作系统**命令**，我们告诉计算机去做什么。在某种意义上，操作系统是用来控制其它程序的程序。如果另外一个运行程序由于某种错误而出故障，那么操作系统必须修复该系统，然后帮助用户找到错误。

有些操作系统每次监控执行一个程序，而另外的一些操作系统能用时间的交替的方式执行二个或更多的程序；允许每一个程序在被中断前运行几毫秒，这就是所谓的**多道程序设计**。它给用户外表上似乎在同时执行三个或更多程序的

感觉。事实上每次仅只一个程序在执行，但是由于计算机速度很快，它能够给你这个假象。

例如，一个多道程序的操作系统可以在运行一个打印文件的程序同时又运行另一种程序，使用户能通过键盘和CRT屏幕与系统对话。

另外，**分时**操作系统允许二个或更多的用户通过二个或更多的键盘／CRT终端共享一台计算机。分时是用与多道程序同样的方法运行多道程序，即由计算机交替地操作来达到分时的目的。

最后，**多处理机**操作系统是控制带二个以上微理机的计算机系统。因此，通过相连的多微处理机，两个或更多的程序能真正地同时操作。这样的系统常常被称作**分布式**系统，因为是用若干个分散的微处理机代替单个集中的微处理机。

简单的算法和三个基本的控制结构——顺序、分支及循环——用于构造算法，产生了象DOS一类的复杂程序。但这只是事情的一半，因为没有数据，算法不能有所作为。

数据结构

数据必须组织成信息的逻辑组合，以便算法使用。第八章中将详细地讨论最简单的结构，它们是：

位

字符

整数

实数

然而，许多实际的问题需要更完善的结构。我们仅定义

计算机软件中几个最重要的数据结构。

数组是存贮器中元素，即整数实数、字符等的连接表。我们通过数组的名字和它的下标来存取数组的元素。例如BASIC中数组元素3将以数组B和A$中的B(3)和A$(3)出现。

指针是指向数据结构中被存贮的信息的一个编号。例如元素B(3)的地址可以说就是元素B(3)的指针。

树是一个形状象一颗树那样的分级数据结构。在树结构中，树的指向依次包含信息的分岔，并且元素越多则指针也越为了对磁盘文件中存贮的信息达到快速存取，树结构是非常多。有用的。例如，二叉树(B树)是一个通常用于磁盘索引信息的结构，在第七章中，们们将讨论二叉树，并且给出一个构造它们的程序。

文件是存贮在磁盘上的数据构结文件。文件的元素称作**记录。顺序文件**是存贮在磁盘上的一个数组，存取该数组必须从表的开头直到结尾。**随机文件**也是存贮在磁盘上的一个数组，可以在任何时候存取该文件的任何元素。随机文件似乎很灵活，为什么还要用顺序文件？

顺序文件虽然麻烦而且慢，但它们能存贮可变长度的记录(元素)。这就是说，顺序文件可以用来存贮记录长度可以变化的信息。另一方面，随机文件只能存贮固定长度的记录。

在多数应用场合，我们希望从文件中检索到一个记录，它给出存贮文件中的某种唯一确定的信息。**关键词**是识别存贮在文件中某一记录的信息。

索引文件是包含关键字和指向另一文件的指针的文件。另一文件包括完整的记录，这些记录与通过访问索引文件来

定位的关键字对应。索引文件正如一本书的索引一样，用来寻找书中关键字所定义的唯一信息。

实现一个索引文件的一般方法是用B树数据结构。因此，为了用某些关键字定位记录，首先用索引文件(B树)搜索，一旦在索引文件中找到了关键字，相应的指针就被用来检索完整的信息。

程序员的作业要选择适当的算法和数据结构，这些适当的算法和数据结构要用最短的时间、最少的存贮空间和最可靠的方法解决一定的问题。这些目标常常与另一些目标发生矛盾，因此通常有必要在算法复杂性和数据结构复杂性之间作一些困难的权衡。二者之间选择不当，则会导至低劣的性能和极差的可靠性。

语　言

我们已经讨论了计算技术中程序设计语言的重要性。语言是一种软件工具，因为它能够加快程序员生产产品的过程。这一节我们介绍程序员使用的一些其它术语。

程序由实现一种算法的控制结构及组织被处理数据的数据结构所组成。各种各样的命名习惯常常是为了使程序员详细、直观地认识它们。

给变量一个名字，所以程序员不需要记住对应于变量值的RAM地址。给变量赋值是指把数值送到与变量相对应的RAM的单元中。因此，A＝5意思是把5这个值赋给变量A。

在许多情况下，变量和常量与数据类型是相联系的。数据类型是一种值集。因此，所有的实值形成一种称作“实数集”

的类型，而所有的字符串形成一种称作“字符串集”的类型例如 A $是一种字符串值的变量，而X是一个实质的变量。

在设计上或实现上有缺陷的程序是一个含有“故障”的程序。更换或“修补”程序可以消除故障，这个过程称作**调试**。

减少故障可能性（以及它们对整个程序的影响）的方法之一是把程序分解成称作**过程**的一些小块块。一个过程是源程序的一段，一旦需要时可由另外的程序“调用”。通常程序员把一个大程序当作许多的子程序或过程的集合来设计。

程序执行期间，如果用数字进行数值计算的结果太大或太小，我们就说程序内部产生了**上溢**或**下溢**的故障。如果一个数据结构错误地写到RAM或一个磁盘文件上，我们则说程序遇到了**奇偶校验**错误。这些错误及其它错误均由操作系统**捕捉**。**致命的错误**将使程序停止，並且操作系统负起将控制转向计算机系统的责任。

由于不同应用中特殊处理的需要及历史的原因，存在着各种各样的程序设计语言。BASIC是一种非常老的语言，这种语言是为初学者所开发的。Pascal是一种现代化的语言，它是为了教授一种好的程序设计风格（结构程序设计—译者注）而设计的。FORTAN是一种早期设计的适合于数值计算的语言。而COBOL则是一种早期为事务数据处理设计的语言。没有一种明确的方法来决定哪一种语言是“最好的”，因为它依赖于个人的风格及用途，然而，象COBOL、Pascal和FORTAN77标准化语言很少依赖于机器的特性，所以它们减轻了把程序从一个机器翻译到另一个机器上去的负担。

而且，现代化语言—象Pascal，PL／1和Ada—是为了用来减少实现新软件所需的时间及工作量而设计的。这些语言也努力使程序易读，从而可迅速修正和改动原来的程序。

常见问题

问；计算机有哪二类？

答；模拟的和数字的。

问；信息的基本单位是什么？

答；是二进制数位。它由开关的状态来表示。

问；为什么计算机被称作二进制计算机？

答；因为它们用一组二进制数（0或1）编码信息。

问；一个存贮的程序是什么？

答；是一个当它被用来控制计算机时才加载到RAM中的程序。

问；RAM是什么？

答：随机存取存贮器。一个RAM就是成千个“开—关”状态的集合，根据“记忆”的信息（程序和数据），这些开关被置1或0。

问：汇编程序语言和编译程序语言的不同之处是什么？

答：汇编程序与机器语言一一对应，而编译程序语言则一个对应于多个机器语言。

问：解释程序和编译程序之间的不同之处是什么？

答：解释程序不产生目标代码程序，而直接执行原代码程序，相反，编译程序把源程序翻译成目标代码程序，该目标

程序有时独立于编译程序，在以后某一时刻执行。

问：我们怎样才能测量CPU的速度？

答：CPU（中央处理部件）的速度一般用它的时钟速度（比如说4HMZ）来度量。

问：波特是什么？（发“paud”音）？

答：我们用“二进制位／秒”来表示波特，它是信息传递速度的度量。

问：ROM是什么？

答：ROM（只读存贮器）是一种非破坏性的存贮器。就是说，程序和数据的信息的形式永久存贮在ROM里，甚至电源掉电时也如此。相反，RAM是可以破坏的，一旦电源掉电，它存贮的信息就丢失了。

问：什么是点阵打印机？

答：打印出来的字符形式是一些排列的点阵而不是固定的线，这种打印机称作点阵打印机。

问：接口做些什么？

答：接口是一种连接装置，该装置把信号从一台机器转换成另一台机器可识别的信号。

问：什么是ASCII？

答：一种标准字符码：美国的信息交换标准码。

问：温盘主要优缺点是什么？

答：它们的速度快且能存贮上百万个字符，但它们往往不能从磁盘机上拆下。

问：什么是调制解调器？

答：调制解调（调制器解调器）器是计算机与电话线之间相连接的一种设备。

问：一个算法的组成部件是什么？

答：在一个算法中，顺序、分支及循环支配程序的控制

问：什么是操作系统？

答：控制其他程序执行的一种程序。

问：为什么操作系统並不都是多道程序和分时系统？

答：因为这样的话要附加 RAM 和引起开销（时间损失）。如果不是必须，操作系统不应包含这些“多余的东西”。

问：如果所有的信息以二进制的形式存贮在计算机存贮器里，那么被存贮的数据结构是怎样的？

答：也是一些二进制位。但是，采用编码方案在这群二进制位信息中添意义。

问：随机文件似乎很好，为什么还要用顺序文件？

答：在大多数情形随机文件是最好的，然而，只有顺序文件才能存贮可变长度的记录。

问：由于同样的检索问题存在于索引文件和普通文件之间，为了用唯一的信息标号检索信息，索引文件似乎是不必要的，为什么？

答：首先，索引文件可以用来从其它的文件顺序检索信息，不用对原文件作分类，其次一个文件关键字可以有一个以上的索引文件与之联接，这样可以用很多标号来检索。

问：推荐一种程序设计语言而否定另一种程序设计语言的理由是什么？

答：选择一种程序设计语言代替另一种程序设计语言的理由是：（1）程序设计容易，（2）标准化，（3）可维护性，以及（4）价格.

第 三 章

如何开动IBM个人计算机

有一个太阳按钮，一个月亮按钮。当她把银色的操纵杆拔下时，所有的栎树、枫树和榆树叶子都变成了条条彩虹。

我想起了四月下旬某一天的情景。当时，她关掉了雨水，按下了太阳按钮。她把草地涂成绿色，同时鸟儿顺从地唱起春天的歌。

IBM个人计算机剖析

IBM个人计算机是一种台式**计算机系统**，既有硬件又有软件。若要深入了解各项硬件和软件的术语，请参阅第一、二、八章。下文的前提是你已熟悉这些术语。

硬　件

系统部件

IBM个人计算机（大“盒子”）的主机箱（IBM称它为系统部件）中有一个Intel公司的8088CPU，有从16KB到576KB的RAM，有一个再现声音的机内扬声器，此外，还有多达四台的磁盘机！XT型除容纳一台软盘机外还可容纳一个或多个固定盘。

硬盘，或者说**温盘**，是一种高容量密封硬面转盘。一个硬盘，或一个温盘，一般可贮存一千万字符（甚至更多）。

IBM PC和IBM PC／XT的一个明显区别是硬盘，在XT中硬盘是标准设备。若需要的存贮容量大，那么你最好选择XT。然而，如果软盘能够满足你的处理要求，那么大概PC就足够了。

PC／软盘系统的好处是起价低及有能容纳360,000个字符信息的可拆卸软盘。另一方面，XT／固定盘起价高，但是有能容纳一千万字符信息的不可拆卸固定盘。由于固定盘是不可拆卸的，因此你至少仍需一台可拆卸的软盘机。此外，IBM个人计算机还可容纳一台选购的盒式磁带机及一个或两个CRT(电视机或IBM监视器)屏幕。系统部件用一个分离的键盘作输入，键盘上有83个键，可输入文本、数字和控制信息。你大概也注意到了在系统部件的背后还有几个插口，它们是用于接入这种或那种外部设备的。有关如何使用这些插口的图片及详细说明请参阅《IBM操作指南》。

加电次序

系统部件以家用电作动力。为了开动机器，首先必须查明全部电缆和电源软线是否确实都插在一起了，然后再轻按电源开关，接通电源。

电源。IBM个人计算机的电源开关位于系统部件的右边，是一只很大的红色开关。当系统部件不带软盘加电时，CRT屏幕就会显示下列提示（见屏幕显示3—1）：

```
The IBM Personal Computer Basic
Version C1.00 Copyright IBM Corp 1981
61404 Bytes free
```

OK

但是，如果接通电源开关时DOS软盘片在磁盘机A中（见描述软件的章节），则显示下列提示（见屏幕显示3—3）：

Enter today's date (m-d-y):

在按下电源开关后，显示这两种提示都需要较长的时间，因此要耐心。另外，如果关闭电源开关后要再次接通系统电源，则应注意等候五秒钟。如果要不关掉电源开关就重新起动IBM个人计算机，那么使用CTRL+ALT+DEL键是一种好办法。同时按下这三个键，系统就会按上面介绍的情况重新起动。

加电时之所以会显示两种不同的信息，其原因如下。IBM个人计算机包含有一个40KB的ROM，保存着直接解释BASIC程序设计语言语句的程序。这是一种比较简单的BASIC语言解释程序。尽管如此，即使你拥有的是最廉价的机器，这种简单的BASIC解释程序仍然给了IBM个人计算机"理解"BASIC的"智能"。因此，如果在IBM个人计算机中不能找到一台容纳外加程序的磁盘机，那就可以说它是一台"简单的"BASIC机器。上面所示的第一条提示信息就是一例"简单"机器的提示行。

另一方面，如果IBM个人计算机系统中有一台或更多台磁盘机，那么你能得到另外的"DOS"机功能。DOS代表磁盘操作系统，它是"机内"程序的集合，能用来管理存贮在软盘中的信息。本章中我们将详细考察这些DOS程序。

打印机的连接

为了作文字处理（EasyWriter程序，见第四章）并得

到一份永久性的付本，你大概非常想在系统中接入一台打印机。办法有好几种，可任选一种。若采用IBM打印机，那就应该用37针的**并行**接口插口。若用其它打印机，且IBM个人计算机的各针脚和打印机各针脚的任务都相同，那么就用25针的并行插口，否则就要用专用接口。若你的打印机采用的是**串行接口**，如RS-232C标准接口（详见第八章），那么你就必须买一块串行接口板插入系统部件。

假定你已买了一台IBM个人计算机打印机，那么按下“换行”（LF）按钮，同时接通打印机的电源，行式打印机会不停地重复打印其存贮中的每一字符，这样就可测试该打印机了。一旦你对它的工作感到满意，则关闭电源，然后再接通。此时打印机就可作输出设备使用了。

磁盘机

假定你买了带有两台磁盘机的IBM个人计算机。位于左侧（面向系统部件时）的**磁盘机**称为磁盘机A，右侧的为B。一般，人们都要按下述方法选择一台磁盘机，访问存贮在软盘片上的文件：要访问放在磁盘机A上的信息，则用

A：文件名.附加部份

要访问放在磁盘机B上的信息，则用

B：文件名.附加部份

这里“A：”和“B：”是指磁盘机，“文件名.附加部分”是指文件名称及其**类型**（它包含些什么）。这样，你就可用磁盘机访问软盘A或软盘B了。

通常在XT左方有一台软盘机，指定为A：，右方的一个固定盘指定为C：，另外还可加上一台软盘机，称为B：。

此外，XT使用的是2.00版DOS，而不是1.00版或1.10

版。2.00版DOS可把任何磁盘(软盘或硬盘)分为几个部分，给每一部份规定一个名称，并把这一名称存在分级目录中。我们以后将讨论这种分级目录（称为**树**）是如何工作的。现在，记住存贮在磁盘上的信息可以通过附加的文件名进行检索，附加的文件名包括树中子目录的名称。不过，它只适用于DOS2.00版的系统。

每一个使用IBM个人计算机的人都要记住下列规则：

装盘。插入软盘时，打开磁盘机的门，再把软盘推入磁盘机，一直推到底。即，用手握住软盘，姆指盖住标记，然后再把软盘插入磁盘机，最后再关紧磁盘机的门。

写错误。用户第一次使用时经常忘记揭去“写保护”封条，发生错误。这种错误就是写保护错。在软盘的凹口处贴上封条就可防止再写入，使软盘受到保护，不能写入。如果发生一起写保护错，那就把纸带从凹口上揭去，错误就不会再发生了。

以后，当我们考察IBM个人计算机软件部份时将进一步深入讨论软盘。

CRT屏幕或监视器

CRT**屏幕**可以是各种带有附加“调频器”（把它转换为字符显示）的普通电视机，或IBM监视屏幕。如果用的是彩色电视，那么你就能以八种颜色的两种不同强度显示字符和图象信息（总共16种色调）。但是这种选购件需要有彩色图象适配器。CRT屏幕是“光标寻址的”。它的意思是有一个光标（象一条划在字下的闪烁的线），它能在屏幕上的任何地方移动。例如，不管是打正文还是敲键盘的空格键，光标都移向屏幕右边。不过更重要的是EasyWriter字

处理程序和VisiCalc工作单程序一类软件包都要用光标来指导你使用它们。

你可以选用40列（黑体字）或80列（宽行）格式的CRT屏幕。40列的易于阅读，用彩色电视机作CRT屏幕时更是如此。当然，80列屏幕可容纳更多的信息，并与行式打印机一致。除非你按下列方式命令改变格式：

A>mode 40

否则IBM个人计算机软件自动选择80列格式。重新输入命令：

A>mode 80

屏幕就回到80列了。

本章中我们还要讨论这一命令及它的其它用途。

键盘

IBM个人计算机有一个加大的打字机键盘，即，它有一台普通打字机的键（这些键是浅色的），加上用于程序设计和运行EasyWriter一类程序的外加键。为避免损坏，你必须先认识到某些外加键是“专用键”。如，有几只“触发开关”，它们控制大小写、数字小键盘等。触发开关使人想起它很象你家墙上的电灯开关：每次轻按时它就会调整到反向位置。因而，每次使用它时不是“开”就是“关”。

键盘提示

CAPS LOCK：CAPS LOCK键是一只触发开关，它使打字机键以大写字母方式工作。

NUM LOCK：NUM LOCK键是一只触发开关，它使数字小键盘的键（右边的键）既作数字键又作光标控制

键〔观察箭头和“home(复位)”、“page-up(页上移)”及“page-down(页下移)”键〕。

←：“左箭”键是退格键，能消去最后送入的一个字符。

PRTSC：打印屏幕键，与“Shift(换档)”键联用时可在打印机上打印屏幕显示的内容。

CTRL：该“控制键”与其它键联用时可控制计算机。下面是某些CTRT功能简表：

CTRL+NUM LOCK=暂停，计算机等待你再按一个键，然后继续工作。

CTRL+ALT+DEL=重新起动，使计算机停下它所作的一切工作，重新起动时与关闭电源一样，通电，再开始。

CTRL+C=撤销，撤销你刚送入的行。

CTRL+PRTSC=回应，同时打印你录入的和屏幕上显示的。

功能键：10个功能键在不同的程序中有不同的意义。当不运行程序时，它们用于重新显示前面的命令，等等。

箭头：有几个标有箭头的键，它们的用法如下。

向左(←)=最上面一行的为退一格；右边中间行的为“回车”。回车键用于结束一行的输入。

向上(↑)=左右各一个，大写字母键。

向左向右(←→)=在左边第二行，制表键。

ALT：ALT用于生成扩充的ASCII+进制码。当送入ALT+n时，83个字符的键盘能生成255个字符。若n等于48，生成字符零。还有，注意ALT+216是生成一个特殊符

号（见用诊断软盘得到的显示）。这一键的重要特点是能显示和打印希腊字母。

控制字符说明

很多计算系统的控制键采用标准的含义集。例如，IBM的键盘按下列方法使用时与很多其它键盘（如，CP/M键盘）非常相似。（CTRL表示CTRL键）

CTRL+C	空去这一行，另起一行。
CTRL+H	退格
CTRL+I	制表
CTRL+J	换行
CTRL+M	返回
CTRL+P	回应印出
CTRL+S	显示大量信息中的暂停，如在TYPE命令中暂时停止屏幕卷动（见下两节）。
CTRL+Z	操作台输入终结记号。

软　件

IBM个人计算机的软件录制在5 1/4" 的软盘上，随机器一起供应。在本章中，我们将只讨论标为DOS（磁盘操作系统）和DIAGNOSTICS（诊断）的两种软盘。在以后几章中，我们将讨论其它软盘，包括文字处理、展开单计算和解一般问题的软件。当然，你也可以另外再买些软件。

注意每张软盘上的标签，它标明软盘上存贮的信息。此外，还要注意沿着每张软盘的一边有一条“写保护”凹口，在软盘上记录任何数据或指令前必须打开这一凹口。

在开启电源前应把标为“DOS”或“System”的DOS操作系统软盘插入磁盘机A（左边的那一个）。30秒钟加热后你就能得到屏幕显示3—3中所示的“热启动”提示。

送入数据（如5—10--83）和敲打ENTER（回车）键（←），这样就会使DOS操作系统负起控制计算机系统的责任，等待你送入一条命令。命令示提行

A>

意为DOS正运行于磁盘机A，并等候你的指令。

此时，一般都在磁盘机B内装入一张软盘，然后选取存贮在该盘上的某一程序。例如，你可以在磁盘机B内插进一张VisiCalc软盘，并指令DOS运行VC80（80列形式的VisiCalc）。

A>B：VC80

或者，你可用BASIC或高级BASIC开始。此时你可送入BASIC或BASICA（高级BASIC）命令来启动。

A>BASIC

A>BASICA

如果你想立即开始使用DOS软盘提供的BASIC示范程序，那么就在高级BASIC命令后再加上程序单名，如下：

A>BASICA

略停一下后，屏幕上就显示出屏幕显示3—2中的提示，然后你就可送入下列BASIC命令：

LOAD “SAMPLES” (ENTER)

RUN

这是IBM个人计算机的速成法，但是要真正做工作我们还必须先做一些头等重要的事。

在第一次见到IBM个人计算机时你就应该进一步熟悉DOS，把软件软盘全部复制下来，如DOS、诊断软盘等等。但是请注意有些软盘软件是得到保护的，不能复制，而另一些软盘则含有自己复制的程序。在下几章中，我们要学习几种这类程序软盘的使用方法。

```
The IBM Personal Computer Basic
Version C1.00 Copyright IBM Corp 1981
61404 Bytes free
OK


 1 LISI  2 RUN  3 LOAD''  4 SAVE  5 CONT
6''LPT1  7 TRON  8 TROFF  9 KEY  OSCREEN
```

屏幕显示3—1 不带DOS软盘时的启动标识记号。

ROM BASIC

```
The IBM Personal Computer Basic
Version A1.00 Copyright IBM Corp. 1981
35707 Bytes free
OK



1 LISI  2 RUN  3 LOAD''  4 SAVE''
5 CONT  6''LPT1  7 TRON  8 TROFF
9 KEY  0  SCREEN
```

屏幕显示3—2 A>BASICA

从DOS软盘得到的高级BASIC

```
Enter today's date (m-d-y): 5-16-82
The IBM Personal Computer DOS
Version 1.00 (C) Copyright IBM Corp 1981
A>_
```

屏幕显示3－3　DOS启动标识记号
这是当DOS磁盘在磁盘机A中时
得到的屏幕显示

头等重要的事——DOS1.10

当你第一次使用IBM个人计算机时，或你认为机器可能坏了时，首先运行诊断软盘。这一工作的正确做法是将诊断软盘插入磁盘机A，再重新启动计算机(见前节装盘说明)。记住，你可以轻按电源开关重新启动IBM个人计算机，也可以在接通电源的同时按下三个键来重新启动机器：

CTRL+ALT+DEL

无论何种情况，30秒钟延迟后（时间长短取决于系统中存贮量的大小)都会出现屏幕显示3—4中的“菜单”(menu)。这一软盘上的程序让你做三件非常重要的事情：

0　检查硬件是否正确工作。

1　把一张或多张新软盘格式化，使它们能用于存贮程序、数据或正文。

2　原盘发生故障时复制下当时的软盘，这样你就有一份备用付本了。一般，你应该有两份软盘上有用信息的付本。

```
The IBM Personal Computer
DIAGNOSTICS Version 1.00 (C)
Copyright IBM Corp 1981

SELECT AN OPTION
0 -RUN DIAGNOSTIC ROUTINES
1 -FORMAT DISKETTE
2 -COPY DISKETTE
9 -EXIT TO SYSTEM DISKETTE

INSERT DIAGNOSTIC DISKETTE IN
DRIVE A AND ENTER THE ACTION
DESIRED
0
```

屏幕显示3—4　诊断程序主菜单
选择零是检查机器是否有故障

运行诊断程序

格式选择

屏幕显示3—4所示为运行诊断软盘时显示的内容。如果你只想测试硬件是否正确工作，那么选择0。然而，我们在冒着破坏有用信息的风险前要先复制下整个系统软件作备用。因此，我们必须格式化一些软盘，然后把系统软盘（及其他软盘）复制到新格式的“空白”处（空盘是没有记录任何信息的软盘）。

1　（ENTER）

现在，在磁盘机B中插入新盘，然后送入B：

B:　(ENTER)

实际上，只要你反复选择1，你就能格式化所需的全部盘软。要多格式化一盒软盘留作以后使用。屏幕显示3—5这是一过程的一个例子。

小心：在存入信息后不要再格式化固定盘。格式命令会擦除磁盘内的全部信息，一千万字符的信息！

```
SELECT AN OPTION
 0 —RUN DIAGNOSTIC ROUTINES
 1 —FORMAT DISKETTE
 2 —COPY DISKETTE
 9 —EXIT TO SYSTEM DISKETTE
INSERT DIAGNOSTIC DISKETTE IN
DRIVE A AND ENTER THE ACTION
DESIRED
1
WHICH DRIVE CONTAINS DISKETTE
TO BE FORMATTED? b
SELECT AN OPTION
 0 —RUN DIAGNOSTIC ROUTINES
 1 —FORMAT DISKETTE
 2 —COPY DISKETTE
 9 —EXIT TO SYSTEM DISKETTE
INSERT DIAGNOSTIC DISKETTE IN
DRIVE A AND ENTER THE ACTION
DESIRED
2
WHICH DRIVE CONIAINS SOURCE
DISKETTE? a
WHICH DRIVE CONTAINS TARGET
DISKETTE? b
INSERT DISKETTES—PRESS ENTER?
```

屏幕显示3—5　用诊断程序格式化和复制系统软盘

复制选择

接着，按下法一次复制一张系统磁盘。选择2。

```
2 (ENTER)
WHICH DRIVE CONTAINS SOURCE
   DISKETTE? a (ENTER)
WHICH DRIVE CONTAWS TARGET
   DISKETTE? b (ENTER)
INSERT DISKETTES=PRESS ENTER
```

把DOS软盘放入磁盘机A，格式化后的空盘放入磁盘机B，然后按下ENTER键。全部能复制的软盘都要这样重复做一遍。这些是你的备用付本，所以应保存在安全的地方。

诊断选择

下一步使用IBM个人计算机系统的准备工作是检查硬件。从屏幕显示3—6所示的菜单中选取0就能完成这一工作。

```
0 (ENTER)
```

IBM系统部件答以系统包括的全部设备和存贮器清单。如，你可能会得到下列清单。

```
SYSTEM BOARD
128KB MEMORY
KEYBOARD
COLOR/GRAPHICS ADAPTER
2 DISKETTE DRIVES &
   ADAPTER
```

```
2 -COPY DISKETTE
9 -EXIT TO SYSTEN DISKETTE
INSERT DIAGNOSTIC DISKETTE IN
  DRIVE A AND ENTER THE ACTION
  DESIRED
0
THE INSTALLED DEVICES ARE
SYSTEN BOARD
  64KB MEMORY
KEYBOARD
COLOR/GRAPHICS ADAPTER
2DISKETTE DRIVE(S) & ADAPTER
IS THE LIST CORRECT (Y/N)? y
SYSTEM CHECKOUT
0 -RUN TESTS ONE TIME
1 -RUN TESTS MULTIPLE TIMES
2 -LOC UTILITIES
9 -EXIT DIAGNOSTIC ROUTINES
ENTER THE ACTION DESIRED
?
```

屏幕显示3—6　诊断程序查出装入的设备，然后显示系统检验单

如果这一清单是正确的，送入Y（ENTER）。诊断程序将继续逐个检查这些东西。下一张清单见屏幕显示3—6，它让你选择测试的次数。这里我们只解释选择0（测试一次）。

0（ENTER）

测试每个部件都需要你的帮助。如果有一个部件失效，应记下故障号码，以便向授权的商人报告该故障。系统检查每一部件并列出清单。如SYSTEM UNIT 100 指的是RAM。

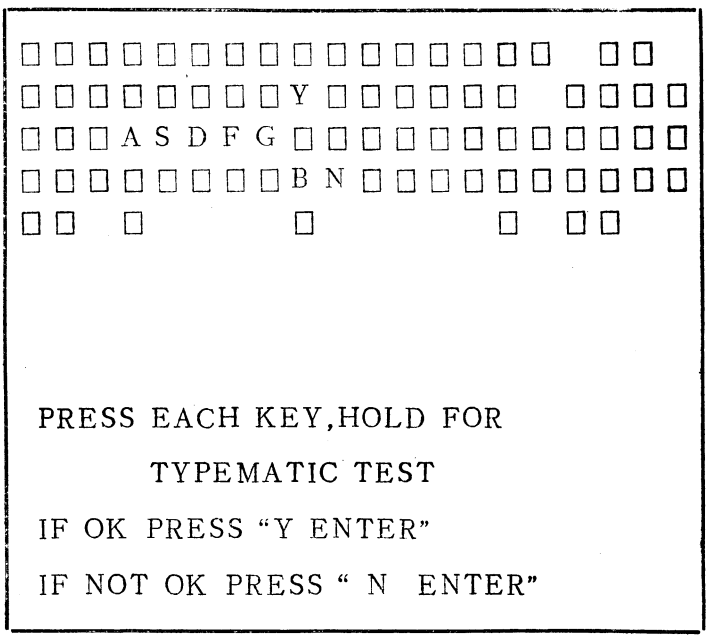
SYSTEM UNIT 100（主存贮器）

SYSTEM UNIT 200

SYSTEM UNIT 300 （键盘）

SYSTEM UNIT 500 （磁盘机）

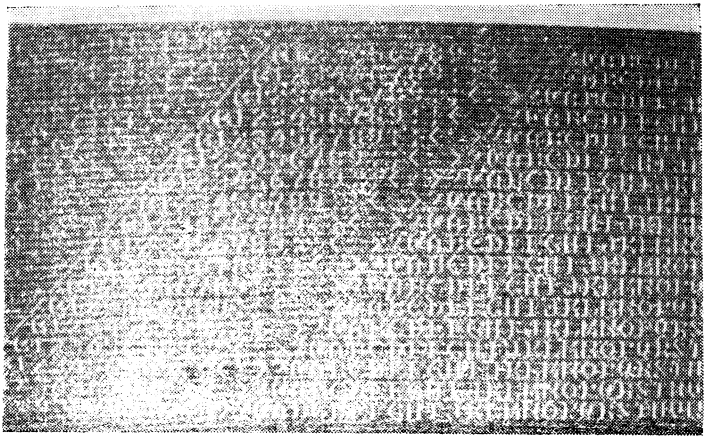
屏幕显示3—7、3—8、3—9和3—10是测试中的几个步骤。如果想要整套显示图，那你就必须自己运行诊断软盘。



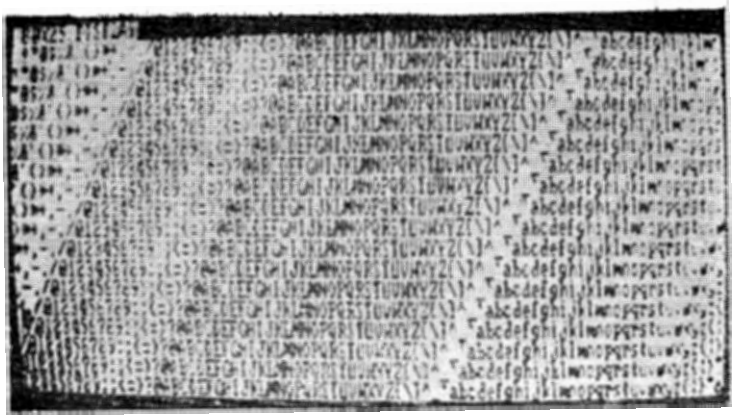
屏幕显示3—7 键盘测试。
用户敲打每一个键，
检查其显示是否正确。

```
DISPLAY ATTRIBUTES
THIS LINE IS AT NORMAL INTENSITY.
THIS LINE IS INTENSIFIED.
THIS LINE IS IN REVERSE VIDEO.
THIS LINE IS BLINKING.
    BLUE
    GREEN
    CYAN
    RED
    MAGENTA
    YELLOW
    WHITE
IS THE SCREEN CORRECT? (Y/N) _
```

屏幕显示3—8　屏幕测试用户在彩色TV监视器上检查颜色品质



屏幕显示3—9　40列字符宽的字符集测试



屏幕显示3—10

80列字符宽的字符集测试

DOS种种

现在你已经可以开始使用机内DOS软盘的程序了。把DOS软盘装入磁盘机A，按下CTRL+ALT+DEL这三个键，略过一会儿，它就会指点你送入日历日期。如果你送入以下日期（或正确日期）：

5—12—84

你就会看到IBM个人计算机的标识记号和DOS“注册磁盘机”的提示，见屏幕显示3—3。

A>

送入B：（ENTER），“注册磁盘机”就变为B。

B>

最后，如果你送入A：，注册磁盘机又回到磁盘机A。除非你规定其它不同情况，否则DOS程序认为**注册磁盘机**就是你要使用的磁盘机。

屏幕宽度

用MODE（方式）命令可以把CRT屏幕宽度设定为40个字符或80个字符。MODE命令也可用于调整显示的左右方向，如下例。

MODE 40, R, T

这条命令是设定屏幕的宽度为40个字符，允许你调整左边边界（R）。

MODE 40, L, T

这种形式的MODE命令也是设定40个字符宽，但允许你调整右边边界。

MODE 80, R

MODE 80

这两条命令让你设定80个字符宽，但是不能测试边界调整。

打印机宽度

MODE命令也用于设定打印机的参数，如：

MODE LPT1：, 80, 6

MODE LOT1：, 132, 8

这两条命令设定打印的页为80列和132列，每英时行数为6或8。

时间和日期

接下去就是建立时间和日期。送入下述命令就可做到这一点。

TIME（ENTER）

当时时间将以“小时：分：秒”的形式显示出来（缺省是零）。你可以送入一个新的时间，也可只按下ENTER键，不送入新的读数，听任时钟自然走下去。以后当你想看时间时就再送入一次TIME（ENTER）。

DATE（日期）命令的工作情况与TIME（时间）命令非常相似。

DATE （ENTER）

它使你能看到现在的日期（偶然忘了时），或改变日期——如果你想改变日期。

磁盘上有些什么？

每一张格式化的软盘有一张内含文件表，表中有一张存贮在软盘上的全部其它文件的清单。这一张内含文件表称为软盘目录，因为它是软盘上的目录。用DIR命令可以查阅软盘目录。这里说明如何使用DIR命令：

DIR
DIR A:
DIR B:

上述第一种形式是在屏幕上显示注册软盘的目录。第二种和第三种形式是不管注册磁盘机发生什么情况，分别显示目录A和目录B。

DIR *.COM
DIR A:*.COM

这种形式的DIR命令是只显示COM（命令）文件名称（见屏幕显示3—11）。注意文件名称中“*”和“？”的区别。

DIR A:BASIC?.COM

这一条命令可以显示下列两个文件，这是因为“？”可以是空白，也可以是单个字母A。

BASIC COM 10880 08—04—81

BASICA COM 16256 08—04—81

它们显示出文件名、附加部份或类型、字符长度、上次在盘上记录文件的日期。因而，BASICA是一个含有COM软件的文件，16256字符长，上次记录时间是1981年8月4日。

```
A>dir *. COM
COMMAND     COM     3231    08—04—81
FORMAT      COM     2560    08—04—81
CHKDSK      COM     1395    08—04—81
SYS         COM      896    08—04—81
DISKCOPY    COM     1216    08—04—81
DISKCOKP    COM     1124    08—04—81
COMP        COM     1620    08—04—81
DATE        COM      252    08—04—81
TIME        COM      250    08—04—81
MODE        COM      860    08—04—81
EDLIN       COM     2392    08—04—81
DEBUG       COM     6049    08—04—81
BASIC       COM    10880    08—04—81
BASICA      COM    16256    08—04—81
A>
```

屏幕显示3—11

DOS命令DIR*. COM只显示命令文件

何时磁盘满了

本章后面部份我们将看到如何生成一个文件和放入信息。每张软盘能生成64个文件，每个文件可保存160KB信

息。由于每一扇区（软盘空间的物理单位）保存512个字符，因此，你能用一个、两个、直至四十个数据扇区生成文件。但是如何知道软盘已满了呢？

INSUFFICIENT DISK SPACE

如果你要存入一张磁盘的信息太多了，没有那么多空间，那么它就会显示出这一信息。如果用CHKDSK命令，那就可事先作检查，如屏幕显示3—12即是一例。

CHKDSK

CHKDSK B:

这些命令告诉你现有的文件占了多少空间，更重要的是它告诉了你还有多少空间可用，然后你可决定是否启用一张新软盘，或按下法把不用的文件去掉。

```
A>chkdsk a:
          43  disk  files
      160256  bytes total disk space
        4608  bytes remain available
       65536  bytes total memory
       53392  bytes free

A>_
```

屏幕显示3—12

检查磁盘机A中磁盘的可用空间

擦除磁盘文件

ERASE命令可以从软盘中擦去任何一个磁盘文件。记住，一旦一个文件被擦除，它就不能再恢复。所以擦除前一定要确信你不会再使用该文件了。

```
ERASE A:NOTE.TXT
ERASE B:?O?E.*
ERASE *.*
```

第一个例子是从磁盘机A中擦去文件NOTE.TXT。第二个例子是擦去磁盘机B中分别以O和E为第二和第四个字母的全部(.*)类文件。第三个例子是擦去全部注册磁盘机上的文件!

改变文件名称

有时，你想要改变一个文件的名称。例如，你可能已生成文件NOTE.TXT，但是后来想把它作为一个BATCH(批)文件使用，而不是TXT文件。RENAME(重新命名)命令可以改变文件的名称:

```
RENAME B:NOTE.TXT  B:NOTE.BAT
RENAME B:NOTE.TXT  B:*.BAT
```

这两个例子做的是同样的事。"*"的意思是在新的和旧的文件中使用同一个文件名称。RENAME命令的一般形式如下:

```
RENAME 旧文件名  新文件名
```

你也可使用单个字母置换规则，如下例所示。

```
RENAME MAKE.BAT*??D?.*
```

这条命令是在注册磁盘机上把MAKE.BAT改为MADE.BAT。

先试一下这些基本DOS命令，然后再翻到下一节，进一步学习机内DOS程序。

有关文件类型的分界线

在DOS中，一般都用某些文件扩充码来指明文件中存

贮信息的类型。这些习惯上的扩充码分列如下：

ASM	汇编语言源程序
BAK	备用文件
BAS	BASIC语言源程序
BAT	批命令文件
COM	命令程序文件
DAT	数据文件
DOC	文献文件（来自文字处理机）
EXE	可执行（机器语言）文件
HEX	十六进制格式机器语言文件
INT	编译程序产生的中间编码文件
LIB	库存程序文件
OBJ	目标码（机器语言）文件
OVR	复盖文件（程序使用）
PRN	打印机的程序列表文件
REL	浮动机器语言文件
$ $ $	未知文件或临时文件
VC	VisiCalc工作单文件

普通的DOS操作

加电，或仅在磁盘机A中装入适当的磁盘，重新启动机器，这样就可直接执行很多程序。实际上，EasyWriter、VisiCalc和本书中讨论的其它程序可以说是IBM计算机中使用得最多的程序，但决不能说只有这些程序。一种称为DOS（磁盘操作系统）的专门程序集合体是“建在”IBM

个人计算机系统之内的。这些程序非常普通，非常有用，已成为系统的一个部份，因此被称为“系统软件”。让我们看一下其中某些非常有用的普通DOS程序，看看它们能做些什么。

记住，提示线“A>”意为注册在磁盘机A上。提示线后只需立即打上程序名就能执行磁盘机A上的任何程序了。

如果你打入一个并不存在的程序名（拼错文件名或弄错磁盘机），DOS就会答以下面所示的信息：

```
A>myprog
Bad command or file name
```

如果不想执行注册磁盘机，而是想执行存贮在磁盘机B上的某一程序，那么，你可以在程序名前加上前缀B。例如，程序VC80存贮在磁盘机B上：

```
A>B:VC80
```

这一命令是从磁盘机B中检索80列形式的VisiCalc，并把它装入RAM以便执行。

另一种访问磁盘机B中程序的方法是转换注册磁盘机：

```
A>B:
B>
```

现在，B是缺省磁盘，除非加上前缀A:取出的程序都在磁盘机B上。

当IBM个人计算机与磁盘机A中的DOS磁盘一起启动时，如果DOS程序存贮在主存贮器RAM内，此时DOS程序是内部的（I）；如果它们在需要前一直存贮在DOS软盘

中，那么它们就是外部的（E）。内部DOS程序在加电后总是驻留在RAM中，所以即使取出DOS软盘，我们仍可使用它们。相反，外部(E)程序只有在DOS软盘放入磁盘机时才能使用。这两种DOS程序都称为“命令”，但是由于外部命令来来往往，出入于RAM，所以，它们有时就被称为瞬时命令。内部（1）命令也被称为固有命令。下面我们列出这两种命令。

固有的DOS命令

COPY　把一个磁盘文件从一个磁盘区复制到另一区。
Dir　列出磁盘内容表。
Erase　从一张磁盘中擦除一个磁盘文件。
Pause　停顿下来等待按一下键再继续下去。
Rem　处理批文件时显示记事。
Rename　重新命名一个磁盘文件。
Type　显示磁盘文件内容。
(batch)　用户用其它命令定义的命令。

瞬时的DOS命令

autoexec　计算机第一次启动或重新启动时自动执行的（批）文件。
Chkdsk　显示磁盘状态。
Comp　比较磁盘文件以相互验证。
Date　送入日期。
Debug　程序调试工具。
Diskcomp　为验证一张磁盘将它与另一张磁盘作比较。

Edlin	用于程序准备的行编辑程序。
Format	准备适合DOS使用的磁盘。
Link	把两个或更多的程序连结起来。
Mode	设定显示屏幕或打印机的宽度。
Sys	把DOS文件转移到另一磁盘。
Time	显示时间，送入新的时间。

现在，我们考察一下这些命令程序，用举例的方法说明它们是如何工作的及有什么用途。每一命令的详细情况见IBM磁盘操作手册。

DIR

DIR目录命令是在屏幕上显示 盘内容表。要显示DOS磁盘目录，我们可打入DIR或DIR A：。要显示屏幕显示3—11所示的瞬时命令文件，送入下列命令：

DIR*. COM

注意，命令文件的名称打为“.COM”文件。它的意思是这些文件是COM命令程序，可以按DOS命令的一部份来执行（它们是DOS命令）。这些是非常有用的命令程序。

“*.COM”的意思是显示磁盘上的全部命令文件。每当我们想要得到有关DOS命令的全部文件名时，如上面举的例子，我们就用“*.”。但是如果我们只想显示一个文件，那就一定要用全称。一个问号可以代替名称中的一个字母。若要显示那些名称为一个字母的文件，其命令如下：

DIR?.*

如果我们想看那些存贮在磁盘机B上的名称为四个字母的文件，我们可以按下法做：

DIR B: ????.*

DIR显示是显示出DOS磁盘上的全部命令，但是不显示系统磁道。它们含有随时供你使用的机内命令。

这些程序主要对系统程序员和应用程序员有用，然而，其中有些对一切使用IBM个人计算机的人来说都是非常有用的。我们在本章的其余部份将只讨论最有用的程序。

CHKDSK

打入

```
CHKDSK
CHKDSK A:
或          CHKDSK B:
```

你就能知道磁盘中还有多少空间（前面已简要讨论过这一命令）。

例如，打入下面所示的CHKDSK，你就可检查磁盘机A中的DOS磁盘：

```
A>CHKDSK
```

它的回答会告诉你有多少文件存贮在磁盘上及还有多少字节（字符）的空（见屏幕显示3—12）。

```
40      disk file
160256  bytes total disk space
6144    bytes remain available
65536   bytes total memory
53392   bytes free
```

因此，DOS磁盘的内容表中有40个文件，它们总共占用（160256－6144）＝154112字节磁盘空间。RAM中总共有65536字节，还有53392字节空着。

FORMAT

如前节所示，你可以用诊断软盘程序把一张新软盘格式化。但是，你也可以用DOS中的FORMAT（格式）命令在任何时候格式化一张新软盘：

FORMAT B:

这一命令是使磁盘机B中的磁盘作好准备，以备IBM个人计算机系统使用。这里再提醒一下，新软盘在磁盘机可以使用前必须先格式化：把软盘中的“空白”格式化、建立空白目录以及其它第一次使用时要作的杂务——如预置文件分配表（FAT）。这张表保存已分配给文件的磁盘扇区的全部磁道，依次是全部“自由”扇区。注意，一个文件可以由软盘上的邻近扇区组成，也可以由非邻近扇区组成。有时把文件分成很多不相邻的扇区是必要的。因此，一定要预置一张软盘表，这样DOS才能在以后使用它们。

Formatting……Format complete

在成功地进行FORMAT操作时及完成后，屏幕上就会显示这一信息。在这一命令中，你能格式化随意数量的软盘。

如果你打算让系统软件使用一张软盘，即，以后用下面介绍的SYS（系统）命令把DOS内部固有命令转移到软盘，那么你可以用下列形式的FORMAT命令（见屏幕显示3—13）：

FORMAT B:/S

FORMAT A:/S

```
A>format b:/s
Insert new diskette for drive B:
and strike any when ready

Formatting……Format complete
System transferred

Format another (Y/N) ?n
A>chkdsk b:
            3 disk files
       160256 bytes total disk space
       147968 bytes remain available
        65536 bytes total memory
        53392 bytes free
A>_
```

屏幕显示3—13　系统磁道格式化
三个系统文件见CHKDSK命令

这是使FORMAT命令既可以格式化又可记录磁盘上的系统文件。不过，你大概不想把这些DOS文件放到一个新软盘上，因为它们要占用很多空间。你更想用这些软盘空间存贮数据。

如果你在空白软盘上格式化和记录系统文件，那么CHKDSK命令将展现三个使用中的文件。这些文件是DOS系统文件，包含固有命令。

FORMAT也检查新软盘是否有损坏的扇区。全部坏的扇区都分配给称为BADTRACK的文件。如果CHKDSK报告了三个以上文件（FORMAT中没有/S的0文件），那么剩下的文件是BADTRACK。损坏扇区的数量等于BADTRACK的长度除以512。

SYS

现在，假如你已格式化一张空白软盘，留下了存放DOS固有命令的地方。用下列方法你就可以把一张现有DOS软盘上的系统文件放到空盘（在磁盘机B内）上：

SYS B:

这一命令是在磁盘上复制一份新的DOS系统文件。如果软盘不是空的，那么在完成这一命令之前一定要先把系统文件分配给该软盘。不能执行这一命令时它就会报告下列信息：

No room for system on destination disk

回避这一障碍的唯一办法是用系统文件格式化另一软盘（FORMAT B:/S），把文件从老软盘复制到新格式化的软盘上。下面讨论的DISKCOPY（复制磁盘）命令是用于复制整个软盘的。COPY（复制）命令是复制一个或一个以上的单个文件。

DISKCOPY

这一命令可以把整张源盘复制到一张目标盘。

DISKCOPY 源 目标

如果源盘和目标盘的磁盘机不同，那么就要用两台磁盘机，否则用一台磁盘机。

DISKCOPY

DISKCOPY A: A:

上面这两条是单磁盘机磁盘复制命令。下面是两台磁盘机的命令（见屏幕显示3—14）。

DISKCOPY A: B:

DISKCOPY B: A:

这两个付本是等同的，互为付本（镜式映射）。即，不必重新组织文件和扇区（见COPY A:*.*B: 的说明）。

```
A>diskcopy a: b:
Insert source diskette in drive A
Insert target diskette in drive B
Strike any when ready
Copy complete
Copy another? (Y/N)
—
```

屏幕显示3—14　把磁盘机A中的软盘复制到磁盘机B中的软盘

DISKCOMP

在制作一张软盘的镜式映射付本后，一定要验证这两张磁盘是否等同。DISKCOMP命令是用来捕捉复制软盘中的错误的。因而，如果源盘放在磁盘机A中，那么新复制的软盘就放在磁盘机B中，做法如下：

DISKCOMP A: B:

如果怀疑软盘是否正确，但是磁盘机又坏了，那么你可以运行诊断程序，或按下面所示比较软盘本身：

DISKCOMP

DISKCOMP A: A:

这种形式的DISKCOMP也能用单台磁盘机比较两张软盘。

COPY和COMP

除下列几点外，COPY和COMP命令的工作与

DISKCOPY和DISKCOMP一样：

（1）COPY和COMP只处理单个文件。

（2）COPY能重新组织文件，可以提高系统性能。用COPY复制那些有大量磁盘活动的程序时能改进程序性能，这是之所以用COPY而不用DISKCOPY的一个原因。

COPY 源 目标

这一命令是把源文件复制到目标文件。一条命令可以复制一个或一个以上文件。例如，把整张软盘从磁盘机A复制到磁盘机B中的软盘上（并重新组织）：

COPY A:*.*B:

这一命令把A中的全部文件复制到B（*.*），目标中不必重复文件名称。

COPY命令用于把信息从某地转移到另一地方，它并不考虑地址分配。

COPY A: NOTE.TXT A:NOTE.BAT

这一命令是把磁盘机A中的NOTE.TXT的内容转移到磁盘机A的NOTE.BAT文件中去。这两个文件在转移后含有相同内容的信息（见屏幕显示3—15）。

现在研究下面的例子：

COPY MAKE.TXT??D?.*

你可能会认为这一命令是把注册磁盘上的MAKE.TXT的内容转移到一个新的文件中去，这个新文件的名称由称为MADE.TXT的源文件构成。对了！COPY命令构成目标文件时用的是下列名称：MADE.TXT。

```
A>copy a: note.txt b:
          1 File(s) copied
A>-
```

屏幕显示3—15　把单个文件复制到磁盘机B中的软盘上。文件名相同

打印屏幕上的一个文件

最后，如果你想看一下文件内部有些什么内容，那么用下列TYPE（打印）命令就可以做到这一点：

TYPE　文件名

这一命令是　把文件显示在屏幕上。例如，用下列命令可以显示MAKE.TYT：

TYPE　MAKE.BAT

TYPE　B:MAKE.BAT

屏幕显示3—17所示是这条命令的一个例子。

文件可能太大，屏幕不能胜任。如果这样，文件就会飕

```
A>copy con:  note.txt
Grocery list
eggs
butter
meat
milk
bread
salt
^Z                      1 Flie(s) copied
A>_
```

屏幕显示3—16　从键盘把正文送入文件

```
A>dir*.bat
CREATE      BAT        69
XFER        BAT        18
AUTOEXEC    BAT       173

A>type create.bat
type %1 :create.bat
%1 :format %2:
diskcopy %1: %2:
%1: chkdsk %2:
A>create a on b_
```

屏幕显示3—17　生成批文件及其使用

飕地飞过你的眼前，无法阅读。用CTRL+S键或专门组合的CTRL+NUM LOCK键可以停止屏幕的卷动。按下CTRL+S一秒钟后，或在CTRL+NUM LOCK后再按任何一个键，显示就又会继续下去。

如果你在文件名称中使用了“*”或“？”，那么TYPE命令就会取出并显示它找到的第一个文件。即使还有其它文件也符合这一文件名形式，它们也不会显示出来。例如，假设已有MAKE.TXT和MADE.TXT文件，那么它只会打印出磁盘目录中的第一个文件。

```
TYPE MA?E.TXT
TYPE *.TXT
TYPE *.*
```

所有这些命令都只显示它们找到的符合该文件名形式的第一个文件。（注意，有些文件含有不可打印信息，所以显示“.COM”等类文件时要准备得到“废物”）

在你的IBM个人计算机上试一下这些命令，然后继续读下一节。如果你只是使用该计算机，并不为它写程序，那么你可以跳过下一节。然而，如果你想成为DOS行家，那你就会对这里介绍的高级特点感兴趣。

高档的DOS操作

对一个非程序员来说，很多DOS操作并不是日常使用时所必需的。如，你并不关心如何构成BAT文件、复制系时磁道等等，你只简单地使用已随机器一起供应的程序。然而，偶尔你也可能要复制一个文件，或仅仅是想看一看如何使用某些命令。

首先你想试的大概是CTRL+PRTSC。送入

A>DIR (CTRL+PRTSC)

注意，通常在CRT上显示的输入和输出也连到你的打印机（若你有的话），因此，打印出来的磁盘机A的目录与显示的内容相同。

要关掉打印机可以再次发出CTRL+PRTSC。因而，CTRL+PRTSC是一只触发开关，用于开动和关闭打印机。这一特性与其它命令组合起来时是非常有用的。如，前面讨论的TYPE命令可以用来在CRT上和打印机上显示一个文件：

TYPE B:NOTE.TXT (CTRL+PRTSC)

现在，文件内容同时出现在屏幕上和系统打印机上，要关掉打印机就再次触动该键：CTRL+PRTSC。

在本章的其余部份我们将考察几种控制键与设备的有趣

的结合，它们对内行的DOS用户来说是非常有用的。

再谈COPY

前面我们已经学过，COPY是一种命令程序，它从源盘中把一个文件复制到目标盘或目标设备上。注意，在COPY主目中包括**设备**及磁盘文件名称。例如，我们可以用COPY把信息从磁盘文件复制到打印机。不必为此另写一个程序。设备的名称见下。当设备用这种方式命名时，我们称它们为**伪文件**，因为它们可以在命令中任何允许有文件名称的地方使用。

CON：操作台键盘和屏幕。如果键盘就是你想得到字符的设备，那么就把它作为输入伪文件名称；如果是屏幕，那就把它作为输出伪文件名称。记住，在结束CON输入时要用：CTRL+C或专用功能键F6，再接一个ENTER。

AUX：或COMI：通过电话线进行通讯的通讯适配器。它需要在主机箱中再加一块接口板。

LPI：或PRN：把打印机作为输出伪文件。

假如我们要在打印机上打印B：MAKE．TXT的内容。

COPY B：MAKE.TXT LPTI:

这一命令把存贮在磁盘机B上的文件内容传送到行式打印机。注意这一命令与TYPE命令加上CTRL+PRTSC控制键之间的区别。COPY在打印时并不同时在屏幕上显示文件。

从键盘到打印机

COPY命令可以用来直接在打印机上复制键盘输入，恰

如一台打字机：

COPY CON:LPTI:

在某一输入行的第一列送入CTRL+Z或F6键时，这一命令就结束了。

与TYPE命令一样，文件也可以显示在屏幕上：

COPY B:文件名.附加部份 CON:

从键盘到磁盘

最后，你可用COPY命令在磁盘上生成文件：

COPY CON:B:新文件.附加部份

这一命令把键盘处的全部输入都转移到磁盘机B中的软盘上。如果你以后想得到这一文件的硬拷贝，那么就再次使用COPY命令：

COPY B:新文件.TXT LPTI:

假如你想为自己写一条忘录，把它存入称为NOTE.TXT的文件中，那么就用COPY命令：

```
COPY CON:NOTE.TXT
This is a reminder to get groceries:
    Eggs
    Butter
    Milk (2% for Ted's diet)
    Soup stock
    Wine (for Sat. nite)
(F6, ENTER)
```

注意如何用F6功能键和ENTER键来结束输入。它必须在最后一行后另起一行的第一列。同样，记住每串最多只能有127个字符。如果你用ENTER键来结束一个字符串，那么

你就不必担心这一最大限量。然而，如果你把输入作为一个连续字符就来打，那么你在送入127个字符后就会遇到麻烦。这样，你就可以用ENTER来结束该字符串。

“COPY CON:文件名”的一种最普通用法是生成BAT文件。这是下一部份的题目。

批文件

BATCH（批）命令实际上是一条超级命令，它使你能一次执行一批命令。它的设计思想是建立一个命令文件，用“COPY CON:文件名.BAT”把它们存入磁盘。确认文件类型是BAT，因为只有它代表BATCH，而且只有DOS能以这种方式执行BAT文件。

当你想运行一整批命令时，只要输入文件名（不要使用附加的.BAT）就行了。例如，我们把格式化和生成系统软盘备用付本的命令组合在一起，称它为CREATE.BAT，用下列COPY录入：

```
COPY CON:CREATE.BAT
FORMAT B:/S
COPY A:B:
（F6）（ETER）
```

现在，每当我们要格式化并复制磁盘机A中某一磁盘时，我们把上述文件按下列所示交付给DOS：

```
CREATE
```

这一命令使得DOS先执行FORMAT命令，接着再执行COPY命令。如前所述，FORMAT和COPY命令需要用户提供另外的信息。

批文件参数

当执行BAT命令时，我们可以用已定义的有关参数来归纳批文件。批参数是一个整数，以百分符开始。如，这里所示的是一个参数化CREATE文件，这一命令的运行情况见屏幕显示3—17。

```
COPY CON:CREAE.BAT
TYPE %1:CREATE.BAT
%1: FORMAT %2:
DISKCOPY      %1: %2:
%1: CHKDSK    %2:
(F6)(ENTER)
```

现在，送入

```
CREATE A B
```

第一个参数（%1）定下A的值，第二个参数定下B的值。因此，DOS执行下面形式的CREATE.BAT：

```
TYPE A:CREATE.BAT
A: FORMAT B:
DISKCOPY A:B:
A: CHKDSK B:
```

这种形式的CREATE.BAT完成下列工作：

1、TYPE显示出一张CREATE.BAT文件目录表，它是固有命令（驻留在DOS中的机内的和内部的命令）。

2、FORMAT格式化一张空盘，该盘必须在磁盘机B中。

3、DISKCOPY把A:的内容传送到B:。

4、CHKDSK显示出磁盘机B中已占用的存贮量及剩

余的存贮量。

为了便于使用，我们甚至可以加上某些冗余参数，更加精巧地制作CREATE.BAT。假如我们用三个参数建立一个称为CREATE.BAT的文件（不用参数%2仅仅是为了便于理解批文件）。

```
COPY CON: CREATE.BAT
TYPE %3:CREATE.BAT
%3: FORMAT %1:
DISKCCPY %3:%1:
%3: CHKDSK %1:
```

现在，使用时要先规定哪一个是源磁盘机，哪一个是目标磁盘机。如：

```
CREATE B FROM A
```

这条命令使B成为A的备用付本，而

```
CREATE A FROM B
```

则使A成为B的备用付本。不过，请注意批文件中不使用参数%2（FROM），那仅仅是为了便于使用命令。

我们可以用这一设计思想为DOS制作某些扩充的命令。下面例子可以说明你能做些什么。

某些有用的批文件

假如我们想把一台磁盘机中的全部文件转移和重组到另一台磁盘机，但并不想去记如何使用TYPE。这样，设计一个下列转移命令文件XFER.BAT，我们就可完成这项工作。

```
COPY CON:XFER.BAT
COPY #3:*.*%1:
```

(F6)(ENTER)

然后，从一台磁盘机复制到另一台，

XFER B TO A

或者，

XFER A TO B

这条命令把一张软盘上的文件全部复制到另一张软盘。

如果记住命令或文件名有困难，那你就可以建造一个有意义的批文件来代替TYPE等命令。这里是某些设计思想。

通常，要在屏幕上显示一个文件，

SHOWME 文件名 ON 磁盘机

例如，

SHOWME TEXT.DOC ON A

这一命令的批文件至少包括下列内容：

TYPE %3:%1

再谈另一种设计思想。假如我们想在打印机上做同样的事。

PRINT 文件名 FROM 磁盘机

如，

PRINT TEXT.DOC FROM B

这一命令的批文件至少包括下列内容：

COPY %3:%1 LPT1:

PAUSE和REM

REM命令在批文件操作中用于在屏幕上显示一条注释或记事。同样，PAUSE命令也显示一条记事，但是它让计算机停下，等待再按某些键后才继续下去。这里是一个既包括记事又包括暂停的例子。请在你的IBM个人计算机上试一

这条下命令：

```
COPY CON: AUTOEXEC.BAT
REM 如不想改变日期和时间就请按ENTER键。
DATE
TIME
PAUSE 屏幕宽度改为40
MODE 40
DIR*. COM
PAUSE 屏幕宽度改回80
MODE 80
REM 现在已准备好了
(F6) (ENTER)
```

AUTOEXEC.BAT是一个非常特殊的批文件，它经常在系统复位（加电或CTRL+ALT+DEL）时执行。因而，你可制作AUTOEXEC文件，叫它做你想让它做的任何事情。例如，上述批文件越过标准的IBM加电再启动标识，作了下面几件事：

REN	显示上述记事
DATE	显示并允许你调整日期
TIME	显示并允许你调整时间
PAUSE	显示记事并等待你再按一个键
MODE	改变屏幕宽度
DIR	只显示目录“.COM”的文件

在每一条MODE命令后屏幕都被消除干净，因此，如上所示，PAUSE命令对我们来说也是必需的。请试一下这种使用方便的小型AUTOEXEC文件。

DOS 2.00的增强

如果你对XT感兴趣，你就必须使用DOS 2.00。〔注〕因为它采用了固定盘，DOS 1.00和1.10不能支持大容量的固定盘。

PC给你一种选择操作系统的余地。如，你可以使用DOS 1.10或DOS 2.00。DOS 1.10需要较小的主存贮器和较少的软盘空间。大多数为PC开发的软件都能在1.10上运行（如果有足够的存贮空间）。DOS 2.00需要较大的主存贮器和较多的软盘空间。与DOS 1.10相比，它在软盘上填有更多的信息（12%），但是如果你在软盘上存入系统文件，那么这一额外空间也就消耗殆尽了。此外，有些应用程序不能在DOS 2.00上运行。

DOS 2.00具有很多比DOS 1.10更复杂的特性。DOS 1.10最终会消失，DOS 2.00或以后的其它版本会取代早期版本，但是在那一天到来之前，你有选择的余地，可以选择具有较少特点的小型DOS 1.10或功能更强的DOS 2.00。

DOS 2.00的附加特性

配置文件。你可以生成一种特殊的个性文件，称为系统

注：其它操作系统也可在PC和XT上工作，但是它们不是本书的内容。Softech Microsystem公司的P系统是一种典型的可移值的操作系统，用UCSD Pascal程序设计语言。Digital Research公司的CP／M—86操作系统是另一种运行其它应用程序的操作系统。

配置文件。这一文件列出每当计算机再起动时建立的特殊文件名称和特殊参数值的表。软件开发者要用到这一文件，但是大多数XT或PC用户并不需要该配置文件。

固定盘的支持手段。DOS 2.00可以控制一个或多个固定盘。即，你可以把XT的存贮能力扩大到一千万、两千万甚至更多字符。然而，由于固定盘是不可拆卸的，你必须把固定盘上的重要数据复制到备用软盘上，以免因意外事件失去数据（ERASE命令、FORMAT命令或磁盘故障都可从固定盘中擦除文件）。

DOS 2.00含有固定盘与一张或更多张软盘间的备用命令和恢复命令。BACKUP（备用）命令把数据从固定盘复制到软盘。RESTORE（恢复）命令相反，把数据从软盘复制到固定盘。

增加的软盘容量。DOS 2.00格式化每一张软盘，所以每磁道有9个扇区，而不是8个，把每一软盘的容量从327680字符提高到了368640字符。如果你只把数据放在软盘上，那么整个空间都可用。但是，如果系统文件也在格式化的软盘上，那么增加的容量被增加的DOS 2.00系统文件所抵销。

多缓冲存贮器。增加主存贮器中可以同时容纳的缓冲存贮器的数量，这样你就可以提高某些程序的性能。DOS 2.00让你决定在主存贮器中同时保存多少文件缓冲存贮器，从而节约了从主存贮器到软盘间的交换时间。

分级目录。分级目录在一个树形结构中包含一个或多个子目录。这一特点使你能把文件划分成组，让机器运行得更快。如，你可以把全部字处理文件放在一个目录内，而把全

部展开单文件放在一个次级目录中。

磁盘卷标号。DOS 2.00允许你给软盘命名，这样就避免偶而发生的擦错文件和格式化时搞错软盘。

扩充的键盘和屏幕控制。DOS 2.00让你重新定义整个键盘，所以，你能把阿拉伯字符转换到希腊字符，等等。

输入／输出的重新定向。DOS 2.00提供一种把键盘和屏幕改成文件的方法，这样你就能把从键盘和屏幕输入输出重新定向为软盘文件输入输出。如，你想从运行程序向一个称为PAYROLL.DAT的文件发送全部输出，只要改变运行程序MYPROG的输出方向就行了：

A>MYPROG>PAYROL.DAT

管道。管道只是这样一种结构，通过二个或更多的链式程序，它能把一个程序的输出送入另一程序的输入。在DOS 2.00中你可以建造一条程序管道，使每一程序以前面一级程序的输出作为它的输入。这一特点在运行一长串程序时非常有用，不需人们干预。

DOS 2.00命令

在使用DOS 2.00支持的固定盘前，要用FDISK命令把它格式化。这一命令需要有一定数量的固定盘空间分配给DOS 2.00文件，然后它才能建立起供以后使用的磁盘。有关这一命令的例子请查阅DOS 2.00手册。

现在假定你想使用DOS 2.00中的新的树形结构文件系统。这一树的根是该分层结构中的最高级（如，军队中的上将）。假设下一级你想要二个子目录(如，军队中的少将)。让我们命名一个字处理的目录为WP，另一个数据库处理的

目录为DB。WP将掌握字处理文件的线索，DB将掌握数据库文件的线索。

此外，假定你想使用这两个目录及放在磁盘机C：上的它们的文件。如果注册磁盘机是A：，你可用下列方法建造子目录WP和DB：

```
A>MKDIR C:WP
A>MKDIR C:DB
```

第一条命令在磁盘机C：上生成称为WP的子目录。第二条命令在磁盘机C：上生成DB的其姐妹目录。如前面所讨论的，两个目录都是根目录的子目录。

如果你想节约目录WP中的信息，那么有两种办法“得到”该信息。首先，你能用CHDIR（改变目录）命令从根上改变目前的目录到WP：

```
A>CHDIR C:WP
```

全部后面的文件存取发生于目录WP之下。为了把目录复位到根级，可以用下列形式的CHDIR：

```
A>CHDIR
```

这一命令把根目录变为缺省目录。

现在让我们来说明第二种访问子目录WP下面文件的方法。这种方法需要有一条从根到你想要信息的**通路**。

假设你要用一个字处理程序来生成一个存贮在LETTER文件中的文献。LETTER可以用TYPE命令来显示，只有此时你才必须改变目录，或者给TYPE命令一条前进的通路：

```
A>TYPE C:\WP\LETTER
```

WP之前和之后的“黑斜线”字符指定了一条子目录前进的

通路。然后TYPE使存贮在磁盘机C：上WP目录下的、被称为LETTER的文件显示在屏幕上。

从根到任何文件的通路是由黑斜线分隔开的一连串子目录名。通常，你能在通路中规定一个无限长的序列。

擦除一个子目录。在擦除WP中全部文件后，你可以从子目录分级树中除去WP。RMDIR（除掉目录）命令能擦除一个子目录。

```
A>RMDIR C：WP
```

如果你使用MKDIR、CHDIR和RMDIR树命令，迟早你会在该树中迷路的。用TREE命令可以找到办法和显示树形结构外形。

```
A>TREE B：
```

这一命令可以显示磁盘机B：上目录的整个树形结构。然而，这种形式的TREE命令不能显示文件名称。如果还要看文件，那就要用到选加的／F。

```
A>TREE／F
```

这一命令显示树形结构和磁盘机A：上的全部文件（注册磁盘机）。如果没有假设缺省磁盘机，那么你就要规定你想显示哪一台磁盘机。

```
A>TREE／F B：
```

命名一张磁盘

DOS 2.00有一个可以命名软盘的选购件。这一选购件可以用来防止因误用磁给而失去数据。只要按下法使用任选／V就可以在格式化时盘定磁盘的名称：

```
A>FORMAT B：／V
```

当你要“卷识别号”时就送入任何至多由11个字母组成的名称。

常 见 问 题

问：什么是DOS？

答：DOS表示是磁盘操作系统，是IBM个人计算机使用的操作系统程序。

问：什么是瞬时程序？

答：在DOS中，“.COM”文件是程序，执行时要把它们从软盘复制到RAM。执行后，瞬时程序就从RAM中擦除。

问：在IBM个人计算机上，为什么有时NUM—LOCK键会与SHIFT键发生冲突？

答：不是真的冲突，只是锁住了数字小键盘，所以送进计算机的不是数字而是箭头和专用页控制字符，仅仅只要再按一次NUM—LOCK键它就回到了“数字状态”。

问：CRT屏幕尺寸多大？

答：缺省时为80列25行。但是你可以用DOS命令mode 40来改变它，然后再用一条mode 80把它改回来。

问：磁盘机怎么叫法？

答：左边的那个为A：，右边的为B：。

问：什么是光标寻址CRT？

答：阴极射线管（电视机屏幕）上的一条闪烁的字下划线，这条线指出了下一个输入字符的位置。

问：IBM个人计算机能接些什么设备？

答：一台打印机（并行或串行接口），一部带有通讯适配器的电话，一台带有彩色电视适配器的彩色电视，以及能用适配器物理地把它们与主机箱连接上的其它设备。

问：数字键（大写键）的作用是否与数字小键盘键一样？

答：是的，但是要正确开关NUM—LOCK。

问：诊断程序做些什么工作？

答：使你能测试计算机和外部设备。

问：如何格式化一张空软盘？

答：用DOS FORMAT命令。软盘在格式化前是不能用的。FORKAT程序预置软盘内容表等。

问：如何查明软盘的组织情况？

答：阅读一下“IBK磁盘操作系统手册（附录C）”，再执行DOS CHKDSK命令。磁盘各扇区“联在一起”组成文件，每一文件都有一个名称，该名称列在磁盘内容表中。每一扇区为512字符长，每张软盘有40×8＝320个扇区。

问：比较一张软盘的意思是什么？

答：从一张软盘（真正的拷贝）中向另一张软盘读信息，看看两张软盘存贮的信息是否相同，从而验证后者。如果两张盘上的信息不同，那么其中必有一张坏了，不能使用。

问：什么是“注册磁盘机”？

答：注册磁盘机是一台缺省磁盘机，如A或B。一般是A，但是可以通过送入B：来改变。

问：如何显示文件的内容？

答：送入“TYPE文件名”，例如，送入命令“TYPE B：

TEXT." 就可在屏幕上显示B：TEXT的内容。要暂时停住显示可以按CTRL+S或CTRL+NUM-LOCK键。CTRL+S是触发开关，而CTRL+NUM-LOCK可因打一个其它键被取消。

问：COPY命令做些什么事？

答：它是一个命令程序，用于把信息从一个文件或设备移到另一文件或设备。例如，COPY CON：B：NOTE.TXT是让你从键盘把正文送入文件B：NOTE.TXT。

问：什么是伪文件？

答：伪文件是一种似乎要成为一个文件的设备。例如，CON：是一种设备，当用于COPY命令时它又能作为磁盘文件名处理。

问：BATCH文件的作用是什么？

答：它用于在一条简写的命令中执行一批DOS命令。

问：文件名中的*有什么用？

答：它是"随意"的速记符号。因而，"*.*"的意思是全部种类的全部文件，"*.COM"意为全部COM文件，"TGL.*"意为TGL文件（不考虑类型）。

问：文件名中使用的"*"和"？"有什么区别？

答："*"用于文件名时指的是一个文件的整个名字和整个类型，而"？"在文件名中指的是单个字母。

问：如何打印文件的内容？

答：一种简单的办法是用伪文件PRN：或LPT1：作拷贝的目标。这样，COPY B：NOTE.TXT LPT1：将打印出B：NOTE.TXT的内容。

问：CTPL+PRTSC键在DOS中有什么作用？

答：随着你在屏幕上送入或显示信息，它打印出全部CRT屏幕字符。要使CTRL+PRTSC失效，你必须再次送入CTRL+PRTSC。

问：写入一个完成下列工作的“.BAT”文件：

MAKE NEW SYSTEM ON B（在B上建立一个新系统）

答：按下法建立一个称为MAKE.BAT的文件（用COPT CON：MAKE.BAT）。

```
TYT %4：
COPY A：*.COM %4
（CTRL+Z）（ENTER）
```

现在这一命令已存入注册软盘，所以每当你打入命令

MAKE NEW SYSTEM ON B

它就执行等于B的%4的MAKE.BAT批文件。

问：如何把AUTOEXEC.BAT重新命名为一个简单名称，如DO.BAT？

答：用RENAME AUTOEXEC.BAT DO.BAT。

问：使用IBM个人计算机时首先要做的是什么事？

答：制作全部软件的备用付本；记下受保护软件的制造厂商，以便用较少的钱买到备用付本。

第 四 章

如何进行字处理

文字，人们都认识它，知道它是有用途的。人人都使用文字，有时它令人可怕，有时令人欣慰，但是人们总是离不开它。

于是，文字逐渐地变成了上帝，人们成了它的奴隶、它主宰人们生活中的一切。

但是，文字还在继续发展，它耗费了人们大量的精力而成为祸根。因而，人们把它禁固在书写的页面上，可是诡诈的文字把人们弄得头晕目眩、无能为力并陷入混乱的困境之中。

这时，文字处理机来到了！

IBM个人计算机作为电子打字机

利用象EasyWriter这样的字处理程序，你可以把一台IBM个人计算机变成电子打字机。先把EasyWriter程序装入RAM，然后按照本章中所讨论的指令工作便可实现这一转变。

文字处理程序使你能在软盘文件中送入文字、建立页边、保存和恢复文献(documents)以及控制打印质量（打印质量取决于打印机的性能）。

开始前，一定要有一张DOS格式化的软盘，用它保存你要生成的文本，这就是文献软盘或称存贮软盘。

另外，为防止因软盘的磨损或事故而破坏了Easy-Writer程序本身，你应该复制EasyWriter的软盘付本。其复制过程如下：

复制一个后备付本

假设计算机供应商没有对你的EasyWriter软盘进行“标注”（“PersonaLized”），为使它能在IBM计算机上运行，那么请你花几分钟的时间对你的EasyWriter软盘进行“标注”

为了“标注”你的EasyWriter软盘，需要做下面的工作：假设在磁盘机A中装有一个DOS软盘，在磁盘机B中装有EasyWriter软盘，那么你必须把A中的DOS软盘的系统磁道复制到磁盘机B中的EasyWriter软盘上。

SYS B:

完成了这个“标注”步骤后，EasyWriter软盘就包括了DOS系统磁道，此时你将得到如下的信息：

System transferred

（系统已传送）

格式化存贮软盘

如果你还没有复制EasyWriter程序软盘，那么现在你应该做这一工作了。关于复制后备付本的详细情况，请参阅第三章。你必须要在一张已具有系统的DOS格式化软盘上复制程序的后备付本。

FORMAT B:/S

现在，把EasyWriter的后备付本作为程序软盘放入磁盘机A中，用“加电”或CTRL+ALT+DEL重新启动这个系统部件。先把EasyWriter程序装入RAM，然后再执行。屏幕将出现如屏幕显示4—1所示的内容。把格式化的存贮软盘插入磁盘机A中（卸下程序软盘），接着按下ENTER键。

EasyWriter存贮软盘必须用EasyWriter识别的代码格式化。为了准备供EasyWriter使用，你必须再次格式化DOS软盘。但是，格式化会破坏一个软盘上的全部信息，因此，只能进行一次格式化操作！

在你按下ENTER键之后，屏幕则显示4—2所示的菜单。至于这个菜单我们以后再作解释。要格式化存贮软盘就需输入Y两次。完成这种格式化操作须花20～30秒。

在输入一份文献时，如果你决定要使用两个软盘的话，那么，你就还要格式化磁盘机B中的第二个软盘。这是一个很好的想法，因为你可以把第二个软盘用作第一个软盘的后备付本。这样每份文献都保存在两个软盘上。

为了格式化第二个软盘，当处于屏幕显示4—3所示的命令方式时，输入一个2就可选择磁盘机B。这就把“注册磁盘机”转变为B了。其次，你还可以用初始化A中软盘的方法对磁盘机B中的软盘进行初始化。

小心！ 如果EasyWriter没有在磁盘机B中找到DOS格式化软盘，那它就会转到磁盘机A。如果不特别地提防这个问题，你非但没有把磁盘机B中的软盘格式化，反而会无意中从磁盘机A擦去了你的全部文献！

一旦你已格式化一张或一张以上的存贮软盘，就可以开

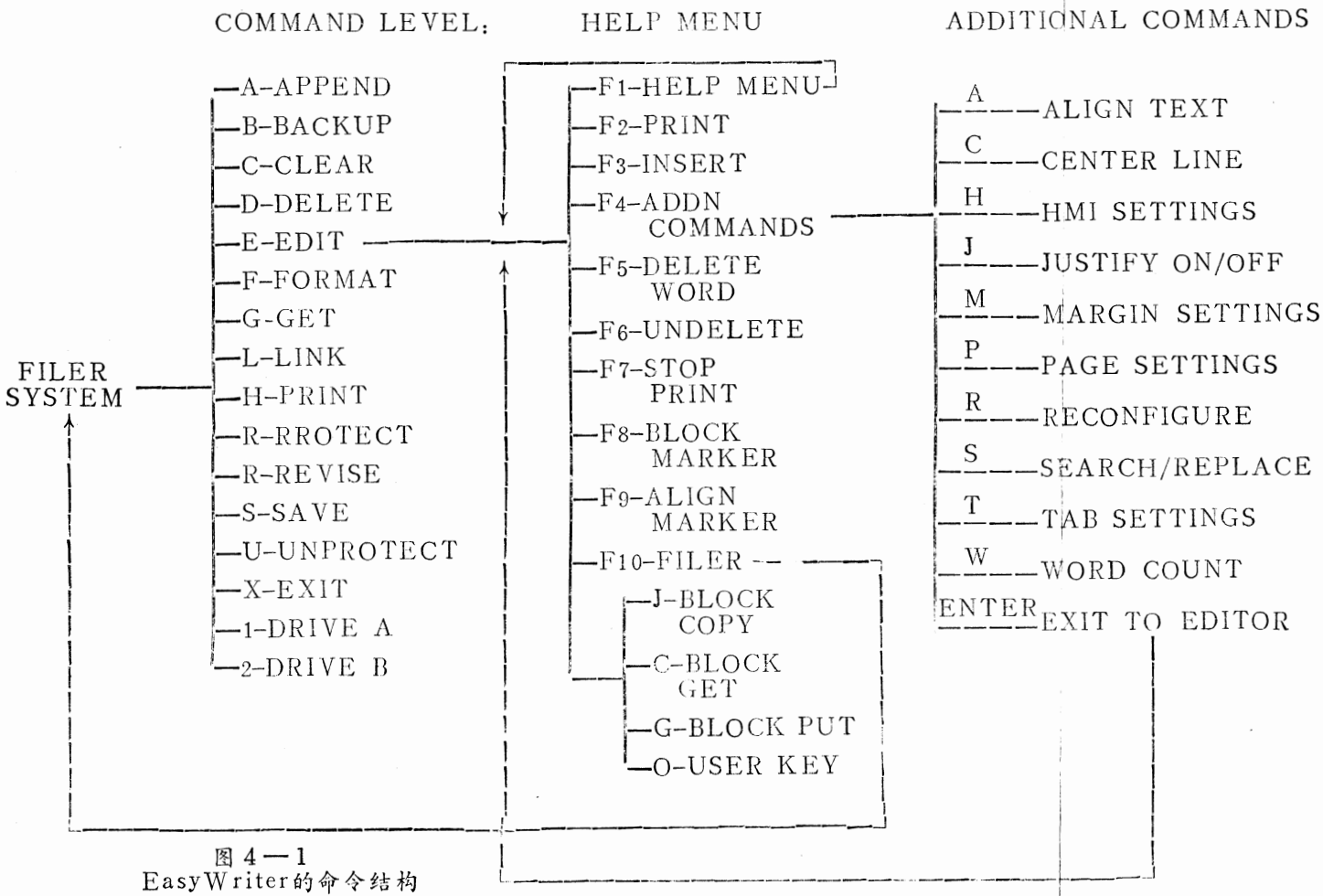


图 4—1
EasyWriter的命令结构

始使用EasyWriter。注意，存贮软盘都可以使用这两个磁盘机。EasyWriter程序完全脱离RAM运行。

```
IBM
Personal Computer

Easywriter
Version 1.00
(c) Copyright IBM Corp. 1981
(c) Copyright IUS, Inc. 1980

Insert storage diskette in Drive A,
then press ENTER
```

屏幕显示4—1　　启动EasyWriter

```
EASYWRITER FILE SYSTEM
----------------------------------------
A—APPEND FILE      E—EDIT FILE
B—BACKUP           F—FORMAT DISK
C—CLEAR TEXT       G—GET A FILE
D—DELETE           L—LINK FILES
H—PRINT FILE       U—UNPROTECT
P—PROTECT FILE     X—EXIT
R—REVISE A FILE    1—DRIVE A
S—SAVE FILE        2—DRIVE B
----------------------------------------
DISKETTE NOT INITIALIZED
DO YOU WISH TO FORMAT?
```

屏幕显示4—2　　格式化一个存贮软盘

```
EASYWRITER FILE SYSTEM
-----------------------------
A-APPEND FILE      E-EDIT FILE
B-BACKUP           F-FORMAT DISK
C-CLEAR TEXT       G-GET A FILE
D-DELETE FILE      L-LINK FILES
H-PRINT FILE       U-UNPROTECT
R-PROTECT FILE     X-EXIT
P-REVISE A FILE    1-DRIVE A
S-SAVE FILE        2-DRIVE B
-----------------------------
NO FILE   FILESIZE= 1   AVAIL=18559
          %USED=0   DRIVE A
NO LINKS

COMMAND:
```

屏幕显示4—3　　命令方式

EasyWriter的命令结构

下一节我们将对EasyWriter命令加以概述。但是在开始使用这个程序前，先理解这个“大画面”是很有帮助的。所以，让我们回到表示EasyWriter命令级的屏幕显示4—3。

屏幕显示4—3表示一个三级命令结构的“最上一级”。你可以输入一个命令（一个单一字母后跟ENTER），並且你可以执行相同的操作，也可以“下降”到另一个命令级上。

例如，G命令从软盘上得到一个文件。H命令打印现行

文件。但是，在打印一个文件前，你必须先得到它。

EasyWriter的三级命令结构如图4—1所示。在该命令结构的“顶部”（图4—1的最左列）是我们称作“文件管理”（“Filer”）的文件系统菜单。文件管理的主要用途是在RAM和存贮软盘间传送文件。

E命令可以撤消文件管理菜单而由编辑程序接替。在编辑状态时，如果不按下功能键F1，你就得不到如屏幕显示4—4所示的HELP MENU。所以要从文件管理级中得到屏幕显示4—4，你必须输入E，紧接着再输入F1。

正如你从屏幕显示4—4中所看到的，编辑程序被用于在文献中插入、删除、标记和向各处传送文本。我们将在后面的一些例子中更详细地解释这个菜单。现在请你记住菜单是字处理程序的核心问题。

附加命令示于屏幕显示4—5。在处于编辑程序级时，按下功能键F4，即可得到这些附加命令。这个菜单也相当于在图4—1所示的命令结构的第三级。或许可以这样说，最重要的附加命令是A—ALIGN和J—JUSTIFY命令。ALIGN执行段的对齐，JUSAIFY是调平右页边的转换开关。

```
                    EASYWRITER HELP MENU
-
--------------------------------------------------------------

  F1-HELP MENU      F2 -PRINT                 J -BLOCK COPY
  F3-INSERT LINE    F4 -ADDN COMMANDS         C -BLOCK GET
  F5-DELETE WORD    F6 -UNDELETE              G -BLOCK PUT
  F7-STOP PRINT     F8 -BLOCK MARKER          O-USER KEY
  F9-ALIGN MARKER F10-FILING SYSTEM

L                                   R
+-----+-----+-----+-----+-----+-----+-----+---
```

屏幕显示4—4　　编辑程序的求助菜单

```
                    ADDITIONAL  COMMANDS
-------------------------------------------------------------

A-ALIGN TEXT      M-MARGIN SETTINGS    T-TAB SETTINGS
C-CENTER A LINE   P-PAGE SETTINGS      W-WORD COUNT
H-HMI SETTINGS    R-RECONFIGURE        ENTER-EXIT TO EDITOR
J-JUSTIFY ON/OFF  S-SEARCH AND REPLACE

COMMAND?
L                                                        R
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----
```

屏幕显示4—5　编辑程序的附加命令

图4—1表示如何一次返回到上一级上。ENTER键导致退出附加命令级，于是你可以返回到编辑程序。按下功能键F10则退出编辑程序而返回到文件管理级。按下功能键X，则终止文件管理级，也使EasyWriter终止。

现在你理解了这个大画面，我们就可以总结命令结构中每个命令的含义了。这些命令能使你输入、插入、删除、传送、复制、改变、存贮、恢复和修改新的以及已存在的文献。

EasyWriter命令概要

在说明EasyWriter的用法以前，我们需要把这些命令列成表并给它们的应用下一个简要的定义。这些表可供以后参考，同时也作为应用它们的指南。现将图4—1命令结构中的每一级解释如下：

文件系统（屏幕显示4—3）

A　通过把第二个文件传送到RAM中的方法而增补一个文件。第二个文件增补到RAM中现行文件的末尾。如果你没执行R—REVISE，软盘上的文件就不改变。

B　把磁盘机A中软盘上的全部内容复制到磁盘机B中的软盘上。磁盘机B中软盘上以前的信息全部丢失。

C　清除RAM区的所有文本信息。该命令从RAM中擦去一个文献，但并不改变存贮在软盘上的那

些文献。

D　删除软盘的一个文件。即擦去软盘上一个文献。

E　编辑现行RAM文件。如果没有现行文件存在，屏幕呈现空白并等待你输入文本，即怎样开始。

F　格式化或初始化供EasyWriter使用的存贮软盘。这个软盘必须是已DOS格式化的。要提防未格式化的软盘！

G　从软盘中得到一个编号文件并把它装入RAM中，使它成为现行文献。

L　把两个或两个以上的文件连接在一起，这样，即使文献没有放在RAM中你仍可以打印，检索和置换一个以上的文献。该命令与A—APPEND不同，不是要增补文件而是把软盘上的文件连接在一起。

H　打印现行文件和同它按L—LINK命令连接的所有其它文件。你也可以利用编辑程序级中的打印命令F2。

P,U　保护和不保护一个文件。一个被保护的文件是不能删除或修改的。

R　修改一个文件。重写包含要修改文献的软盘文件，从而达到保存现行文献的目的。如果你不修改文件，那么一个新的软盘文件就以相同的名字，不同的编号建立起来。

S　保存一个文件。用唯一的编号（1—31）建立一个新的文件，把RAM的内容写到这个文件上。如果你要改变软盘上的现有文件，则使用

EASYWRITER FILE SYSTEM

```
A—APPEND FILE  E—EDIT FILE     H—PRINT FILE    U—UNPROTECT
B—BACKUP       F—FORMAT DISK   P—PROTECT FILE  X—EXIT
C—CLEAR TEXT   G—GET A FILE    R—REVISE A FILE 1—DRIVE A
D—DELETE FILE  L—LINK FILES    S—SAVE FILE     2—DRIVE B
```

FILE#: 2 chap2 FILESIZE=3077 AVAIL=15483 %USED=85 DRIVE B
LINKS ARE: 2 4 5 6 7 8 9 10 11 12 13

```
 1 heading    95 2 chap2    3077 3 chap1.5 8864 4 chap2.1 4602
 5 chap2.2  2396 6 chap2.3 1959 7 chap2.4 1844 8 chap2.5 2071
 9 chap2.6  1764 10 chap2.7 1831 11 chap2.8 6171 12 chap2.9 2985
13 chap2.10 1213 14 chap3   2173 15 chap3.1 2430 16 chap3.2 2574
17 chap3.3  4183 18 chap3.4 2627 19 chap3.5 1966 20 chap3.6 2020
21 chap3.7  3297 22 chap3.8 3129 23 chap4   3054 24 chap4.1 2663
25 chap4.2  3557 26 chap4.3 2657 27 chap4.4 2838 28 chap4.5 3432
29 chap4.6  2876 30 letter  1797 31 letter2  2400
```

COMMAND:

屏幕透示4—6　由空格键所得到的文件管理状态

R—REVISE。

X　退出EasyWriter。该命令使EasyWriter终止，这样你可以返回到DOS。

1,2　选择缺省软盘磁盘机A或B。如果在A或B中的软盘不是一个DOS格式化的，则该命令失效。

正如你所看到的，文件管理命令用来建立一个现行文件、传送文件和使文件无效。每个文献都被赋于一个名字和一个编号。编号是唯一的（1—31），但名字可能不是唯一的。为此，当引用软盘文件时，EasyWriter宁可使用编号也不愿使用名字。

在文件管理状态中，你在任何时候按下空格杆都可以显示如屏幕显示4—6所示的文件名字和编号以及其它重要的信息。

在屏幕显示4—6中，现行文件用FILE#2 Chap2指示。该文件存放在RAM 中並占有存贮器3077个字符，从而剩下15,483个字符的RAM空间。把这些文件和它们的字符长度一起列成表。%USED=85告诉你在磁盘机B中的软盘上已用了85%，在本软盘上只有15%的可用空间。

LINK连接信息包含一个连接在一起的文件编号表。如在屏幕显示4—6中所示的文件编号2、4、5、6、7、8、9、10、11、12、13。这时所发出的任意打印或检索命令都应用于所有这些文件。此外，这些打印和检索命令也适用于如上所示的相同序号的那些文件。

磁盘机B中软盘上的所有文件的编号、名字、大小都显示出来了。注意，文件编号是唯一的，但名字不必是唯一的。所有EasyWriter命令都用文件的编号查找文件。

现在，假设你选择E—EDIT FILE命令，接着按下功能键F1，这就把你置于EDITOR的HELP MENU方式下工作。

编辑程序的求助菜单（屏幕显示4—4）

当处在编辑状态时，按下F1便出现HELP MENU。再次按下F1，该菜单便消失了。不管这个菜单是否出现，你都可以进行编辑。另外，你可以利用箭头键和CTRL键控制屏幕。

在屏幕的原有字符上打入一字符可以把原字符顶替掉。你还可以按下面的介绍作插入或删除。

F1—HELP	显示HELP MENU
F2—PRINT	打印文献。
F3—INSERT	插入一个空白行。插入正文的方法比利用INS键快，因为每当你把另一个字符打入该文献里时，不需要传送字符。
F4—ADDN	附加命令。请看下一小节的附加命令概要。
F5—DELETE	删除紧靠光标右边的一个字。这比重复利用DEL键快。
F6—UNDELETE	这个命令可使以前被删除的正文一次一个字符地恢复。它仅在你删除一个字或字符之后，没有移动光标之前才起作用。
F7—STOP	当打印机正在打印时，若按下

F7—STOP，则迫使打印机停机并放弃打印命令。

F8—BLOCK MARKER　标记文献内你要复制或传送一段正文的起止位置。该命令和CTRL+J，CTRL+C及CTRL+G一起使用。

F9—ALIGN MARKER　利用它来保护一段正文，不受ALIGN命令的作用。它避免一段正文的对齐。

F10—FILER　返回到文件管理级。

I，C，G　和CTRL键一起用来控制“剪裁和拼贴”块传送。利用F8和J，C，G命令可以把一段正文从一个地方复制到另一个地方。

Q　由CTRL+Q加上一个用户定义的代码（通常用ASCII）启动用户定义键。

注意：在屏幕显示4—4中已经建立了左、右页边标志符。L标志左页边对齐，R标志右页边对齐。利用F4—ADDN COMMAND可以改变左、右页边。

附加命令（屏幕显示4—5）

图4—1所示的命令结构中的附加命令是最低一级的。借助于建立页边、传送正文以及对连接的文件执行检索和置换操作，你可以在这一级中格式化文献。

A—ALIGN　从当前所显示的页面开始至该文

	献的最后一个页面为止对齐正文。用J—JUSTIFY，M—MARGIN和F9—ALIGN MARKER命令来控制被对齐的正文格式。
C—CENTER	水平地移动一条线直至可以把它放在中心。光标可以置于该线的任一地方。
H—HMI	在IBM个人计算机上不使用H—HMI。
J—JUSTIFY	每当你使用J—JUSTIFY时，转换开关就拨向ON／OFF，当拨向ON时，正文的左、右页边保持均匀。
M—MARGIN	建立左、右和段落缩进的页边。
P—PAGE	设定起始页面的编号，以便打印时编页码。
R-RECONFIGURE	重新配置程序软盘的缺省设置。它用于重置约定页边，调整转换开关，标记、页面行数，行间的空白、打印机型号和打印机接口板。
S—SEARCH	检索和置换正文，检索一个字符，字或片语和用某一其它的方式置换正文。它处理连接的文件。
T—TAB	置标记空格。

W—WORD	告诉你在现行文献中有多少字。
ENTER—EXIT	按下ENTER键退出本级并返回到编辑程序。

除了这些菜单命令外，你还可以在文献本身里嵌入一些“打点命令”。所谓打点命令是用于控制一个文献的打印。一个打点命令必须是出现在一行上的第一个片语。Easy-Writer 打点命令提要如下：

打点命令

全部打点命令都必须从第一列开始，而且每行包含一个单一的打点命令。所有单打点命令必须以一段的终止标志符（使用ENTER键）结束。全部的打点命令行间不能有空行。

打点命令可以出现在文献的任一地方。一旦启动，它们就保持有效直到另一打点命令强使其改变。

· EJECT	页面逐出。转到下一个页面的顶部。
· EJECTnn	若从现行页面倒数只有nn行或少于nn行，则逐出页面。
· EOL	规定打字机使用的行终止字符。
· FORMSTOP · FORMSTOPOFF	使用这二个打点命令可以一打次印一张数据记录纸。在每个页面的结束处打印机停机。要继续打印，请按下空格杆。用这个方法你可以一次向打印机插入一张数据记录纸。

·LINEnn	置每页打印的行数为nn，缺省值nn=54。
·MARGINnn	置左页边为nn列。该设置只影响打印，而不影响编辑。
·PAGErr, cc	在每一打印页面的第rr行，第cc列打上页码。
·PAGELINESnn	每页的行数。每页的缺省值nn=66行。
SPACEnn	设定行间空格。若nn=0，则该打印页是隔行的，若nn=1，则打印页是隔二行的，依次类推。
TITLEA, nn, text TITLEB, nn, text TITLEC, nn, text	设定在每页的第nn行上最多打印三个题头，而且题头是作为正文给出的。
TOPnn	设定每页顶部空格的数目。从nn+1行开始打印。
USER	规定ASCII编码字符。例如，USER#96把#符号定义为你从ASCII 96中得到的字符。在这种情况下，#表示打印一个撇号。

这些打点命令都要送入文件之中，其方法与送入文献一样。然而，大部分打点命令都出现在一个文献的开头。下一节所列举的例子将对此加以说明。

用EasyWriter准备文献

为了说明如何用EasyWriter输入和编辑一个文献，我们用图4—2、图4—2（a）所示的例子来说明文献打入计算机时是什么样的。图4—2（b）所示为打印出来的同一个文献。

在图4—2（a）中页面顶部的打点命令控制EasyWriter在打印时做以下的工作：

top2　　在每页的顶部空二行。

Lines64　　每页最多打印64行。

titles,3　　在每页的顶部打印"Treatise on Goubers"

space1　　在打印的输出中行间须留一个空白行。

你可以把打点命令得到的结果同图4—2（b）所示的打印结果进行比较。请注意，该页是隔二行的，另外还要注意这些段落没有缩进。只有向你说清如何准备文献才能解释清这些情况。

文献输入例子

用输入下面的命令和正文的方法准备图4—2的文献。如上一节所述，首先启动EasyWriter程序，达到命令级后打入一个E进入编辑程序，EasyWriter答以PLEASE STAND BY（请准备好）。

```
E (ENTER)
PLEASE  STAND BY
.top2
```

.lines64
.titlea,3, Treatise on Goubers
.spacel
1.1 The Average Gouber.
The average gouber is computed by summing over the meso-nerdic space of micro-goubers as follow:

Avg=Sum (meso-nerdics)

This expression is applied to the meso-nerdics that are easily observed in furd reactions. However, the measurements must be taken quickly, as every meso-nerdic lasts for 3 nano-seconds.

图 4—2 (a) 文献输入

Treatise on Gouber.

1.1 The Average Goubers.
The average gouber is computed by summing over the meso-nerdic space of micro-goubers as fo-llows:

Avg=Sum (meso-nerdics)

This expression is applied to the meso-nerdics that are easily observed in furd reactions. However, the measurements must be taken quickly, as every meso-nerdic lasts for 3 nano-seconds.

图 4—2 (b) 文献打印

过一会儿，屏幕被清除，光标出现在屏幕的顶部。Easy-Writer等待着你按下功能键或开始输入正文。

假设为了得到HELP MENU，我们按下F1。现在我们希望做的是建立屏幕的页边和把转换开关拨到“接通”。于是，按下F4。

F4

M

页边命令使EasyWriter请求三个加有L、R、I标记的数字。第1个数字L是左页边的列数。第2数字R是右页边的列数。I是缩进列数。

0，60，5（ENTER）

它表示左页边为0列，右页边为60列，段落缩进5列。

下一步，送入J，J-JUSTIFY转换开关接通。

J（ENTER）

它会在字间安排适当的空格，使右页边取齐。最后，按下ENTER键，我们就退出了编辑程序中的HELP MENU级。

ENTER

菜单撤去，屏幕又返回到我们退出前的位置。于是，如图4—2（a）所示的正文就被输入进去了。

然而，图4—2（a）没有说明全部情况，请注意，正文中的平均公式被缩进在该页面的中间。这里说一下它是如何做到这一点的。

尽管J—JUSTIFY转换开关是接通的，F9键却能保护一段正文不作自动调整和向左对齐。平均公式的上一行包含一个F9符号（在IBM屏幕上显示出一付令人满意的画面），

并且在这个被保护的公式的下一行中也包含一个F9符号。现在，该公式驻留在我们所放的位置，它与页边设置情况无关。

要使这一行（包含AVG那一行）居中，用箭头键把光标移到该行上的任一字符。先按下F4，然后再按下C，这样便可进入编辑程序级。

F4
C（ENTER）
ENTER

为了终止编辑程序级，必需有第二个ENTER。

最后，我们可以存入这个文献了。先按下F10，接着再执行S—SAVE命令就可完成这项工作。

F10
S（ENTER）

该文献被保存在你提供的一个名字下。如果要在以后某一时刻检索该文献，你可以用G—GET命令和该文件的编号：

G（ENTER）
1（ENTER）

这一命令检索了软盘上的第一个文件。

你可能已经注意：图4—2（b）所示的打印输出看来不象你期望的那样。其中之一就是段落没有缩进或右页边也不齐，即没有一个“向右取齐”的页边。

如果要得到一个如图4—3所示的对齐文献，那么，在打印一个文献之前，你必须使用A—ALIGN命令。要做到这一点就必须在ADDITINONAL COMMAND级中输入

A（ENTER）：

A（ENTER）

ALIGNING

如果不在这个合适的级上，请使用F4，以达到该级。

要完成这个对齐操作应花一点时间。每一行都要根据页边设置和JUSTIFY转换开关进行调整。在图4—3中每一段都是缩排的，而且按要求向左、右取齐（请记住设定值是0,60,5）。AVG公式不能撤消和再调用，因为它受F9对齐符保护。

除了在本例中给出的打点命令外，我们还可使用下述的命令：

EJECT 3　　若在倒数3行范围内，则逐出现行页。

MARGIN 10　把每一个字符（向右）移10列。

PAGE 2,3　　在第2行第3列编页码。

正如本例所示，这些命令可以置于该文献的顶部。

现在我们回过头来进一步举例说明编辑程序：一旦文字进入EasyWriter文献文件内，编辑程序是如何处理这些字的。

用EasyWriter编辑文献

假设我们用图4—2中所示的同一文献作为要编辑的文献的样本。首先我们必须用文件管理命令G把文献放在RAM中：

G（ENTER）

I（ENTER）

在这里，我们已经假设把这个文献存贮在第一个文件里。

Treatise on Goubers

1.1 The Average Gouder.

The average gouber is computed by summing over the meso-nerdic space of micro-goubers as follows:

Avg=Sum (meso-nerdics)

This expression is applied to the meso-nerdics that are easily observed in furd reaction. However. the measurements must be taken quickly, as every meso-nerdic lasts for 3 nano-seconds.

图 4—3　　图 4—2 的对齐正文

把第 1 个文件里的文献装入RAM，用E命令进入编辑程序级：

E (ENTER)

这时，我们得到了如图 4—2 (a) 所示的显示屏幕。

控制文献

借助于移动屏幕上的光标及其它方法，下面这些控制键可以用于控制一个文献。

Arrows	4 个箭头方向不同的键可以使光标上、下、左、右移动，一直移到你希望的文本位置。
Backspace	擦除紧靠光标左边的一个字符。
CTRL+ARROW	跳过左边或右边的字。

HOME	光标移到页面的顶部。
PGDN,PGUP	向上/向下移动整个显示屏幕。
CTRL+PGUP	移到文献的开头部分。
END	移到文献的结束部分。
TABS	移到预置的标记停机处。
CTRL+END	删除到行末。
CTRL+Q	用户可定义的打印机命令。
DEL	删除紧靠光标右边的一个字符。
F3	在光标的下面插入一空行。
F5	删除光标右边的一个整字。
F6	恢复以前被删除的字符。只有当还没有移动光标时，它才起作用。
F8	被移动的一段正文的起始和终止标志符。
F9	对齐保护标志符，前例已作说明。
INS	在光标指示的地方送入插入方式。
ENTER	退出插入方式。

这些键可以用来控制整个屏幕的正文，然而，你经常要进行全局检索和置换正文。要做到这一点，你可以用屏幕显示4—5 ADDITIONAL COMMANDS（附加命令）菜单中指示的SEARCH AND REPLACE（检索和置换）命令。

检索和置换

假设你要用字“hexglub”和“hexglubs”置换图4—2中出现的每个“gouber”和“goubers”，你可以执行如下的命令：

F4
S

这两条命令使你下降到ADDITIONAL COMMAND级并启动编辑程序的SEARCH AND REPLACE命令。

```
SEARCH WORD:      Gouber
REPLACE WITH:     Hexglub
ALL OR SOME ?       A
```

它将检索全部的“Gouber”并代之以“Hexglub”。注意小写的“gouber”没有被置换。如果还要置换小写方式，那就再次执行S命令：

```
SEARCH WORD:    gouber
REPLACE WITH:   hexglub
ALL OR SOME?      A
DONE !!
```

发生的这些变化都显示在屏幕上。

再者，假设你要用字“a”置换某些“the”字。为此，输入S而不是A：

```
SEARCH WORD:      the
REPLACE WITH:      a
ALL OR SOME?       S
TYPE:     K—KEEP,  D—DELETE,
```

R—REPLACE, X—EXIT

如果你用K响应，则跳过这个置换，用D响应，则检索字被删除，用R响应，则检索字被置换，用X响应，则放弃检索命令。

如果你对一批连接的文件执行一个检索和置换命令，选择SOME操作时，所有的文件都要执行这些相同的操作。此外，你可以决定当扫描这些连接文件时是否要保存逐个文件的变化。

块编辑

块编辑是把一个完整的正文块从你的文献中的某一地方传送到另一个地方的过程。一个正文块实际上是你要处理的一组字。它可以是一个或一个以上的句子、段，甚至是若干个页面。

在EasyWriter中进行块传送时，你应关注以下三件事情：

1、由F8块标志符指定被传送的块。

2、你要传送的块到达的那个地方即目的。它可以是文献中任何你要插入块的地方。

3、在块传送过程中，有一个用作暂时保存该块的3500字符容量的字符缓冲区。

把块标志符F8放在被传送块的开始和末尾，再把这个块传送到缓冲器，然后再从缓冲器传送到转移的目的地，从而实现上述三项操作。因此在编辑程序的HELP MENU中你可使用三条命令：

CTRL+J　　块复制

CTRL+C	块获得
CTRL+G	块放置

CTRL+J命令实际上是一个开关。当你把开关接通时，它使你对一个块复制多份付本。当把它断开时，你只能把一个块从一个地方传送到另一个地方。在下面的例子中，我们将使用这个转换开关来复制一个块。

CTRL+C命令能把一个加标志的块传送到缓冲器里，但是这个加标志的块会从该文献中擦除。为了防止它被擦去，应使CTRL+J接通。若CTRL+J接通，则你可执行一条CTRL+G命令，于是又恢复了被擦去的块。

CTRL+G命令可以把缓冲器里的块插入到文献中由光标指定的目的地。若CTRL+J接通，则建立缓冲器的付本。

这样看起来似乎颇为混乱，为此我们先列出块传送的步骤，然后再列出块复制的步骤：

块传送（BLOCK MOVE）

1、块的起始地方必须用一个块标志符标志。块标志符必须单独地列一行。把光标移到该块前一行的空行上。如果没空行，则用F3建立一个空行。

2、把F8插到被传送块前一行的空行里。完成这样一个步骤须利用INS+F8+ENTER+ENTER。注意需要二个ENTER，其一用来终止INS，另一个用来终止含有该标志符的那一行。F8标志符作为一个“箭头”符号出现。

3、F8标志符还插入到被传送块最后一行的后面。它规定块在何处结束。

4、把光标移回到起始块标志符F8的前一行上。当你

完成一个BLOCK GET：

CTRL+C

该块就被传送到缓冲器。这时，你就会看到这个块消失了。因为它已经被传送缓冲器了。

5、现在，把光标移到文献中你要插入块的地方。用CTRL+G命令可以把块从缓冲器传送到文献中去。于是，这个块就出现在正文中。

6、两个F8块标志符将保留在正文中，因此，把光标移到这二个标志符处，并删除它们。

注意：BLOCK MOVE过程破坏性地把一个块从文献中的一个地方传送到另一个地方。如果你只想复制这个块，那么你必须对上述的过程作如下的修改：

块复制

采用块传送中的1—3步，但用下述的几个步骤代替上述的第四步：

4、在第4步中，你要用CTRL+C命令把块复制到缓冲器。但是为了重现这个块，你必须做下列几步：

4（a）、在你把光标移到块的起始部分之后、完成第4步的CTRL+C命令以前，用CTRL+J命令接通块复制开关。

4（b）、用BLOCK COPY ON（块复制接通）完成下列的工作。用CTRL+G命令。把该块传送到缓冲器，然后在不移动光标的情况下，执行一条CTRL+G命令。这将使被复制的块正好返回到它以前所在位置。可是在这个缓冲器里还有该块的一个付本。

4（c）重复BLOCK MOVE过程第5和第6步。无论你把光标移到哪里并执行一条CTRL+G命令，都会出现一个付本。

CTRL+J命令断开BLOCK COPY开关，使BLOCK COPY过程终止。当然你也应擦除F8块标志符。

让我们概括一下块传送和块复制的区别：

BLOCK MOVE　CTRL+C继之以CTRL+G

BLOCK COPY　CTRL+J,CTRL+C,CTRL+G
继之以CTRL+G，CTRL+J。

重复CTRL+G命令，你可以按需复制多份付本。

有时，缓冲器区还含有以前块复制操作时遗留下来的“无用数据”。如果出现这种情况，只要拨动开关两次，你就可以“刷新”这个缓冲区：

CTRL+J
CTRL+J

结果清除了缓冲区，于是不需要的正文就不会混入你的块传送和/或块复制中去了。

常见问题

问：你必须格式化多少次EasyWriter软盘？

答：二次。一次使它和DOS一起工作，另一次使它和EasyWriter程序一起工作。

问：当使用EasyWriter程序时，有必要把EasyWriter软盘留在一个磁盘机里吗？

答：没有必要。只要你不再重新配置这个程序，你可以

取下程序软盘。

问：如果你想使用一张未格式化的软盘，会出现些什么情况？

答：有时对你要进行的操作无反应，有时会丢失有价值的信息。

问：你应该在什么时候使用A—ALIGN？

答：通常在你已把所有的正文输入之后，并且要使页边取齐。由于取齐后的结果可能不是你期望的结果，所以你应该尽可能地保存原来的文献。

问：为什么A—ALIGN会给出意想不到的重新排列？

答：请记住A—ALIGN能使所有的页边取齐。但是，你要特别地缩进一个正文块，就应使用F9标志符使这个缩进的块免于对齐。

问：重叠打印会出现什么情况？

答：新打入的字符代替了原来的正文。

问：如果你利用功能键F5偶然地删除了一个字，这些被删除的字符能恢复吗？

答：如果你还没有移动光标的话，那么利用功能键F6可以一次一个字符地恢复那个字。

问：如果你在使用EasyWriter时不需要程序软盘的话，为什么还要两台磁盘机呢？

答：你应该使用磁盘机B来保存一个后备付本软盘。用BACKUP命令完成后备工作。

问：在编辑程序中打点命令MARGIN和M-MARGIN SET命令之间有何区别？

答：在你输入并对齐文献时，MARGIN SET命令建立页边。而打点命令只在打印期间起作用。

问：你认为要记住哪些最重要的功能键?

答：大概是F10和F4。因为这两个功能键将给你主要的菜单，告诉你接着应做些什么事情。

问：要克服可供编辑一个文献所使用的RAM大小的限制，你能做些什么?

答：你可以把文件连接在一起，克服18,500个字符大小的限制。L—LINK命令使你按顺序把文件的编号列成一个表，在这些连接的文件上完成检索和置换，因此执行H—PRINT。

问：如果你的打印机支持象黑体字、底横线和双倍宽这样一些特殊打印方式，你怎样利用这些特点呢?

答：检查你的使用手册，启动那些特殊控制符号。另外，请你仔细地阅读EasyWriter手册中打点命令部分。

问：如果打点命令不工作的话，那么你在使用它们的方法上可能出什么错误?

答：你可能没有按照打点命令的全部规则使用它们。一定要保证它们在第一列开始，而且单独地在一行上，一个命令对应一行。

问：使用打点命令，怎么建立空行?

答：若使用SPACE 1，则打印页面空二行。若使用SPACE 0，则打印页面空一行。

第 五 章

如何进行展开单计算

人们可见一幅三维全息图。顺便说一句，它是一个闪着绿光的立方体。这是教授第一次用新型的全息“黑板”讲课。

大家期待着这堂课的开始。今天的课程是有关爱因斯坦著名的能量方程式的求导。

教授坐在键盘前，一边讲一边打了几行字。课堂上，三维立方体随着符号的跳动闪闪发光。

我们来看看便明白了。

用VISICALC启动

在第一章中，我们已简要介绍了展开单计算。此外，你还需要弄清第二、三章中介绍的几个简单术语和DOS命令。如果你没读过前几章，那么在使用本章介绍的VisiCalc软盘片之前应回过头浏览一下这两章。以下例子的前提是你已知道如何操作IBM个人计算机。

把VisiCalc软盘插入磁盘机A并复位系统部件（通电或CTRL+ALT+DEL），如果此时出现了一张工作单，那么你的软盘已正确地装配到机器中了。然而，软盘可能没有按你的要求装配，也可能没有自动地从磁盘机A“跳出”。

如果发生这种情况，复位系统部件时就毫无反应，因此你是可以知道的。此时，你必须按下面所述的去做。

标注你的VisiCalc软盘片

如果要在操作员不干预的情况下自动启动VisiCalc软盘，那么我们必须先配备三个文件组。安装这三组文件要执行下面的DOS命令。

把写保护封条从VisiCalc软盘片上揭去，并把该软盘装入磁盘机B。当然，DOS软盘要放在磁盘机A中，再按下列操作把系统文件传送到VisiCalc软盘中：

```
A>SYS B:
```

传送成功之后，屏幕便出现下面信息：

```
system transferred [系统已传送]
```

下一步你还需要复制COMMAND.COM file：

```
A>COPY A:COMMAND.COM B:
        1 File(s)copied
```

最后，你还需要建立一个AUTOEXEC.BAT文件以标注启动过程。这是用第三章中论述过的COPY命令来完成的。这个批文件中几乎可放置你想放入的任何东西。下面只是一种建议，其前提条件是MODE命令和上面提过的三个文件组都已存储在VisiCalc软盘片中。

```
COPY CON:B:AUTOEXEC.BAT
DIR B:
PAUSE
MODE 80
VC 80
```

(F6)(ENTER)

VisiCalc有两种形式，VC 80是80列形式，它使用宽屏幕，因而由MODE 80(方式80)保证。VC 40则是40列形式的，它使用黑体字屏幕，因此，MODE 40可在另一种批文件中使用。

```
COPY CON: B: AUTOEXEC. BAT
DIR B:
PAUSE
MODE 40
VC 40
(F6)(ENTER)
```

现在，取出软盘，把VisiCalc软盘放入磁盘机A，并把工作单软盘(用于贮存工作单)放入磁盘机B。复位系统部件，观察其变化。从这个时候起，你只需把VisiCalc软盘放入磁盘机A，把存储软盘放入磁盘机B，从而开始工作。

VisiCalc键盘

送入专用码，你可以控制屏幕显示5—1所示的VisiCalc工作单的列和行。下面列举了这些专用码，并对其用途作了简单的说明。

/　所有VisiCalc命令都在打印出“斜线”字符后才能输入。你还可看出所有需要输入的命令都是单字符命令。VisiCalc会显示出所有命令的完整拼写，这样，你便可在实际执行之前逐条核对命令。例如，

	成令／F命会变／FORMAT。VisiCalc便在屏幕顶端显示出FORMAT。
>	该符号表示移到某一单元。所以，如果你输入>A1，VisiCalc便移到第1行的第A列上。
!	该符号表示重新计算工作单的全部公式。
;	这个符号用于改变窗口。只有选择分割屏幕时才使用它，见／WINDOW命令。
"	这个符号用于输入正文（标号、题目、题头），而不是输入公式或数字。
箭头	IBM键盘上的四个箭头（即数字小键盘上的8、6、2和4）可用于控制工作单的行和列。记住，这些箭头只有在正确触发NUM LOCK键时才起作用。
CTRL+Z	用于撤消一条命令并重新启动。

VisiCalc术语

回顾第一章，我们知道工作单是一个具有列和行的矩阵。在VisiCalc中，你可从屏幕的左上角开始“生成工作单”。左上角即第1列、第1行，我们称它为A1单元。如果你顺水平方向移动（用箭头），后面的单元便是B1、C1、D1等等。若向下移动，后面的单元则是A2、A3、A4等等。VisiCalc的最大列数是63（A到Z、AA到AZ、BA到BK），最大行数是254。一个行和列的块就是一个矩形数组，由块左上角单元和右下角单元定义的。图5—1示出VisiCalc工作单如何随块的规模增大而生长。一个块用左上单元和右

下单元标明，其间用三个句点分开。

```
A1                                          C
Command: BCDEFGIMPRSTVW—                    35
    A  B  C  D  E  F  G  H
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
```

屏幕显示5—1 VisiCalc工作单

A1…A2

这个块有四个单元，见图5—1。

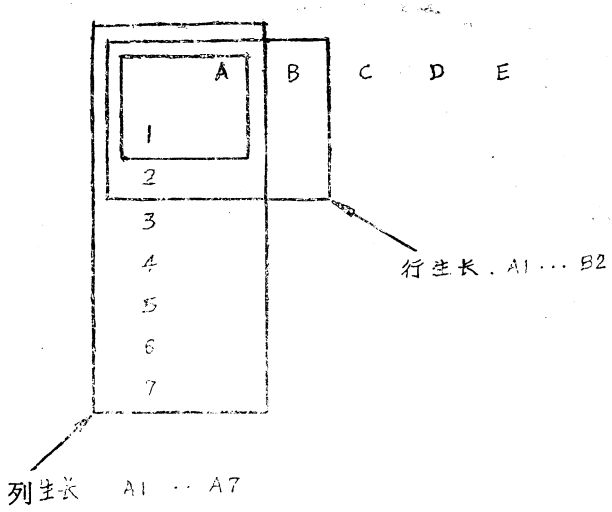


图5—1 VisiCalc工作单的生长

用箭头键在VisiCalc显示的空白工作单上实施行至行、列至列的移动。记住，以下术语在VisiCalc中具有特殊的含义：

激活单元（Active Cell），这是由屏幕上的反向视频指定的当前单元。在你输入一个数字、一个公式或一段正文之后，这个单元便可接受输入。它也称为当前单元。

空白单元（Black Cell），它的值是0。

块（Block），这是一个由左上单元和右下单元确定的行和列的矩形数组。如B3…B5定义了具有两列的矩形数组单元，即：B3、B4、B5和C3、C4、C5。

当前单元（Current Cell），激活单元或由反向视频光标当前定位的单元。

全局命令（Globol Command），该命令适用于工作单中的全部单元。

部分列（Partial Column），一列中相邻的一组单元。例如：C3…C7是部分列。

部分行（Partial Column），一行中相邻的一组单元。例如：C3…H3是部分行。

工作单（Worksheet），一个具有63列、254行的块。每个单元可具有正文标号、数字或公式三者之一。

现在你可以使用VisiCalc了（见屏幕显示5—2）。在下一节中，我们列出VisiCalc的现有命令作为以后参考。本章后几节将用许多例子详细说明如何使用这些命令。

VISICALC命令提要

VisiCalc命令使你能进行许多操作。工作单命令可让你形成几乎是任意长度和宽度的行和列。你可插入和删去所有的行和列。还可得到“元、分”之类格式——以两位小数的精度显示单元值的全局参数，或迫使工作单按行或列的次序计算。

data entry（送数）命令使你能将一行正文、一个数字或一个公式输入到某个或某些单元中。/FORMAT命令让你选择列的宽度、单元内容格式和左右取齐显示。例如，你可以控制一个VisiCalc单元的内容向左或向右调整。

/STORAGE（存贮）命令可将一个工作单送往磁盘文件、打印机或CRT屏幕。该命令用于打印工作单公式和单元格式，或打印贮存在工作单中的全部数值。你还可指定

打印某一个块，从而只打出部分工作单。

```
   A1
(C)1979,1981 Software Arts,Inc. VC-156Y0-IBM 35
   1164238
      A    B    C    D    E    F    G    H
   1
   2
   3
   4
   5
   6
   7
   8
   9
   10
   11
   12
   13
   14
   15
   16
   17
   18
   19
   20
   21
```

屏幕显示5—2 VisiCalc工作单

命令参考表

命令	实例
/BLANK	/B 抹除一个单元的全部内容，空白单元的值是0。
/CLEAR	/C 清除（空白）工作单中送入的全部内容。由于这一命令会擦除整个工作单，所以VisiCalc在做这件事之前先要你“打入Y加以确认”。
/DELETE	/DRC 删去一行（R）或一列（C）。该行或列的内容消失后，其它单元都向上移动，填入工作单。
/EDIT	/E 显示当前单元的内容以便让你编辑。一旦编辑工作完毕，其内容便送回到激活单元中。注意，激活单元在屏幕的反向视频中是光线最亮的部分。注意，使用这条命令时首先要选择激活单元。 /EDIT命令让你用箭头键将一串正文后移，并改变一个或多个字符。数字小键盘中的后移箭头作删除，而左

右箭头则移动单元内容。插入内容只需移到你想插入的地方，然后打印。

/FORMAT　　/FDGILR $*

该命令用于格式化工作单中的当前单元。格式化控制码列举如下：

D　省缺全局格式化，见/G

G　通用的十进制记数的单元内容。

I　整数单元内容

R，L　向右、向左调整单元内容

$　元分记数法

*　图示曲线（条形图）

/GLOBAL　　/GCORF

每用一次/G命令，可置位或复位以下全局触发器。

C　设定窗口中所有列的宽度。例如：/GC 15使现有窗口所有列中的单元宽度都为15个字符。见/WINDOW命令。

O　在计算行和列的值时，规定计算次序。/GOC指一列一列地计算，/GOR指一行一行地计算。计算次序的重要性见后面的例子。

R　指定手动、/GRM或自动/GRA的重新计算。如果你想让VisiCalc等到全部数据都被送入工作单后再进行计算，那么你就要指

定/GRM。否则，整个工作单就得在每个单元值或公式输入后重新计算。“!”是在需要的时候重新手动计算。

M 手动重新计算。用“!”。

A 自动重新计算。

F 设定全局格式。用它指定以前/F命令没有设定格式的所有其它单元的格式。

/INSERT /IRC

插入新的（空白）行或列。相邻的行或列单元后移让位。

/MOVE /M A5…C5

将列或行的内容移到新的列或行，必要时自动调整各单元的公式以保持相同的意义。在插入整个列和行之前，/MOVE先作一次删除。例如A5…C5是将A列删去并插入C列。由于A列已删除，该列的值和公式便插入新工作单的B列中，先前的B列移到新的A列中。

/PRINT /P

该命令用于打印工作单值。但不能打印公式。打印公式要用/SS LPT1:命令。

/REPLICATE /R B5…B5, B6…B10

把一个单元复制到一组单元。复制过的或未复制的公式均可调整。若选择R（＝相对调整），那么公式就会发生变化，以适应它们的新单元。如果选择N（＝无变化），那么公式就照搬复制。

/STORAGE　　/SLSDQ*

将当前的工作单整个地或部分地写入打印机或软盘中。你可任意选择是显示整个工作单还是列出内容（公式和格式的设定）。

L　将文件从软盘输入到工作单中。文件单元的内容在工作单上重叠，因此，新的内容取代旧的内容。其它单元保持不变。你若只需要文件，开始前可先用/Clear命令。

S　把工作单贮存在软盘中。/SS，P任选项的用途是将工作单公式和格式直接送入打印机。见/P。

D　删去VisiCalc工作单中的文件。

Q　退出VisiCalc，返回到DOS。

*S　将工作单按DIF标准格式贮存。DIF标准格式使你能用其它程序处理工作单。例如，你可用BASIC程序或DOS的TYPE命

令存取DIF工作单。

*L 装入DIF格式的工作单文件。

/TITLE /THVBN

锁存标题。将当前的行、列或两者一起锁存起来，使它们不卷动。撤消该命令可用/TN。

H 锁存水平方向的行。所有位于已锁存行上面的行也都锁存起来。用>命令可以进入某一已锁存行中的一个单元。

V 锁存垂向列。用>命令可以消去某一已锁存行中的单元。

B 锁存列和行。

N 一个也不锁存。释放所有已锁存的行和列。

/WINDOW /WHV1SV

将CRT屏幕分成包括工作单不同部分的两个窗口，可以水平（H），也可以垂直（V）分开。消除（1）、同步（S）和/W命令都用I或S，该命令还可用“；”在两个窗口之间进行转换。

H 将屏幕分成“上下”两半。

V 将屏幕分成“左右”两半。

1 恢复一个屏幕。

S 在两屏幕中同步光标的运动，使

它们一起移动。

U　使两屏幕中的光标不同步。

功能参考表

VisiCalc工作单的每个单元可容纳一串正文（“）、一个数或一个公式。每次改变一个与其它单元有关的单元时，单元值将重新计算(除非触发了/GRM)。计算主要是以行的次序从左至右或按列的次序从上到下扫描进行的。

工作单单元的表达式是代数表达式，它包括数、单元名称或下面列举的函数：

+，－，*，／，∧	加、减、乘、除和指数的算术运算。
=,〈 〉.〈,〉,〈=,〉=	等于、不等、小于、大于、小于或等于、大于或等于的比较运算。比较表达式，并计算其成立还是不成立，供IF操作使用。
@ABS (value)	绝对值。
@AVERAGE(list)	单元表的平均值。例如：B9…B12，C1…C6。
@COUNT (list)	计算单元表或块中的非空单元数。
@ERROR，@NA	显示“ERROR”或“NA”。
@EXP (value)	让2.71…自乘到等于该值次幂。
@OR (val1，val2)	如果val1或val2其中一个是真，计算TRUE。
@AND (val1，val2)	若val1和val2都是真值，计算

	TRUE。
NOT (value)	如果值是FALSE,计算TRUE。如果值是TRUE,计算FALSE。
@IF(val1, val2, val3)	如果val1的值是TRUE，用val2，否则用val3。
@INT (value)	丢弃该值的小数部分，恢复原整数部分，其值是截尾而不是舍入。
@LOOK UP(val1,range)	查找块范围内小于或等于val1的最新值，然后，对所找出最新值右列中的单元进行计算，或者对所找出的最新值下一行单元值进行计算。究竟计算哪一个(行、列)则取决于范围(列、行)。
@LN (value)	值的自然对数。
@LOG10 (value)	值的十进制对数。
@MAX (list)	返回单元表的最大值。
@MIN (list)	返回单元表的最小值。
@NPV(discount,range)	计算由范围指定的一个块的纯现存值。
@PI	3.14159…至第16个数字。
@SIN,@ASIN,@COS, @ACOS,@TAN,@ATAN	弧度制三角函数。
@SQRT (value)	平方根。
@SUM (list)	块的总和。

上述函数和其它公式在下面实例中将作说明。

注意，括号可用于构成表达式，如：

(A3+B5)／@SUM(A4…A10)

事实上，在VisiCalc表达式中，括号可避免不确切的含义。若不带括号，整个表达式求值时是从左至右进行的。例如，下面两个表达式计算的竟是相同的值！

+A3+B5／A1

(A3+B5)／A1

要注意，一个表达式必须由加、减等开始，或用括号将它与标号区别开来。

假设A1、A3和B5单元分别是1.5、3.2和6.6，根据上面表达式计算出来的值则是6.533333。现在，假设我们用括号写表达式如下：

+A3+(B5／A1)

计算出的值是7.6!换言之，除非用括号改变求值顺序，否则VisiCalc表达式总是从左至右求值。

在对VisiCalc表达式的含义抱有怀疑时，务必使用括号。括号用得多并无害于VisiCalc表达式；若有遗漏，则可能造成错误。最后我们再列举一对表达式，尽管它们形式上有所不同，但却完成同样的计算。

+C12+B6／A3−A5

(((C12+B6)／A3)−A5)

VisiCalc中最常出现的错误是表达式求值的错误。

公制换算表

也许工作单最简单的例子是换算表。假设我们要设计和执行象图5—2（a）中所示的公制换算表。该表包含一个数量列（A3…A7）、一个公制度量列（B3…B7）和相对应的英制当量。因而，1开标准温度和压力的水等于0.26加仑、0.1057夸脱或0.0353立方呎。

图5—2（b）示出如何使用公制换算表。在这个例子中，你可用箭头键或“go to”将工作单光标移到A6单元。

>A6

输入3.4，第6行中的所有数都要重新计算。于是，3.4公斤等于7.5磅或120盎司。这是怎样计算的呢？

建立公制表

图5—2（a）的公制表是由打出的虚线行和示出的等号行构成。而列的输入形式如下：

>A3

1

现在，在列的其余单元中重复这个值：

/R A3…A3，A4…A7

这就把保留的1放入部分列A4…A7中。注意，你不必在键盘上打上面的所有命令，而将VisiCalc光标放置于A3单元上，然后输入/R。这时，A3便显示出来了。其次，如果你打出ENTER，你会得到短语“…A3”。最后，你若需要指出目的单元，那就只需要将光标移到A4单元，输入一个

Metric To English Conversion Table			
1liter = .26gallons	= .1057quarts	= .0353cubic ft.	
1meter = 1.1yards	= 3.281feet	= 39.37inches	
1kilometer= .62miles			
1kilogram= 7.5pounds	= 120ounces		
1knot = .87miles	= .0002feet	= .0005meters	

图5—2（a）输入VisiCalc的公制换算表

Metric To English Conversion Table			
1liter = .26gallons	= .1057quarts	= .0353cubic ft.	
1meter = 1.1yards	= 3.281feet	= 39.37inches	
1 kilometer = .62miles			
3.4kilogram = 25.5pounds	= 408ounces		
1knot = .87miles	= .0002feet	= .0005meters	

图5—2（b）将A6单元变为3.4的公制换算表

句点，然后把光标移到A7单元，并敲ENTER键。当你在屏幕四周移动光标时，你就能得到命令行。当你打下一个单句点“·”时，它便为你提供三个句点“···”。

接着把B列的正文一次输入到一个单元。D、F和H列也一样。

最后，通过把换算公制值的公式放入各单元，从而完成其它列的输入。例如：C列包括以下公式，

```
0.2642*A3
1.0936*A4
0.621 *A5
2.205 *A6
+A7/1.151
```

实际上，我们可通过输出工作单的内容看到整个工作单中的公式。图5—3所示出的是从下面一系列命令中产生的。

首先确保VisiCalc光标停留在A1单元上（或停留在准备打印的块的右上角单元）

```
          /SS
          LPT1:
>H7 : "meters
>G7 : .0005*A7
>F7 : "feet     =
>E7 : .0002*A7
>D7 : "miles    =
>C7 : .87*A7
>B7 : "knot     =
>A7 : 1
```

```
>F6 : "ounces
>E6 : 120*A6
>D6 : "pounds     =
>C6 : 7.5*A6
>B6 : "kilogram   =
>A6 : 1
>D5 : "miles
>C5 : .62*A5
>B5: "kilometer=
>A5 : 1
>H4 : "inches
>G4 : 39.37*A4
>F4 : "feet   =
>E4 : 3.281*A4
>D4 : "yards      =
>C4 : 1.1*A4
>B4 : "meter      =
>A4 : 1
>H3 : "cubic ft.
>G3 : .0353*A3
>F3 : "quarts   =
>E3 : .1057*A3
>D3 : "gallons   =
>C3 : .26*A3
>B3 : "liter   =
>A3 : 1
>H2 : "==========
>G2 : "==========
>F2 : "==========
>E2 : "==========
>D2 : "==========
```

```
>C2："==========
>B2："==========
>A2："==========
>F1：/FR"Table
>E1："Conversion
>D1："English
>C1："To
>B1："Metric
/W1
/GOC
/GRA
/GC10
/X>A1：>A1：
```

图5—3 公制换算表的公式

这就使得工作单的公式和格式都贮存在打印机中。即把公式送往伪文件I.PT1：中，这样它们就出现在你的行式打印机中。

注意，在图5—3中，正文串前面缀有双引号。所显示的公式和数字都不带引号。带有前缀“/”的“其它”信息给出单元格式信息。例子见F1单元。其它“/”信息列举如下：

/W1　一个窗口

/GOC　按列的顺序计算

/GRA　自动重算

/GC10　列的宽度定为10

我们必须设定各列的宽度以便更好地利用空间。

```
/GLOBAL C 10
```

该命令使所有的列都“扩展”到10个字符的宽度。

在F1单元中，我们用／FORMAT L命令使单元标号向左对齐。即／FL命令使单元的内容移到单元的左边。

显示工作单

图5—2中的输出是由／PRINT命令得到的，输出的是数值而不是公式。因此，要获得工作单的结果（不是公式），其做法如下：

／PP

你可用／PRINT中的／PF形式将完成的工作单送入软盘文件中。

／PF

B：CONVERT

该命令可将打印文件“B／CONVERT·PRF”写在磁盘机B的软盘上，接着，用COPY命令把文件送往打印机，并在软盘上保留一个副本。若使用DOS，则按下面形式完成：

COPY B：CONVERT·PRF LPT1：

除行和列的标号被消除外，输出完全与所见的屏幕一样。

保存工作单

你可用下面的命令在软盘文件中贮存工作单和它的公式、格式和数值。

／SS

B：CONVERT

该命令使所有重建整个工作单所需的信息都保存在B：CONVERT·VC文件中。尔后，在需要3.4公斤的相应值

时，输入原表，并将存储在A6中的值改为3.4。这样，与该值有关的其它值都会自动地重新计算，并给出你想得到的换算值。

/SL

B: CONVERT

该命令可以在屏幕上重建存入前的原工作单。

公制换算表可推广到其它的换算。/INSERT命令可用于增添新行，也可以仅在表下面再加上几行。

所得税申报表

在这个例子中，我们可学到如何使用/REPLICATE命令以及调整选择的方法。假设我们要建立如图5－4所示的所得税申报表。该表用于计算B1单元所显示的校正后总收入的所得税数额。见屏幕显示5—3。

E4 /FG(V)+C4+(D4*(B1−A4)/100) C

	A	B	C	D	E	F	G	33
1	Gross $		55000					
2	over——		Not over——		Taxes	%	Tax Due	
3	————————————————————							
4		3200	4200		0	14	7252	
5		4200	5200		140	15	7760	
6		5200	6200		290	16	8258	
7		6200	7200		450	17	8746	
8		7200	11200		620	19	9702	
9		11200	15200		1380	22	11016	

10	15200	19200	2260	25	12210
11	19200	23200	3260	28	13284
12	23200	27200	4380	32	14556
13	27200	31200	5660	36	15668
14	31200	35200	7100	39	16382
15	35200	39200	8660	42	16976
16	39200	43200	10340	45	17450
17	43200	47200	12140	48	17804
18	47200	55200	14060	50	17960
19	55200	67200	18060	53	17954
20	67200	79200	24420	55	17710
21	79200	91200	31020	58	16984

屏幕显示5—3税收工作单

总计 $ 55000

超过	未超过	税	为	应付税
3200	4200	0	14	7252
4200	5200	140	15	7760
5200	6200	290	16	8258
6200	7200	450	17	8746
7200	11200	620	19	9702
11200	15200,	1380	22	11016
15200	19200	2260	25	12210
19200	23200	3260	28	13284
23200	27200	4380	32	14556

27200	31200	5660	36	15668
31200	35200	7100	39	16382
35200	39200	8660	42	16976
39200	43200	10340	45	17450
43200	47200	12140	48	17804
47200	55200	14060	50	17960
55200	67200	18060	53	17954
67200	79200	24420	55	17710
79200	91200	31020	58	16984
91200	103200	37980	60	16260

图5—4　　所得税表

建立税表

由图5—5中所示的公式，我们获得图5—4。在得出图5—5时，所置的“全列”宽度为10个字符：

/GLOBAL C 10

其次选用了下面的命令：

/SS

LPT1:

图5—5中E列的公式全是很相似的，它们都有相同的模式，如令行号为i，那么Ei的一般公式表示如下：

Ci+Di*(B1—Ai)/100

这个公式对于i＝4直到i＝22都是正确的。

如果按下面的办法去做，我们可以不用对E列中各个单元都重复这个公式。如图5—5所示，先把这个公式送进

	A	B	C	E	G
1	Gross Income 55000				
2	Over——But not over——Taxes				Tax due
3	———————————————————————————————				
4	3200	4200	0+ 14%		C4+E4*(B1—A4)/100
5	4200	5200	140+ 15%		C5+E5*(B1—A5)/100
6	5200	6200	290+ 16%		C6+E6*(B1—A6)/100
7	6200	7200	450+ 17%		C7+E7*(B1—A7)/100
8	7200	11200	620+ 19%		C8+E8*(B1—A8)/100
9	11200	15200	1380+ 22%		C9+E9*(B1—A9)/100
10	15200	19200	2260+ 25%		C10+E10*(B1—A10)/100
11	19200	23200	3260+ 28%		C11+E11*(B1—A11)/100
12	23200	27200	4380+ 32%		C12+E12*(B1—A12)/100
13	27200	31200	5660+ 36%		C13+E13*(B1—A13)/100
14	31200	35200	7100+ 39%		C14+E14*(B1—A14)/100
15	35200	39200	8660+ 42%		C15+E15*(B1—A15)/100
16	39200	43200	10340+ 45%		C16+E16*(B1—A16)/100
17	43200	47200	12140+ 48%		C17+E17*(B1—A17)/100
18	47200	55200	14060+ 50%		C18+E18*(B1—A18)/100
19	55200	67200	18060+ 53%		C19+E19*(B1—A19)/100
20	67200	79200	24420+ 55%		C20+E20*(B1—A20)/100
21	79200	91200	31020+ 58%		C21+E21*(B1—A21)/100
22	91200	103200	37980+ 60%		C22+E22*(B1—A22)/100

图 5—5　税表中的公式

E4单元，然后通过相对调整来复制这个公式：

/REPLICATE　　E4…E4，E5…E22

它使该公式对所有单元名查询：每次见到一个名字，VisiCalc等候着一个“R”（相对调整）或“N”（不变）。因而，我们送进如下的“R”和“N”来响应各个单元名：

C_4　　“R”

D_4　　“R”

B_1　　“N”

A_4　　“R”

这样，除了B_1以外所有单元名都作了调整，其结果见图5—5。

如果我们改变B_1的值，那么部份列E_4…E_{22}中的所有项都要重新计算过。这些值给出了每个收入等级应缴的所得税额。不过让我们把下面的公式加进单元B_{23}，稍微改变一下这张工作单。

修改后的税表

假定我们把E列移到B列，如下：

/MOVE　From … To

E_4…B_4

输入这个命令时，首先把光标定位到E_4单元，再打/M，然后将光标移到单元B_4，这样“E_4…B_4”就显示在VisiCalc编辑行上了。

正如图5—6所示，所有的公式都自动得到修正。老列B，C，…中的单元都已经右移了一个列。这就意味着新列B中的公式已作修正，同老列E中的公式意义相同。

```
>B23 : @LOOKUP ( C1, A4…A22 )
>A23 : "Tax due =
>E22 : 60
>D22 : 37980
>C22 : 103200
>B22 : /FG+D22+ ( E22* ( C1−A22 ) /100 )
>A22 : 91200
>E21 : 58
>D21 : 31020
>C21 : 91200
>B21 : /FG+D21+ ( E21* ( C1−A21 ) /100 )
>A21 : 79200
>E20 : 55
>D20 : 24420
>C20 : 79200
>B20 : /FG+D20+ ( E20* ( C1−A20 ) /100 )
>A20 : 67200
>E19 : 53
>D19 : 18060
>C19 : 67200
>B19 : /FG+D19+ ( E19* ( C1−A19 ) /100 )
>A19 : 55200
>E18 : 50
>D18 : 14060
>C18 : 55200
>B18 : /FG+D18+ ( E18* ( C1−A18 ) /100 )
```

```
>A18 : 47200
>E17 : 48
>D17 : 12140
>C17 : 47200
>B17 : /FG+D17+ ( E17*( C1−A17 ) /100 )
>A17 : 43200
>E16 : 45
>D16 : 10340
>C16 : 43200
>B16 : /FG+D16+ ( E16*( C1−A16 ) /100 )
>A16 : 39200
>E15 : 42
>D15 : 8660
>C15 : 39200
>B15 : /FG+D15+ ( E15*( C1−A15 ) /100 )
>A15 : 35200
>E14 : 39
>D14 : 7100
>C14 : 35200
>B14 : /FG+D14+ ( E14*( C1−A14 ) /100 )
>A14 : 31200
>E13 : 36
>D13 : 5660
>C13 : 31200
>B13 : /FG+D13+ ( E13*( C1−A13 ) /100 )
>A13 : 27200
```

```
>E12 : 32
>D12 : 4380
>C12 : 27200
>B12 : /FG+D12+ ( E12*( C1−A12 ) /100 )
>A12 : 23200
>E11 : 28
>D11 : 3260
>C11 : 23200
>B11 : /FG+D11+ ( E11*( C1−A11 ) /100 )
>A11 : 19200
>E10 : 25
>D10 : 2260
>C10 : 19200
>B10 : /FG+D10+ ( E10*( C1−A10 ) /100 )
>A10 : 15200
>E9 : 22
>D9 : 1380
>C9 : 15200
>B9 : /FG+D9+ ( E9*( C1−A9 ) /100 )
>A9 : 11200
>E8 : 19
>D8 : 620
>C8 : 11200
>B8 : /FG+D8+ ( E8*( C1−A8 ) /100 )
>A8 : 7200
>E7 : 17
>D7 : 450
```

```
>C7 : 7200
>B7 : /FG+D7+ ( E7* ( C1−A7 ) /100 )
>A7 : 6200
>E6 : 16
>D6 : 290
>C6 : 6200
>B6 : /FG+D6+ ( E6* ( C1−A6 ) /100 )
>A6 : 5200
>E5 : 15
>D5 : 140
>C5 : 5200
>B5 : /FG+D5+ ( E5* ( C1−A5 ) /100 )
>A5 : 4200
>E4 : 14
>D4 : 0
>C4 : 4200
>B4 : /FG+D4+ ( E4* ( C1−A4 ) /100 )
>A4 : 3200
>F3 : "−−−−−−−−−−
>E3 : "−−−−−−−−−−
>D3 : "−−−−−−−−−−
>C3 : "−−−−−−−−−−
>B3 : "−−−−−−−−−−
>A3 : "−−−−−−−−−−
>E2 : /FR"%
>D2 : /FR"Taxes
```

```
>C2：/FR"Not over-
>B2：/FR"Tax Due
>A2："Over--
>C1：55000
>A1："Gross  $
  /W1
  /GOC
  /GRA
  /GC10
/X>A1：>A3：/TH
/X>A1：>A1：
```

图5—6　修改税表的公式与格式

查找税表

让我们看一下图5—6中的B_{23}单元。LOOKUP（查找）函数是用来计算定额为$55000（用单元C1表示）的税收的。

LOOKUP（C1，A_4…A_{22}）

该函数把存在C1中的值（55000）作为查找值。然后往下查找列A_4…A_{22}，找出小于55000但又最接近55000的数。该数是单元A_{18}中的值47200。LOOKUP回送值是单元B_{18}中的值。因而单元B_{23}中的值为17960.00。（参看图5—7）。

这里说明的修改是利用LOOKUP函数来检索工作单。因为LOOKUP使用到紧挨着右边的列，我们必须把E列移

到B列，因而，为了利用LOOKUP，我们移动了“应付税”列。

在行方面，LOOKUP的工作也一样。比如，我们指定一个部份行为待查找的行，那么回送值将是下面出现的值。

VisiCalc税表使得计算你的所得税变得很容易。困难之处在于做出IPS！（分期支付展开单！）

Gross $		55000		
Over--	Tax Due	Not over-	Taxes	%
3200	7252	4200	0	14
4200	7760	5200	140	15
5200	8258	6200	290	16
6200	8746	7200	450	17
7200	9702	11200	620	19
11200	11016	15200	1380	22
15200	12210	19200	2260	25
19200	13284	23200	3260	28
23200	14556	27200	4380	32
27200	15668	31200	5660	36
31200	16382	35200	7100	39
35200	16976	39200	8660	42
39200	17450	43200	10340	45
43200	17804	47200	12140	48
47200	17960	55200	14060	50
55200	17954	67200	18060	53
67200	17710	79200	24420	55

79200	16984	91200	31020	58
91200	16260	103200	37980	60

Tax due = 17960

图5—7修改后的所得税表

分期支付展开单

VisiCalc对于“如果…会怎么样？”的计算类型是最有效的。我们可以先构建一个财政情况的数值模型，然后求“如果…会怎么样？”的问题。这个概念由图5—8所示的分期支付工作单加以说明。

分期支付工作单			便条计算
借支总额	$	60000.00	1.0100585666（年金）
年利率	%	16	013333333333（按月）
月偿还	=	348	100.41774326（现在）
月支付	$	808.05	
总支付	$	281200.30	
总利率	$	221200.30	

图5—8分期支付模型

建立分期支付模型

我们用来构建如图5—8所示工作单的财政模型是以下

面的年金公式为基础的：

支付＝（总额）*（利率）*（年金）

年金＝V／V－1

V＝（利率＋1）∧月数

因此，C_5单元装有V，而C_3单元装有（年金）等等。这个模型取决于保存在单元B_3…B_5中的值。例如，如果我们改变其中某一值，那么所有的计算必须重新做过，以反映相应的变化。这意味着我们能问“如果…会怎么样？”的问题。

“如果…会怎么样？”的问题

我们先来问几个问题。如果我们按16%（利率）借入$0,000，348个月（29年）后会怎么样？得出的值见图5—9所示。

月支付	$	808.05
总支付	$	281,200.30
总利率	$	221,200.30

简单地说，我们获悉：借入$60,000，29年后偿还，要多付近二十五万元！

那么，如果将利率变为15%会怎么样呢？这就改变了保存在C_4单元中的值，并且依次改变C_5、B_7、B_8和B_9的值。注意，这也是一个相当麻烦的问题。

```
>C10：〃－－－－－－－－－－－－－－－
>B10：〃－－－－－－－－－－－－－－－
>A10：〃－－－－－－－－－－－－－－－－
>B9：/F$（BB－B3）
```

```
>A9 : "Total inserest      $
>B8 : /F $ ( B7*B5 )
>A8 : "Total Payment      $
>B7 : /F $ ( C4*C3*B3 )
>A7 : "Monthly payment      $
>C6 : "----------
>B6 : "----------
>A6 : "----------
>D5 : " ( present )
>C5 : ( C4+ 1 ) B5
>B5 : /FI348
>A5"Months to repay      =
>D4 : " ( per month )
>C4 : +B4/1200
>B4 : /FG16
>A4 : "Annual interest %
>D3 : " ( annuity )
>C3 : +C5/ ( C5- 1 )
>B3 : /F $60000
>A3 : "Amount borrowed      $
>C2 : "          Calculations
>B2 : "----------:
>A2 : "----------
>C : /FR"Scratchpad
>B1 : "     Worksheet
>A1 : /FL"Installment payment
```

```
/W1
/GOR
/GRA
/GC19
/X>A1：>A1：
```

图5—9　分期支付模型的公式和格式

单元C_3取决于单元C_5中的值。但是，因为VisiCalc自上至下复算一切单元（在这个情况是按行），单元C_3中使用的C_5值是“老的”，这就是说，在C_5的值用在单元C_3之前并没有重新算过。这意味着C_3并非准确！我们可以采用复算命令修正这个错误：

```
!
```

例如，在单元B_4中送进15%以代替16%。得出的非准确值是：

```
月支付    $  757.54
总支付    $  263625.29
总利率    $  203625.29
```

我们命令VisiCalc再复算一遍以确保无误：

```
!
```

正确值就得出来了。我们知道它们是正确值，因为跟着的“!”命令并没有使它发生变化。

```
月支付    $  760.08
总支付    $  264507.25
总利率    $  204507.25
```

下面是我们能向这个财政模型提出的另一些“如果…会

怎么样？”问题。如果改变偿还期会怎么样呢？如果改变借入的总数会怎么样呢？在每种情况下，我们都用“!”命令VisiCalc把所有的值都复算过，从而确保其回答正确无误。

在某些情况中，改变计算次序可以避开复算。例如，若逐列扫描工作单可得出正确值，那么就可改为按列计算：

```
/GLOGAL  O  C
```

这时，全部公式是按A列、跟着B列、C列等等求值，直到算完全部列。因为各个列中的公式是首先求值的，因此称谓它为按列计算。在按行计算中，首先求值的是各个行中的全部公式（行1，行2，3…等等）。

但是，为了确保正确无误，我们可以用“!”按两种次序复算整个工作单。如果无变化，你就知道计算次序並不是很重要的了。

最后，可能你还想用手控复算，即，只在“!”命令后，暂停整个计算，这样，你就要触发/GR开关：

```
/GLOBAL  R  M
```

它使所有的复算中止，直到由用户输入一个“!”为止。如此，在手控复算命令给出以前，全部值将保持原样。

这个例子说明了VisiCalc的一些有效的执行过程，也说明了如何用最少的打入来输入一个工作单。通过后面的两个例子，我们将进一步向你说明VisiCalc的特点，这对你本人去做工作单的工作是很有用的。

实验者用数据简化展开单

科学家、心理学家、农艺管理者、工程师以及统计学家

都需要对他们的数据进行各式各样的统计数据简化计算。VisiCalc正是快速实现这些计算的一种理想的方法。在这一节中，我们研究该方法的一个实例，使得你可以在一个观察值的表上使用VisiCalc去做数据的简化。

在统计学术语中，一个观察值表是从某次试验取得的测量结果的一个集合。例如，我们试图在一次尺码测试中确定英寸数。这可以用一钢尺或别的东西多次测量这个尺码。每次测量结果叫做一个观测。可以预料，各个观测都包括真实值加上或减去少量误差。

图5—10所示为实验者5次观测的结果。因为每次都有一个（少量）试验误差，各个观测结果是並不相同的。

平均值（36.306）是尺码真实长度的一个估计量（参看图5—10）。但是，我们知道用来计算平均值的各个观测都包含某些未知的误差，因而平均值含有某些错误。我们希望，正的和负的偏差将互相抵消，使得这个平均值与任何单个观测结果相比是一个较好的估计量。事实上，因为观察次数是无限增加的，观察测量中正的和负的偏差将趋向相互抵消。因而，一个数目很大的观察值的集合产生一个“较好的”平均值。这就是统计学家在平均值计算中用来估计概率误差的“大数定律”。

Number Observations = 5 Diff 2

Observed	
36.75	.19713600003
35.55	.57153599994
34.67	2.6764960006

	39.23	8.5497759649
	35.33	.95257599296
Sum of Observed =	181.53	12.9475199583
Avg of Observed =	36.306	1.2135152903
		=probable error

图5—10　一个实验者数据简化表

图5—10中含有标示“概率误差”的数。这个值“多半”按平均值正或负偏移。这样，尺码的长度估计值为有正或负概率误差的平均值。在图5—10的例子中，我们给出回答如下：

1码＝36.306　　＋或－1.214英寸

这尺码的“大概”长度在35.092和37.520英寸之间。这就是我们如何计算这个有趣的试验的结果。

建立试验者表

观察结果列中的平均值用VisiCalc的标准函数AVERAGE计算。它是在单元B_{11}中计算的（参看图5—11）：

@AVERAGE　　($B_4 \cdots B_8$)

该平均值也是用来估计每个值同真实值“正负”偏移大小的。这样，C列含有差值的平方：

($B_4 - B_{11}$)^2

命令/REPLICATE把该公式复制进剩余的单元，且同时把

B_4调整到B_5, B_6, B_7和B_8,

/REPLICATE $C_4 \cdots C_4$, $C_5 \cdots C_8$

```
>D11 : "=probable error
>C11 : .6745*@SORT ( C10 ) /@SORT ( B1-1 )
>B11 : @AVERAGE ( B4…B8 )
>A11 : "Avg of Observed =
>C10 : @SUM ( C3…C8 )
>B10 : @SUM ( B4…B8 )
>A10 : "SUM of Observed =
>E9 : "----------
>D9 : "----------
>C9 : "----------
>B9 : "----------
>A9 : "----------
>C8 : ( B8-B11 )^2
>B8 : 35.33
>C7 : ( B7-B11 )^2
>B7 : 39.23
>C6 : ( B6-B11 )^2
>B6 : 34.67
>C5 : ( B5-B11 )^2
>B5 : 35.55
>C4 : ( B4-B11 )^2
>B4 : 36.75
C3 : "----------
```

```
B3:"----------
A3:"----------
>C2:/FR"Diff^2
>B2:/FR"Observed
>B1:FI5
>A1:"Number Observations =
/W1
/GOC
/GRA
/GC21
/X>A1:>A1
```

图5—11　试验表的公式与格式

你可能已注意到，C列的各个差数的计算都用了保存于B_{11}中的平均值。但是，如果我们按行计算的话，那么B_{11}在C列之后而不是之前计算。让我们换用按列计算的方法改变一下计算的次序：

```
/GLOBAL    O  C
```

这时，首先计算B_{10}，B_{11}中的平均值以及和，然后计算C列中的差数，最后计算概率误差。

给出的概率误差公式如下：

0.6745*@SQRT(@SUM(C_3…C_8))/@SQRT(N−1)

这里，N是总的观测次数，此处为5。0.6745这个值来自统计学，因为误差多半是正或负的0.6745乘上@SQRT表达式。

改 进 表

在这个实例用过的表中插入附加行（#9），我们就可以扩充这张实验者的表这样做时一定要用另外的公式去改进组合表达式，以反映出这个补充。因此，B_8应改为B_9，等等。

你大概想用这个方法分析实验收集到的试验数据。下面是几条建议：

风力机数据

按哩计算汽油费

个人重量增加／减少

VisiCalc图解

最后让我们再举一个例子，考虑一下如何用VisiCale按条形图方式显示数据，以说明其实用性，在/FORMAT*就可以获得这个特点。

为了用条形图来代替数值显示，我们可使用下列格式命令：

/FORMAT*

该命令放在工作单的C_2单元、然后复制到列$C_2 \cdots C_{18}$。

/REPLICATE $C_2 \cdots C_2$, $C_3 \cdots C_{18}$

图5—12说明一个条形图是怎样出现于VisiCalc工作单中的，A列装年份，而B列装有美国1958年每1千人口的死亡人数，C列装有计算显示字符的数目的公式。星号数目相当于B列中的数字。在图5—12中，一个*等于每1千人

口死亡2个人。

```
Age  Deaths    1     *equals     2
 0     7.08  ***
 5     1.35
10     1.21
15     1.4
20     1.8
25     1.9
30     2.1   *
35     2.5   *
40     3.5   *
45     5.4   **
50     8.3   ***
55     13    *****
60     20.3  ********
65     31.8  ********
70     49.8  ********
75     73.4  ********
80     110   *********
```

图5—12　条形图工作单

建立死亡率表

图5—13所示是为了产生死亡率条形图每个单元中应输入些什么。C列是计算B列单元值对单元D1值的比率。

实际上，工作单是分割成二半的，由屏幕分割命令把屏

幕分为左半面和右半面：

/WINDOW V

这时的屏幕显示将如屏幕显示5—4表示的那样。在送进这个命令之前，光标一定要放在C列位置上。一旦送进后，你必须用“；”命令来回左、右翻动：

；

现在注意右半面的宽度比左半面的大得多。命令/GLOBAL C 50可扩充右半面（C列）、而不管左半边。这是/WINDOW特性的优点之一。这样，C列的宽度是50个字符，而其它列的宽度是10个字符。

我们已规定C列的宽度是50，且显示格式是“星号”。我们也硬性规定正文向右取齐（R），条形图向左边取齐（L）。分割窗口的特点使得有可能在单个工作单中设定不同的格式与列宽度。

为了显示的目的，你可以用／P命令建立MORT．PRF文件。一定要把光标定在你所要显示字块的右上边单元处：

/PRINT F

$A_1 \cdots C_{18}$

这就导致字块$A_1 \cdots C_{18}$保存在文件MORT．PRF之中。但是，你可以看到PRF文件是按左边屏幕规定的格式保存的。因而它是按窄格式印出的。为了克服这个不幸的限制，你得要把左边屏幕的全局格式也改为50。但是，这有可能使工作单太宽了。不能在打印机上显示。为了避开这一问题，你可用PRTSC键打印屏幕的内容，这就产生屏幕显示5—4。

>C18：/F*+B18/D1

```
>B18 : 110
>A18 : 80
>C17 : /F*+B17/D1
>B17 : 73.4
>A17 : 75
>C16 : /F*+B16/D1
>B16 : 49.8
>A16 : 70
>C15 : F*+B15/D1
>B15 : 31.8
>A15 : 65
>C14 : /F*+B14/D1
>B14 : 20.3
>A14 : 60
>C13 : /F*+B13/D1
>B13 : 13
>A13 : 55
>C12 : /F*+B12/D1
>B12 : 8.3
>A12 : 50
>C11 : /F*+B11/D1
>B11 : 5.4
>A11 : 45
>C10 : /F*+B10/D1
>B10 : 3.5
>A10 : 40
>C 9 : /F*+B 9 /D1
>B 9 : 2.5
>A 9 : 35
>C 8 : /F*+B8/D1
>B 8 : 2.1
```

```
>A8 : 30
>C7 : /F*+B7/D1
>B7 : 1.9
>A7 : 25
>C6 : /F*+B6/D1
>B6 : 1.8
>A6 : 20
>C5 : /F*+B5/D1
>B5 : 1.4
>A5 : 15
>C4 : F/*+B4/D1
>B4 : 1.21
>A4 : 10
>C3 : F/*+B3/D1
>B3 : 1.35
>A3 : 5
>C2 : /F*+B2/D1
>B2 : 7.08
>A2 : 0
>D1 : /FL2
>C1 : /FR"1 * equals
>B1 : /FL"Deaths
>A1 : /FL"     Age
  /W1
  /GOC
  /GRA
  /XV23
  /GC10
  /X>A1 : >A1 : ; /GC40
  /X>C1 : >C3 : ;
```

图5—13　条形图的公式与格式

C2 /F* (V) +B2/D1 C34

	A	B		C
1	Age	Deaths	1	1*eduals
2	0	7.08	2	***
3	5	1.35	3	
4	10	1.21	4	
5	15	1.4	5	
6	20	1.8	6	
7	25	1.9	7	
8	30	2.1	8	*
9	35	2.5	9	*
10	40	3.5	10	*
11	45	5.4	11	**
12	50	8.3	12	****
13	55	13	13	******
14	60	20.3	14	**********
15	65	31.8	15	***************
16	70	49.8	16	************************
17	75	73.4	17	*********************************
18	80	110	18	**
19			19	
20			20	
21			21	

屏幕显示5—4 分割屏幕条形图

常 见 问 题

问：可否用VisiCalc单元中的公式去修改其它单元中别的公式？

答：不能。但是你可以用IF函数有条件地利用一个公式或另外的公式，这取决于其它单元中所存的值。

问：一个单元的最大宽度是多少？

答：是77个字符。通常这个宽度是屏幕宽度减3。而最小宽度是3个字符。

问：如果你搞错了，在该空白的单元里放进了东西，该如何清除这个单元呢？

答：用/BLANK。将光标移到该单元处再打/B(ENTER)。

问：/COPY和/REPLICATE的区别是什么？

答：/CORY命令是一对一地制作付本。/REPLICATE命令则是将一个单元复制到多个其它单元。

问：/MOVE与/COPY命令之间的区别是什么？

答：/MOVE总是用来调整公式并且擦掉原来的行或列。/COPY并不擦去原来的行或列。

问：如插入行或列已装有数据，/INSERT会破坏它吗？

答：不会。它把所有东西向一个方向移动，且自动调整公式。

问：怎样将两个或多个工作单编排在一起，使之成为一个大的工作单？

答：用／STORAGE S把它们记录在软磁盘上，然后用/STORAGE L把一个叠在另一个的顶上。

问：怎样从一个工作单的边上卸掉列与行标号？

答：不能直接在工作单上做。但是，从/PRINT获得的FRF文件没有边，所以你可以用它来代替。

问：怎样从头至尾擦掉一个完整的工作单？

答：用／CLEARY。

问：怎样取消一个错误的输入行？

答：有几种取消方法。错误行未送进当前单元之前，先用CTRL＋Z取消该行；而／EDIT命令则可用来取消已写单元。

问：怎样打印一个工作单？

答：/PRINT F或/PRINT P仅用于打印值；命令/S*用于打印公式和格式。

问：／TITLE的用途是什么？

答：为了便于卷动屏幕时保持标题。

问：屏幕右上边的“存储”信息意味着什么？

答：它告诉你RAM还剩多少（以1K为变量单位）。这数目越少，工作单用的存贮越多。

问：用装入命令/STORAGE的时候，有没有办法显示磁盘上的文件名？

答：有的。按SL键和右箭头键（数字小键盘上的键6）。你每按一次箭头键、它就显示另一个文件名，用ENTER取出你要的文件名。例如：

/SL

B:

现在按右箭头，同时眼睛看着编辑行中出现的来自磁盘机B的文件名。当你要的文件显示在上面时，按（ENTER）键，这样就把它加载进系统了。

第 六 章

怎样写你自己的程序

某古代废墟处，两个考古学家小心地放慢了脚步。傍晚，一个考古学家被一大堆纤维物绊倒。她拾起这东西，但是，由于自身的重力，这东西开始破碎。她的同伴迅速地把记录仪对准那个裂变的人工制品，这一古代发现当即被记录下来，送回飞船进行分析。

实验室的检验揭示了这本珍贵书籍的来源及大约写作年代。这本书属于21世纪那个行星上存在的古代文明。但是，这本书使用了奇怪的不可理解语言。例如“BASIC工资计算程序—Scottly Moore著”这句话是什么意思？

软 件 工 具

第一章说明了计算机程序员是如何用抽象与规划的方法来解决种种问题的。讨论的抽象等级有：

要求定义

设计说明

编码说明

执行（本章的主题）

测试

文件编制

每一级都用简单步骤在下一级中细化，最后得出一个完整程序。由于这是一种正确产生有效程序的方法，所以被称之为**结构设计**。

在本章中，我们将研究各种实际编写计算机程序的方法。我们假定读者已经掌握了一套结构设计的方法，因此只专门讨论抽象的执行级。

而且，我们将把本章的讨论限于两种语言的翻译程序：高级BASIC及Pascal。回顾第二章（及第八章）可知BASIC是一种解释语言，而Pascal是一种编译语言。

对计算机程序员来说无所谓那种语言是“最好的”。BASIC语言学起来简单，用起来也容易。这种语言被直接翻译并执行，没有费时的编译这个中间步骤。因此，就快速产生结果而言，BASIC比Pascal更为优越。

另一方面，Pascal语言是“结构式的”，读起来容易。因此，Pascal程序容易维护和修改，而且可以插在其他程序之中。因而，Pascal适用于大系统，可以在以后保留并增补许多独立的程序段。

那末，那一种更好呢？这也许取决于个人，你可以为自己作出一种判断。一般说来，BASIC应当用来写小程序，这样你可以很快获得结果，而Pascal则应当用来写长期使用的大程序。

无论用BASIC或Pascal设计计算机程序，不使用适当的工具就有可能成为一项困难的任务。IBM个人计算机提供了下列软件工具：

编辑程序。DOS 中的 EDLIN 命令及 BASICA 中的 EDIT 命令都是文本编辑程序，可用于输入和修改程序。

文件管理程序。DOS中的COPY、TYPE、RENAME、ERASE命令及BASICA中的SAVE／LOAD／FILES命令都是文件管理程序系统，可用来传送存在软盘上的程序。

语言处理程序。BASICA是BASIC的一种解释程序，通常用于执行和测试新程序。PAS1是Pascal的一种编译程序，常用于把工作程序翻译成快速运行的二进制机器编码程序。

这些工具只有在你把一个程序的抽象等级细化成一个具体的源程序以后才能使用。然后，你就可以把源程序输入到计算机“运行”。

如果你不熟悉这里所使用的术语，请查阅一下第八章以后，再继续往下读。如已熟悉，那么你就可以学习自己编写程序的详细过程。然而，首先需要一些软件工具。

使用编辑程序

编辑一个程序有两种方法：

1、BASICA输入及EDIT命令

2、DOS EDLIN编辑程序

第一种方法通常用于输入和修改BASIC程序，第二种方法通常用于输入和修改Pascal程序。但是，如果你记住用ASCII方式保存BASICA文本，那就可以用BASICA编辑程序代替EDLIN编辑程序，反之亦然。例如，你可以在SO-

URCE（源）文件里保存一个程序，再用EDLIN检索：

SAVE “SOURCE”，A

此后，在使用EDLIN时，按下面所示方法用SOURCE检索正文：

A>EDLIN SOURCE·BAS

反之，你也可以用EDLIN建立一个程序，并且在执行该程序之前按下列所示装入：

LOAO “SOURCE”

肯定你要用EDLIN的扩展类型“.BAS”，纵然是因为BASICA希望文件名是上述的SOURCE．BAS。

第二种方法通常用来产生Pascal源文本。实际上，在写该文本时，这是输入Pascal源文本仅有的切实可行的编辑程序。但是，由于EDLIN是一种行编辑程序，而构成BASICA的是一种屏幕编辑程序，因此我们要写一个叫做“切条”（stripper）的程序，该程序把用BASICA输入的程序转换成Pascal编译程序能够接受的程序。

总之，一个程序可以按下述办法进行输入和修改：

BASICA**编辑方式** 你可以用这个“屏幕”编辑程序输入和编辑一切BASIC程序。此外，你也可以用这个编辑程序输入和编辑一切Pascal程序，不过，你先要做两件事：（a）用ASCII方式保存文件，（b）在用PAS1编译Pascal程序前先用下节所给的“切条”程序。

EDLIN你可以用这个“行”编辑程序输入和编辑一切BASIC或Pascal程序。如果你用这个程序来输入BASIC程序，那么一定要在每条语句前置以行号。

如果你决定全部工作都用EDLIN来做，那么就请参阅

描述EDLIN命令的一节。但是，如果你决定用屏幕编辑程序来构成BASICA（及IBM个人计算机键盘／屏幕），那么请继续阅读下面一节。

BASICA屏幕编辑程序

从DOS启动BASICA只要打出以下命令：

A>BASICA

这将把BASICA解释程序装入RAM并开始执行。当有OK应答后，你方可使用下面的屏幕编辑程序命令。

AUTO	自动显示行号
DELETE	擦掉某些行
EDIT	编辑某一行
LIST	显示某些行
LLIST	打印某些行
MERGE	连结程序块
RENUM	重新编程序行号

除这些命令外，键盘还包括下述屏幕编辑功能。

箭头键：四个箭头键可用来把光标移到屏幕上显示的任何行。

INS：这个键使你能在光标上方那个字符的前面插入正文。输入完正文后，就敲ENTER键（其中一个箭头键）或END键。

DEL：删除键能擦掉光标上的字符。你可看到每按一次DEL键，就会有一个字符消失。

END：这个键使光标跳到当前行的末尾。

左箭头键：这个键反向删除。

让我们举个例子来说明BASICA屏幕编辑程序。用AUTO命令开始输入下面的程序：

```
AUTO  100, 10
```

这一命令能以每次加10行的方式自动显示行号。

```
100 INPUT A, B, C
110 PRINT A+B, A+C
120 STOP
```

你可用CTRL+C终止自动行编号命令。现在，用LIST命令看一下你输入了些什么：

```
LIST
```

它把完整的程序显示在屏幕上。

接着，用箭头键把光标移到已显示的字符下。假定我们要把第110行的A+B变成A－B，那么就把光标移到＋号下面并敲DEL键，＋号就消失。然后敲INS键并打出字符－。现在，这行就不是A+B而是A--B了。把光标返回原位，再次列出被修改的程序：

```
LIST
```

现在，程序的两个付本都显示在屏幕上。如果我们对二者进行屏幕编辑，会出现什么情况呢？当你作两种相反的变换时，最后一个变换将起作用。

你还可以把EDIT命令用于编辑屏幕的某一行。敲打EDIT及行号，该行将在打印的下面一行上显示，然后可用上述方法来编辑这一行。

如果你打算运行你的程序，并用BASICA检测语法错误，那么错误的行就同你输入EDIT命令一样被显示出来，你可以通过屏幕编辑立即改正。

功能键

你还可以看到显示在第25行（在屏幕底部）的功能键。该行告诉你在键盘左边的F1至F10键的意思。你可以用KEY命令改变这些键的含义。例如，要把F1定义为EDIT键，输入下面的命令：

```
KEY1、"EDIT"
```

现在，按F1键就会显示EDIT。这可节省你的打字时间。

加快打字速度的另一种方法是用ALT键。如果在打BASICA某些关键字的第一字母时，仍保持按下ALT键，解释程序将为你拼成一个命令。下面是一些可缩写的关键字，至于完整缩略字表，请参看BASIC手册。

```
ALT+A=AUTO
ALT+F=FOR
ALT+G=GOTO
ALT+I=INPUT
ALT+K=KEY
ALT+N=NEXT
ALT+P=PRINT
ALT+R=RUN
```

最后，记住用SAVE命令（或F4）把你的程序保存在磁盘上：

```
SAVE "TAXES",A
```

它将把当前程序保存在一份称为TAXES·BAS的ASCII格式文件里。要检索该程序时就用LOAD命令：

LOAD "TAXES"

要从RAM擦掉当前程序时就用NEW命令：

NEW

要看软盘上有些什么程序文件时就用FILES命令：

FILES

或者

FILES "B：*.*"

最后一个例子显示磁盘机B上的所有文件。

EDLIN行编辑程序

为了从DOS启动行编辑程序，只要打出命令及你要建立或编辑的文件名：

A>EDLIN TEST·PAS

这就生成并存贮了一个叫做TEST.PAS（包含一个Pasc-al源程序）的新文件：

A>EDLIN B：TEST·PAS

如果该文件以前不存在，EDLIN程序就产生一个新文件，如果存在，则把文件读入RAM。

在输入、修改或检查文件以后，你可以退出行编辑程序，这将使以前的文件用扩展·BAK重新命名。于是，你总是有一份后备副本，可以用来校正编辑错误。（该文件必须重新命名后方可被EDLIN识读。）

EDLIN是一种典型的行编辑程序。即可以一次只"指示"单独一行。"当前行"就是EDLIN当前所选择的行。当前行的第一列都标有 *号。

EDLIN的编辑程序命令操作当前行光标。为了把光标

移到另一行，必须输入行号及ENTER键。当前行显示在一行，其行号显示在第二行。你可以用箭头键移到当前行，并可按以前描述的方法编辑当前行。

你可以用箭头键、INS(插入)键及DEL（删除）键来编辑当前行。如果你用箭头键把光标移到当前行，该行的当前字符就被显示出来。如果你插入新正文，那么新正文将插到屏幕光标所指的位置。同样，每次用DEL键，屏幕光标所指位置的正文就被删除。

为了开始输入正文，或者插入一行或若干行正文，用插入命令：

```
1i
9i
```

这些命令使一行新的正文当即被插在第一行及第九行前。同样，你可以删除若干行，命令如下：

```
10，15d
```

它将删除第10至第15行，剩下的行被重新编号，以“填补”删除行所留下的“空白”。

所有移动当前行光标的命令都是先输入指定该行或有关的行，然后再接命令字母：

〈行号〉　〈命令字母〉

因此，下面的命令用于一行或一行以上，L#后面跟着单个命令字母。

EDLIN命令

L#A	以磁盘读出L#行并把它们增补到文本。它用于那些RAM容纳不下的大型文件。

L#，L#D　删除L#至L#行。如果其中一个数省略，就从第一行开始或者到最后一行结束。

L#　编辑行号L#。

L#I　从行号L#开始插入新行。如果L#行已经存在就往下移让出空行。

L#,L#L　在屏幕上显示L#至L#行。

L#,L#?R

s1(F6)s2　这是检索和替换命令。它操作从L#到L#的所有行。"?"的意思是每当搜索到相等时就询问是否可以替换。找到s1中的模式并用s2代替。功能键F6是用来分离这两个串。如果省略"?"，那么不经你的许可就可以进行替换。如果省略(F6)s2，那么含有s1串的行都将被删除。

L#,L#?S s1　搜索含有模式s1的行。如果省略"?"，第一个相等将停止该命令。如果包括"?"，则在你需要时继续搜索。含有s1的行变成当前行。

L#W　把L#行写到软盘上。它和L#A一起用于编辑RAM容纳不下的文件。

最后，要退出EDLIN，请用下面的命令之一，这些命令不修改当前行光标。

E　结束编辑并把文本保存在RAM中。

Q　停止编辑，但不保存文本。

为了说明某些最有用的EDLIN命令，请看下面的例子。

1，L	在屏幕上显示整页文本。
5L	显示并编辑第5行，使第5行成为当前行。
，5D	删除从当前行开始的以下5行。
1I	开始在#1行插入正文，用来构成新文件，或者在现存文件顶上插入若干行。要退出该方式，请用CTRL+C或CTRL+BREAK键。

10，20？　用dog替换第10至第20行出现的所有cat。
Rcat（F6）dog

现在，你可以在用语言翻译程序处理前先把BASIC或Pascal程序送入一个文件。如果你打算用BASICA作为语言翻译程序，请看下一节有关BASIC的全面讨论。如果你热心于学习Pascal，就复习一下这里所给的EDLIN命令，并参阅有关Pascal编译程序的那一节。

BASICA解释程序

有很多优秀的BASIC程序设计基础手册和教科书。因此，我们不打算重复别的书中容易查到的内容，而只介绍在IBM个人计算机上完成各种有用工作的程序段组。如果你是个初学的程序员，那么，看了下面的程序段后你可以学会用BASIC编写程序。如果你是个老练的程序员，那么你也可以学到各种有助于你更有效地使用BASIC的设计技巧。但是，在开始进行程序设计之前，你能用BASICA解释程序做些什么呢？

命　令

当在BASICA解释程序状态时，你可以使用下面的命令。而在DOS命令级时，打出BASICA就可以起动BASICA解释程序。一旦开始执行BASICA程序，你就可以把下列命令中的任何一条打到BASICA解释程序，该命令当即被执行。

AUTO	自动产生行号
BLOAD	得到二进制程序并把它装入RAM
BSAVE	保存机器语言程序
CLEAR	置程序变量为0，或空串
CONT	续继中断的程序
DELETE	删除一行或若干行BASIC程序
EDIT	编辑一行BASIC程序
FILES	显示软盘上的文件名称
KILL	擦掉软盘上的文件
LIST	显示部分或所有BASIC程序
LLIST	打印部分或所有BASIC程序
LOAD	把软盘的一条程序复制到RAM中
MERGE	连结两个BASIC程序
NAME	重新命名一个软盘文件
NEW	从RAM擦掉当前程序
RENUM	重新为RAM中的当前程序编号
RESET	关闭所有软盘文件
RUN	运行，或者装入并运行当前程序
SAVE	把RAM程序复制到软盘上

SYSTEM	离开解释程序并返回DOS
TRON	打开跟踪（调试）选择
TROFF	断开跟踪选择

记住，在运行程序前或执行程序时，这些命令可以用键盘来执行，下一组命令实际上是编写程序时必须使用的BASICA语句。我们将给出很多应用的例子，但是首先要弄明白这些语句是什么？

语　句

下面是BASICA直接解释的一张完整的语句表格。这些语句将在后面的例子里作进一步讨论。IBM个人计算机BASIC程序设计手册对这些语句作了深刻解释。

BEEP	机内扬声器发出“嘟”的声音
CALL	执行一个机器语言子程序
CHAIN	执行软盘上另一个BASIC程序
CIRCLE	绘圆，椭圆，楔形、饼形
CLOSE	关闭软盘文件
CLS	清除（擦掉）屏幕
COLOR	选择前景和背景的色彩
COM	允许／禁止通信中断
COMMON	连接程序之间传送的变量
DATA	输入数据值表
DATE $	置日期
DEFFN	定义一个函数
DEFSEG	定义一段存储
DEFUSR	定义USR调用的初始地址

DIM	定义数组的长度
DRAW	根据下面的图象命令绘图

绘图用的图象命令

Un	上移n步
Dn	下移n步
Ln	左移n步
Rn	右移n步
En	对角移（东北）n步
Fn	对角移（东南）n步
Gn	对角移（西南）n步
Hn	对角移（西北）n步
Mn，m	移到（n，m）点或增加一个点（n，m）
Bn，m	同M，但不涂色
Nn，m	同M，但做完后返回原点
An	置90度的倍数角
Cn	为n上色
Sn	置步长（标量因数）n
Xs $	执行含有图象命令的子串S $

END	结束程序
ERASE	从不用的数组收回存贮器
ERROR	抵消模拟的错误条件
FIELD	定义条件记录格式
FOR	重复程序段的FOR循环

GET	从直接存取的文件得到记录
GET	从屏幕取得图象
GOSUB	执行一个子程序
GOTO	跳到另一条语句
IF	下条语句的IF—THEN—ELSE选择
INPUT	从键盘或文件输入数据
KEY	根据所按的键作显示，重新定义，或自陷
LET	对一个变量赋值
LINE	绘一条直线，框或实线框
LINE INPUT	在键盘上输入一整行直线
LOCATE	置光标在屏幕某个位置
LPRINT	在打印机上打出一整行直线
LPRINT USING	格式化打印机输出的行
LSET	把向左对齐的字符串送到文件缓冲器中
MIG $	插入或抽出子串
MOTOR	控制盒式磁带机马达
NEXT	结束一个FOR循环
ON	允许中断条件或多路分支
OPEN	为数据准备磁盘文件或通迅线
OPTION BASE	数组的下标从0或1开始
OUT	把字节从端口输出
PAINT	用颜色填充图象屏幕上的区域
PEN	允许／禁止光笔
POKE	把一个字节值放入RAM中
PRINT	输出到屏幕或文件存贮器

PRINT USING	格式打印
PRESET	用背景图形的颜色绘一点
PSET	用给定的颜色绘一点
PUT	把一个记录写到直接文件
PUT	在屏幕上显示图象
RANDOMIZE	开始一个随机数序列
READ	从DATA语句取得数据
REM	注释
RESTORE	复位DATA语句
RESUME	从错误自陷子程序返回
RETURN	从标准子程序返回
RSET	把向右对齐的串送到文件缓冲区里
SCREEN	选择文本显示或图象显示
SOUND	扬声器发出声音
STOP	仃止执行程序
STRIG	允许／禁止操纵杆按钮
SWAP	交换两个变量的值
TIME $	计时
WAIT	延迟
WEND	WHILE语句循环结束
WHILE	只要条件为真就循环
WRITE	显示或把数据输出到屏幕或文件存贮器

这些命令用来处理RAM或软磁盘文件中所存的数据，但是数据是以多种形式存储的，因此，我们在处理以前必须知道其**类型**，这就是下节的论题。

BASICA的数据类型

类型就是某种值的集合，因此，BASICA 中每一个值都属于一种类型。例如，一个整数值属于**整数型**，因为它的编码就象计算机内存中其他所有整数值一样。

在程序设计中采用有类型的变量，这样可以节省存贮空间，减少执行时间，并使程序更容易理解。我们可以把一种类型看成一种测量单位。例如，要把 3 英尺和15英吋相加，必须把所有的测量单位转换成同一“类型”，我们可以把英尺转换为英吋，于是36英吋加15英吋就得51英吋。在可以做有意义的加法前，我们必须把数据转换成相同的测量单位。

BASICA用下列类型作为存贮数据的手段：

整型（%）　　－32,768到＋32,767的整数

单精度型（！）字长不超过 7 位的实数

双精度型（# ）字长超过 7 位的实数

串变量型（ $）长不超过255个字符的字符串

缓冲区变量（ $）磁盘文件缓冲区存贮，长不超过128字

数组变量型（ $）类型相同的一组数，并可用同一变量名存取。

显然，为了使程序执行算术运算等等，必须能够把一种类型转换为另一种类型。为此，使用了一系列类似于上例中把英吋转换成英尺的类型转换规则。

规则 1：可以用特殊函数：CVI、CVS、CVD 将串变量和缓冲区变量的值转换成数值，这些特殊函数分别回送整数、单精度数、双精度数。

特殊函数MKI $、MKS $、MKD $可以把数值转换成串，这些特殊函数分别将整数、单精度数和双精度数转换成等价的字符串。

规则 2：常数的类型总是转换成含有这些常数的变量的类型。例如，单精度数35.8，当其存贮在变量I%中时将转换成35，因为I%是一个整型变量。

规则 3：表达式总是用最精确的类型来求其值的，在此，最精确的类型是双精度型，其次是单精度型，最后是整型。例如，I%+A!+B#用两步来求值：用单精度完成I%+A!并将结果转换成双精度，再加上B#得出总和。

规则 4：逻辑运算符将其操作数转换成整数，并返回整型结果（实数舍入成整数）。因此，求A%ANDB#的值时是将变量A%和B#中的值转换成16位的整数(舍入成整数）而求得的，如果操作数不能舍入成16位整数，则会出错。

以下是BASICA中某些类型及类型转换的例子。

A!=I%　　将I%中的整数转换成A!中的单精度数。

I%=A!　　将A!中的单精度数转换（舍入）成I%中的整数。

I $=MKI $（I%）将I%中的整数值转换成I $中的双字符串。

I%=CVI（I $）将双字符串I $转换成一个整数。

A#=I%-B!用单精度求I%-B!的值，然后将其转换成双精度数存入变量A#。

上面的类型表包括了两种类型的串。**缓冲区变量**是一种特殊串，它含有等待进行磁盘和RAM之间传送的字符。缓

冲区变量规定了一片RAM区域来保存磁盘信息。在输入期间，缓冲区是用串方式记录磁盘信息的地方；在输出期间，程序将缓冲区作为串处理；在写入磁盘时，该串从RAM中复制到磁盘上。在随后的章节中，我们将对其作进一步的讨论。

数组仅是所有具有同一类型的值的集合。因此，可以把BASICA中的一个数组定义的一块整数、一块精度数、一块双精度数或一块串值。数组用DIM语句来定义：

DIM A%(10)，B!(20)，C#(15)，D$(8)

在这个DIM语句中，我们建立了11个与A%对应的整数，21个与B!对应的单精度实数，16个与C#对应的双精度实数，及9个与D$对应的串。

为了选择数组中的某个值，我们必须用整数作为**下标**。因此，A%(3)代表A%中的第三个整数，D$(5)代表D$中第五个串，而C#(0)代表C#中的第0个数。

由于数组使你能写出只要改变下标值就能对不同值执行同样操作的程序，因此它们是非常有用的数据结构，而且，下标也可由一表达式计算出来，如下所示：

A%(I%+2)=A%(I%* J-3)

数组下标的最小值是零，而最大值受现有存贮容量的限制或是32，767。因此，A%(0)表示A%中的第0个整数值，而A%(5280)表示A%的第5280个整数值。

一个数组最多可有255个下标，如：

DIMX(35，10，3)

它是一个3维数组，其大小为36*11*4等于1584个数，在其中选出一个数，必须给出三个下标：

X(I%, J%, K%)

它选择如下位置的数：

I%*11*4+J%*4+K%

即：X(0, 0, 0)存于位置0，而X(35, 10, 3)在X中的位置是1583。

操作台I/O语句

在键盘、CRT和行式打印机上，你可以用二种语句作I/O操作。无格式语句用起来简单，但不能控制格式。格式语句较复杂，但能格式化屏幕和行式打印机的输出。

PRINT或LPRINT

INPUT

这些语句用于将变量值送到屏幕(PRINT)、打印机(LPRINT)、或接收从键盘输入的值(INPUT)。为了提示或标明某值，在PRINT或LPRINT语句中，可用双引号串。如下例所示：

```
100  PRINT "Enter size=" ; :INPUT S%
200  LPRINT "Enter size=" ; S%
```

该序列在屏幕上显示提示符：

Enter size=

因为100号语句中提示符后跟分号，所以输入值在提示的同一行中读入。冒号是用来分隔两个BASICA语句行的。

接收到S%的值后，由200号语句在打印机上打印出提示以及S%的值。

为了使打印结果整齐，我们可以采用格式化I/O。下面

语句用于格式化I／O：

PRINT USING 或 LPRINT USING

LINE INPUT 或 INPUT

LINE INPUT语句仅用于串值输入。例如要输入Y $的值，可用如下所示带有提示的LINE INPUT；

```
1400 LINE INPUT "Do you want more data?"
```

（你需要更多资料吗？）；Y $

然而，LINE INPUT语句不能用于输入数值，下面举例说明怎样才能既输入串又输入数据：

```
100 LINE INPUT "strike RETURN when ready" ; Y $
200 PRINT "Enter size=" ; : INPUTS%
300 LINE INPUT "Is this correct? " ; Y $
400 LINE INPUT "Enter name=" ; N $
500 LINE INPUT "Is this correct? " ; Y $
```

PRINT USING和LPRINT USING语句让你决定每个输出值占用多少列。PRINT USING用得最多的是格式化数字值。下面是一些例子：

```
100 PRINT USING "##.####" ; A
200 PRINT USING "+##.##" ; B
300 PRINT USING "$$####.##" ; C
```

100行中的##.####图形的输出值保留6列。十进制小数点放在所示位置（小数点后有3位数字）。

200行为正或负号保留一个列。300行表示如何强行打印输出开头的$符号，这两行都强行使输出值舍入到2位十进制数字。

以下是在PRING USING或LPRING USING语句中使用的格式控制符简表：

!	仅打印串中的第一个字符
\n个空格\	打印串中的n＋2个字符
&	变长串
#	数字位置或空格
.	十进制小数点
＋	符号
－	数字后跟负号（跟在数的右端）
**	用星号装填
$$	打印美元符
^^^^	留出指数部分的空间，E＋XX
%	如果数值太大打印%

控制语句

BASICA程序通常是一句一句地执行的。然而，用少量的控制语句可以写出非常灵活而有效的程序。控制语句是控制整个程序流程的语句。它决定下一个被执行的语句。

以下的控制语句用来控制整个BASICA的流程：

STOP，END	程序终止，关闭所有文件并且返回到用户。
FOR-NEXT，WHILE-WEND	程序在一个语句段上循环（重复）。
IF-THEN，IF-THEN-ELSE	控制流程在两种可能的路径中选择一种进行，下次不是

执行THEN语句，就是执行ELSE语句。

GOTO, ON-GOTO, GOSUB, ON-GOSUB

程序执行的下一个语句由GO语句指出。

以下例子解释了这些语句：

读入一个数组值：

```
100  DIM DATA(100)
200  FOR  I%=1 to 100
210       INPUT DATD(I%)
220  NEXT I%
```

完成该操作的另一途径是用WHIE-WEND语句，如下所示：

```
100  DIM DATA(100)
200  I%=1
210  WHILE I%<=100
220  INPUT DATA(I%)
230  I%=I%+1
240  WEND
```

只需增加RETURN语句，我们就可以将一段程序改为子程序。例如：

```
900  FOR I%=1 TO N%
910      PRINT DATA(I%)
920  NEXT I%
930  RETURN
```

每当需要打印DATA中的值时，可用GOSUB语句来

“调用”这段程序。例如

```
100  GOSUB  900
     ...  ...
500  GOSUB  900
     ...  ...
750  GOSUB 900
```

每次执行GOSUB900，执行的是900～930行语句，然后，控制返回到GOSUB后面的语句。所以，上例中在RETURN语句后执行的语句分别是101，551和751。

GOTO和ON—GOTO语句转移到另一个语句，但不从转移返回，故GOTO和ON—GOTO使控制流程永远地转移到程序的另一部分。

以下是几个实用的例子，说明如何使用这些控制语句：

菜单处理程序

```
140  FOR I%=1 to 24 : PRINT : NEXT I% '
     Clear CRT
150  PRIET " [1] . Enter Accounts"
160  PRINT " [2] . Print Checks"
170  PRINT " (3) . Print Bills"
180  PRINT "Enter selection 1, 2, or 3" ; :
     INPUT N%
190  IF ( 1<=N% ) AND ( N%>=3 ) THEN 200
     ELSE 140
200  ON N% GOTO 2000, 3000, 4000
```

菜单处理程序在CRT上显示菜单选择项，用户输入一个代表需要的选择项数，这个数是用N%存贮的，然后根据

N%用的值GOTO 2000，3000或4000。注意，190行中是如何检查有效的N%，如果N%不包含1，2或3，则重复菜单。

文件I／O语句

BASICA中有两种文件；

1、顺序的（打开作为I或O）

2、随机的（打开作为R）

顺序文件是从文件的第一记录开始顺序存取的，为读顺序文件的第N个记录，程序必须存取和读出该文件的前（N—1）个记录。

随机存取文件很象一个数组，因为在存取第N个记录时，程序必须建立一个缓冲区串并且执行一个取或存的操作，然后就可立即找到第N个记录。关于文件结构的详细情况请看第七章，以下各例解释了上述思想。

顺序文件输出

```
1020 OPEN "O", 2, "HEADER.DAT" 'Open #2 for output
1025 PRINT#2, ST$ ", " ;ROOT%; LNG%; M$
1030 CLOSE 2
```

这段序序把单个记录输出到磁盘文件HEADER.DAT，该记录包括ST $，整数ROOT%和LNG%，以及串M $。请注意怎样用分号来分隔记录中的值，并且应特别注意插在ST $和ROOT%之间的“，”，必须用这个逗

号来结束ST $串。

你能写一个程序，它仅仅重复PRINT语句就可输出很多记录。因此，上述的2号文件可存放许多记录。如下所示：

```
1000  OPEN "O", 2, "HEADER.DAT"
1020  FOR I%=1 TO N%
1025      PRINT#2, ST$(I%); ","; ROOT%(I%); LNG%(I%); M$(I%);
          ","
1030  NEXT
1040  CLOSE2
```

同样，为输入上述数值的数组，你必须重复执行一个INPUT#语句：

顺序文件输入

```
2020  OPEN "I", 2, "HEADER.DAT"
2025  FOR I%=1 TO N%
2030  INPUT#2, ST$(I%),ROOT(I%), LNG%(I%), M$(I%)
2040  NEXT I%
2050  CLOSE 2
```

这段程序与输出程序不同，因为在203行中不必用分号或“，”符。这些符号用来严格地把每个记录中的串和数值一一分开，输入程序依靠它们来分隔串和数值。

随机存取输出

随机存取文件必须作为定长度记录的集合而格式化，每

个记录仅包括字符串。随机文件中不允许有数字值！这就是说，我们必须在把数值写入磁盘之前先把它转换成字符串，以后，读出随机文件记录时将字符串再逆转换成数值。

随机文件语句包括如前所述的OPEN和CLOSE语句，再加上一个格式化每个记录（缓冲区变量）的FIELD语句以及在缓冲区和磁盘之间传送信息的GET／PUT语句。

```
1135  OPEN "R", 1, FNAME$
1140  FIELD 1, 127 AS R$
```

这两个语句建立了一个随机（R）文件，编号为（1），称为FNAM$。FIELD语句定义一个缓冲区变量R，其长为127个字符的串。程序将用该缓冲区变量进行磁盘和程序之间的数据传送。例如，假设FLAG%、数组ARC（10）和串数组KEY$（10）都作为单个记录输出，串REC$得到转换后的输出值。

```
180  FOR IN%=1 TO 10
181    CH%=SIZE%*(IN%-1)
182    FLAG$="E"
183  MID$(REC$, CH%+1, 1)=ELAG$
184  MID$(REC$, CH%+2, SIZE%)=KE-
       Y$(IN%)
185  MID$(REC$,CH%+SIZE%+2, 4)=M-
       KS$(ARC(IN%))
186  NET IN%
190  LSET R$=REC$
195  PUT 1, P%
199  RETURN
```

根据一个字符FLAG $值，两个字符的整数数组ARC以及称为KEY $的SIZE%个字符的字符串数组，这个子程序建立了一个串REG $。MID $的功能是将转换值以CH%个字符为间隔插入到REC $中，请注意MKS $的作用是把数字转换成字符形式。

190行的LSET语句将串REC $复制到称为R $的域缓冲区。LSET表示左侧对齐的串。

然后，把缓冲区R $送入195行中的P%号记录。把缓冲区的内容传送到磁盘，这样就把1号文件写到P%号位置上了。

随机文件输入

假设我们已作为“R”文件打开1号文件，并假设上述每个记录已存入，那么下列程序就将数据读回到程序中。

```
130  GET 1, P%
131  LSET REC $=R $ Put into string
132  FOR IN%=1 to 10
133  CH%=SIZE%*(IN%-1)
134  FLAG $=MID $(REC $, CH%+1, 1)
136  KEY $(IN%)=MID $(REC $, CH%+
       2, SIZE%)
137  ARC(IN%)=CVS(MID $(REC $, CH%
       +SIZE%+2, 4)
138  NEXT IN%
140  RETURN
```

请注意：怎样用LSET把缓冲区复制REC $。MID $

和CVS的作用是从缓冲区中抽出串的一部分，然后将串逆转成正确的类型。

程序调试

每当你设计和写完一个程序，下一步将是输入它，并看其是否运行，通常首次运行都不能正确地工作，所以，在程序完善之前，必须找出并修改所有的错误。

为有效地调试新程序，可用BASICA的二条命令。它们是：

TRON进入跟踪状态，它使程序运行中所执行的每个语句标号都显示在CRT上。

TROFF退出跟踪状态。

在某些情况下，错误是无法预知的。因此，程序可能在长时间运行后（在你认为程序能正确工作后）才出错，在程序本身中设置一个错误恢复程序就可来处理这种情况。

```
10  ON ERROR GOTO 3999
```

这一语句使得每当程序发生错误时，就执行3999语句的子程序。退出上述ERROR自陷状态可用：

```
99  ON ERROR GOTO 0
```

它使出错自陷无效，因此，随后发生的错误将象以前一样使程序终止。

内建函数表

名称	含义和例子
ABS	绝对值，ABS（X）
ASC	ASCII码值，ASC（X $）

ATN	反正切（弧度），ATN(X)
CDBL	转换成双精度数，CDBL(X)
CHR $	ASCII码字符，CHR $(X)
CINT	转换成整数，CINT(X)
COS	余弦函数（弦度），COS(X)
CSNG	转换成单精度数，CSNG(X)
CSRLN	光标所在行号
CVI, CVS, CVD	将串转换成数值，CVS(X $)
EOF	文件结束，EOF(2)
ERL, ERR	运行错误的行号和错误码。
EXP	e的x次幂，EXP(X)
FIX	截断整数，FIX(X)
FRE	RAM保留量，FRE(0)
HEX $	作为字符串转换成16进制，HEX $(X)
INKEY $	从键盘输入一个字符，INKEY $
INP	从第I部分读入一个字节，INP(I)
INPUT $	从键盘读入一个串，INPUT $(X)
INSTR	根据X $搜索Y $，INSTR(X $, Y $)
INT	X的整数部分，INT(X)
LEFT $	X $串最左边的I个字符，LEFT $(X $, I)
LEN	X $的长度，LEN(X $)
LOC	返回文件I的下一个记录号，

	LOC (I)
LOF	文件的长度，LOF (1)
LOG	X的自然对数，LOG (X)
LPOS	行印机的位置，LPOS (0)
MID $	从串X $中字符位置I开始返回J个字符，MID $ (X $, I, J)
MKI $, MKS $、MKD $	将数值转换成字符串，MKI $ (X%)
OCT $	将8进制数值作为串返回 OCT $ (X)
PEEK	直接读RAM，PEEK (X%)
PEN	读光笔
POINT	从图形屏幕上读颜色号，
POS	返回CRT上光标位置，POS (0)
RIGHT $	从X $的最右边抽出I个字符， RIGHT $ (X $, I)
RND	回送0到1之间的随机数， RND (X)
SGN	返回数值的符号，SGN (X)
SIN	正弦函数（弧度），SIN (X)
SPACE $	返回空格串，SPACE $ (X)
SPC	在CRT上印出I个空格，SPC (I)
SQR	平方根，SQR (X)
STR $	将X转换成串，STR $ (X)
STRING $	回送同一字符的串， STRING $ (I, X $)

TAB	空I格，TAB(I)
TAN	正切函数（弧度），TAN(X)
USR	调用机器语言子程序，USR(X)
VAL	将串X$转换成数值，VAL(X$)
VARPTR	返回变量地址，VARPR(X)

完整的例子：磁盘分类程序

图6—1是建立随机文件的完整程序，该文件由每个最长不超过WIDE%个字符的字和短语组成。这个程序是一个有用的软件，它能用来输入人名、字典的字等一长串表，并将其按增序（字母序）排列。单词表可在任何时候印出。

然而，该程序的主要目的是为了说明BASICA。因此，应当对照图6—1中的程序仔细阅读下面的说明。下面的行号对应于程序的行号：

10～35 程序的首部，用来说明本程序的用途。含有该程序的文件为FILESORT.BAS，采用方法称为“快速分类”。

50 表格快速分类法用一个叫做STACK%的数组来保存要分类主题词的记录号（起和止），因为这是一种先进后出型的数组，故称之为下推栈。如果要分类的表很长（超过100），怎么，必须将该数组的长度增加到50、100等。

60～85 这是提示用户的菜单，如果J%等于1则控制转移到100号语句；若J%等于2，则控制转移到300号语句等等。如果J%大于4，则控制转移到STOP语句（90行）。

100～105　输入子程序的首部，该子程序用来输入（未排序的）源表，并将其存入磁盘。

110～128　用户必须输入表名及字的最大长度。该名字作为文件名，文件中记录的大小等于WIDE%。

129～135　这是打开一个命名为F $的文件作随机访问的例子，该文件号是1，文件中每个记录的大小设为WIDE%。域语句建立了REC $，作为1号文件的缓冲串。

140　P%是用于存取随机文件的记录号。

145～153　输入的每个字（最长为WIDE%个字符）都存于K $中。

155　把K $中的串复制到串缓冲区，REC $。

160　若用户无串输入，则终止输入程序。

165～170　增加记录指针并且将串缓冲区REC $的内容写到磁盘上的1号文件。

175　返回到153取另一个字K $。

240～250　在做完全部处理工作，修改并关闭文件后，执行此子程序。

260～265　关闭存有字表的文件，并打开另一个名为H $的文件，修改引导文件，引导文件仅包含一个等于表长的整数和一个等于记录大小的整数。

270～275　将表长写入1号顺序文件，然后关闭之。

280　返回到主菜单。

300～325　快速分类法。它按如下方式进行。最初选择一中间字，并用其将表分成两个（近似地）相等的部分，重新调整“左”边的子表，使其中仅包含小于中间字的字；重新调整“右”边的子表，使其中仅包含大于中间字的字。在

左子表中找出大于中间字的字，将其与右子表中小于中间字的字进行交换，当找不到可交换的字时，则中间字将表分为两个子表，将其中一个子表的始末地址存入STACK%，并对另一个子表重复上述处理；选择出一个中间字并交换左，右子表中的字，重复该过程，直到排完所有子表的序列。当排完所有子表的序列后，整个表也就完成了分类。

330～345　读入含有要分类的表的文件名，并确认它是正确的，引导文件名具有扩展名“.HED，”而表名具有扩展名“.DAT”。

350　最初下推栈是空的，因此TOS%为0。

355　输入随机文件的长度N%和记录长度、WIDE%。

360～365　打开存有字表的随机文件。

370～375　为每个工作变量指定WIDE%个字符的存贮空间。

380～450　这是快速分类法。它将开始和结尾的记录号（L%，RIGHT%）存入STACK%，直到需要对其分类为止。

子程序1000保存（L%，RIGHT%），子程序1100用（LEFT%，RIGHT%）将存入的记录号取回，子程序1200完成将字存入适当的子表中所必需的实际交换。

460～470　在分类后，关闭文件并返回到主菜单。

500～525　打印例行程序使你能在分类前或分类后的任何时候输出整个表。

530～570　读入文件名，打开文件，并取出长度N%以及宽度WIDE%。

575～580　开始给K$分配空间，并等待接通打印机。

585～610　打印字表，直接存取1号文件中的每个记录。

615～625　关闭文件并返回到菜单。

700～780　增加表项，类似于输入例行程序。

1000～1035保存记录位置（L%、RIGHT%）的进栈子程序。

1100～1135退栈子程序，当要分类时，回送记录位置（LEFT%，RIGHT%）。

1200～1215交换子程序，在左子表中找出所有大于MIDL $的字，并与右子表中的字交换。

1220～2230初始化变量，并从随机文件中取回中间元素。

1235重复交换直到所有等待的交换完成。

1240～1255找出左子表中大于MIDL $的一个字。存入S $。

1300～1315找出一个右子表中小于MIDL $的字，存入R $。

1322～1350交换S $和R $，并修改记录号指针，所以就再也找不到这对值了。

1400　如果两个搜索指针（左端增加，右端减少）相遇，则交换已经完成。

1410　返回到420句。

```
10  REM ----------------------
15  REM
20  REM                 FILESORT.BAS
```

```
25 REM                    Enter, sort, and print
30 REM                    a list of words using
35 REM                    the Quickersort method
40 REM
45 REM ----------------------------
50 DIM STACK%(100)        'pushdown stack
55 REM--------------restart here --
60 CLS: PRINT"(1). Enter a list of words."
65 PRINT"(2). Sort the list of words."
70 PRINT"(3).Print the list of words."
72 PRINT"(4).Add to existing list."
75 PRINT"(5).Stop"
80 PRINT: PRINT"Enter 1 ... to 5: "; :
        INPUT J%
85 ON J% GOTO 100, 300, 500, 700
90 STOP
100 REM ----------------------------
101 REM            Input a random file of
102 REM            words.Width%=max length
103 REM ----------------------------
104 REM
110   CLS:LINE INPUT"Enter name of
      list: "; F$
115   LINE INPUT"Correct(Y/N)? ":Y$
120   IF Y$<>"Y"AND Y$<>"y"THEN110
125 H$=F$+".hed":F$=F$+".dat"
```

```
126   INPUT"Enter max.ward length : ";
      WIDE%
127   LINE INPUT"Correct ( Y/N ) ? " : Y$
128   IF Y$<>"Y" AND Y$ <> "y"
      THEN 126
129 REM --------open,initialize file ---
130   OPEN"R", 1, F$, WIDE%
135   FIELD 1, WIDE% AS REC$
140   P%=0
141 REM ----------------------
145   CLS : PRINT"Enter ENTER key to
      stop."
150   PRINT"Enter up to"; WIDE%;
      "characters : "
152 REM --- repeat until k$ is null ---
153   LINE INPUT"> "; K$
155   LSET REC$=K$          'fill buffer
160   IF LEN(K$)=0 THEN.250
165   P%=P%+1'direct access location
170   PUT 1, P%     'write to file
175 GOTO 163          'get another word
240 REM ----------------------
145 REM              Write header file
250 REM ----------------------
255 PRINT"Done.Writing header file."
260 CLOSE 1
```

```
265    OPEN "O", 1, H$
270    PRINT#1, P%; WIDE%'save length
275    CLOSE 1
280  GCTO 55                'restart menu
300  REM ----------------------
305  REM
310  REM          Quickersort
315  RE M
320  REM ----------------------
325  REM
330    LINE INPUT"Enter name of list: "; F$
335    LINE INPUT"Corret (Y/N) ? "; Y$
340    IF Y$ < > "Y" AND Y$ < > "y"
       THEN 330
345  H$=F$+".hed": F$=F$+".dat"
350  TOS%=0                 'initial stack
352  REM---read header file ----------
355    OPEN "I", 1, H$: INPUT#1, N%,
       WIDE%: CLOSE 1
358  REM---open word list file --------
360    OPEN"R", 1, F$, WIDE%
365    FIELD 1, WIDE% AS REC$
366  REM--- allocate string space -------
370  R$=SPACE$(WIDE%): S$=SPACE$
     (WIDE%)
375  K$=SPACE$(WIDE%): MIDL$=
```

```
      SPACE$(WIDE%)
378   REM
380   REM---start quickersorting--------
382   REM
385   L%=1: RIGHT%=N%: GOSUB1000 'push
390   REM------ loop here ----------
395   REM
400   GOSUB 1100          'pop left%, right%
405   REM--------- inner loop -------
410   GOSUB 1200          'exchange
420   IF NEWL%<RIGHT% THEN L%=
      NEWL%:GOSUB 1000
430   RIGHT%=NEWR%
440   IF LEFT%<RIGHT% THEN 410
442   REM---keep user informed--------
445   PRINT TOS%/2; "lists to do"
447   REM
450   IF TOS%<>0 THEN 400          'end loop
455   PRINT F$; "is sorted."
460   CLOSE 1
470   GOTO 55           'restart menu
500   REM----------------------
505   REM
510   REM            print list
515   REM
520   REM----------------------
```

```
525R  EM
530  LINE INPUT"Enter name of list : "; F$
535  LINE INPUT"Correct ( Y/N ) ? "; X$
540  IF Y$ < > "Y" AND Y$ < > "y" THEN 530
545 H$ = F$ + ".hed" : F$ = F$ + ".dat"
547 REM --- get file length and width ----
550  OPEN"I", 1 , H$
555  INPUT #1, N%, WIDE%
557  REM --- open direct access file -----
560  CLOSE 1
565  OPEN"R", 1 , F$, WIDE%
570  FIEL 1, WIDE% AS REC$
571  REM ------------------------
575  K$ = SPACE% ( WIDE% )
580  LINE INPUT"Printer ready? "; Y$
585  LPRINT"List="; F$
590  LPRINT : LPRINT
592  REM --access, fetch, and print-----
595  FOR I% = 1  TO N%
600    GET 1 , I% : LSET K$ = REC$
605  LPRINT K$
610  NEXT I%
615  CLOSE 1
620  LINE INPUT"Ready? "; Y$
625  GOTO 55     'restart menu
700    REM ------------------
```

```
705 REM
710 REM    Add to list
715 REM
720 REM ----------------------
725 REM
730   CLS : LINE INPUT"Enter name of
      list : "; F$
735   LINE INPUT"Correct ( Y/N ) ? "; Y$
740   IF Y$ < > "Y" AND Y$ <     > "Y"
      THEN 730
745 H$ = F$ + ".hed". : F$ = F$ + ".dat"
746 REM---get length and width of list-
750 OPEN "I", 1 , H$
755   INPUT # 1 , N%, WIDE%
760 CLOSE 1
763 REM --- prepare to append to list ---
765   OPEN "R", 1 , F$, WIDE%
770   FIELD 1 , WIDE% AS REC$
775   P% = N%
780 GOTO 145    'append
1000 REM ---------------------
1005 REM    Push
1010 REM ---------------------
1015 REM
1020 TOS% = TOS% + 2
1025 STACK% ( TOS% - 1 ) = L%
```

```
1030 STACK%(TOS%)=RIGHT%
1035 RETURN
1100 REM ----------
1105 REM    pop
1110 REM ----------
1115 REM
1120 RIGHT%=STACK%(TOS%)
1125 LEFT%=STACK(TOS%-1)
1130 TOS%=TOS%-2
1135 RETURN
1200 REM ----------
1205 REM    Exchange
1210 REM ----------
1215 REM
1220 NEWL%=LEFT%: NEWR%=RIGHT%
1225 MIDL%=(LEFT%+RIGHT%)\2
1230 GET 1, MIDL%: LSEF MIDL$=REC$
1235 REM ----------repeat-------
1240 REM ----------while--------
1242 REM search for K$ > midl$
1245  GET 1, NEWL%: LSET K$=REC$
1250  IF K$ < MIDL$ THEN NEWL%=
      NEWL%+1 : GOTO 1240
1255 LSET S$=K$     'save k$ for later
1300 RME ----------while--------
1303 RME search for k$ < midl$
```

```
1305  GET 1, NEWR%: LSET K$=REC$
1310  IF K$ > MIDL$ THEN NEWR%=
      NEWR%-1: GOTO 1300
1315 LSET R$=K$
1320 IF NEWL% < =NEWR% THEN 1325
     ELSE 1400
1322 REM ---exchange file records ------
1325 LSET REC$=R$
1330 PUT 1, NEWR%     'put left in rt.
1335 LSET REC$=R$
1340 PUT 1, NEWL%     'put right in lt
1345 NEWL%=NEWL%+1
1350 NEWR%=NEWR%-1
1400 IF NEWL%<=NEWR% THEN 1235
1410 RETURN
1420 REM --- end of folesort ---------
```

图6—1　文件分类程序

图6—2说明该程序是怎样对磁盘上一个极简单的表进行分类的，最初的中间字是DOG（（1+4）／2）。首次交换以左子表中的CAT为结束，而右子表包含BEAR、DOG和EAR。第二次交换结束时已排序子表，因此，交换终止。

作者在计算机上用这个程序对书的索引页进行排序（象本书一样）。字和其页号按原稿中出现的先后次序存入磁盘文件。然后，按字母序进行分类并打印出已排序的文件，在

排序后的输出中，所有重复出现的字都出现在一起，因此，很容易删除多余的字。

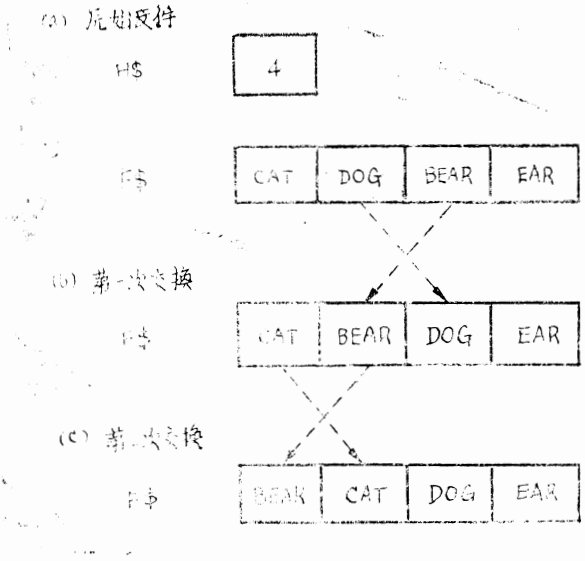


图6—2　以文件F $快速　　一个简表

在IBM计算机上把长度为N的表进行分类，其所需时间约为：

（0.0876）N（log₂N）秒

例如，对52个单字母排序需要18秒，该过程随字的长度以及分类的字数不同而异。

该程序说明了BASICA的许多特性。应记住，对于编译型语言来说，BASICA是解释型语言（立即执行）。下

一节将考察Pascal，它是一种编译型语言。

PASCAL编译程序

Pascal编译程序*是将Pascal语言的源文件翻译成IBM个人计算机的目标文件的程序，我们知道，源文件是某种高级语言的正文文件，而目标文标是某种机器语言（二进制代码）文件。

不象BASICA，用Pascal写的程序在“运行”之前必须翻译成机器语言程序。由于Pascal编译程序太大而不能存入单个软盘片上，因此在IBM个人计算机上要分三个不同的级段完成这个工作。

第一步，用PAS1盘片将Pascal程序翻译成低效率的目标代码程序，这一步将找出在Pascal程序中的语法错误，当然，语法错误是由于Pascal使用不当而引起的。

第二步，用PAS2上的程序优化由PAS1产生的低效率的目标代码，如果程序正确，并成功地通过PAS1和PAS2编译，那么再把产生的机器语言程序存入.OBJ扩展类型的文件中。然后，该文件在其可以独立运行之前，必须与其它“运行时间”程序连接。

第三步，用LINK程序和PASCL.LIB盘片将机器语言文件与Pascal.LIB中的运行时间支持程序进行连接。这一步的目的在于连接执行READ，WRITE，文件输入／输出等的机器代码程序。

---

*需要128K RAM和2个磁盘驱动器。

以下简要地说明这三个步骤以及由此三步所生成的文件，假设进行编译的Pascal程序名是MYPROG.PAS。

第1步：用PAS1将MYPROG.PAS翻成叫MYPROG.BIN的二进制文件。

第2步：用PAS2优化MYPROG.BIN，并生成叫做MYPROG.OBJ的优化目标代码文件。

第3步：用DOS的LINK程序和PASCAL.LIB上的库程序将MYPROG.OBJ与所需要的运行时间支持程序连接起来，生成的文件MYPROG.EXE是一个可执行文件，实际运行该程序时只需输入如下“命令”：

A>MYPROG

上述三步中还生成其它文件，但这是无关紧要的。

显然，在使用Pascal编译程序时，这三步会引起许多不确定性和混乱，为了减少混乱，可建立若干辅助文件，用以自动完成编译任务，这就是下面批处理文件的目的。

PASCAL.BAT文件

让我们创造下列环境，以尽量简化Pascal程序的编译。你可以用EDLIN或COPY命令建立这些文件。

首先，确认编辑程序EDLIN已拷贝到PAS1盘片上，同时为了后面的.BAT文件的应用，要将DOS中的MODE也拷贝到PAS1上。

其次，进行格式化并把装有程序的盘片放入软盘机B，将DOS或PAS1放入软盘须A。

第三，执行以下的批处理文件。

```
A>B：PASCAL MYPROG LPTI：
```

它将执行名为MYPROG.PAS的程序，并列表到行式打印机上，然而，PASCAL是怎样存入磁盘机B上的呢？

为了简便起见，在磁盘机B上如下建立PASCAL.BAT文件。这个文件用到了下面将讨论的另一个叫做LINKPARM的文件。

```
a : mode 80
Rem Compile, Link, and Run a Pascal Program
Rem PASCAL SOURCE LISTING is command Line
Pause.Put PAS1 in drive A, and Your Program diskette in B.
B :
A : PAS1 %1, , %2;
Pause.Pet PAS2 in drive A, and leave Your program diskette in B.
A : PAS2
rename%1.obj result.obj
Pause.Put PASCAL.LIB in A, and leave your program diskette in B.
a : Link b : Linkparm
rename result.obj %1.obj
rename result.exe %1.exe
Pause.Running your program, now.
%1
```

这个批处理文件完成以下工作：

1、屏幕宽度改为80列。

2、将注册驱动器换到B。

3、在源文件（%1）上运行PAS1，并把列表输出到列表文件（%2）上。

4、运行PAS2。

5、为了适应LINK程序，暂时将文件名改为RESULT。

6、将LINKPARM作为LINK程序的输入。

7、将RESULT文件名改回到%1。

8、运行已翻译的程序。

LINK程序的LINKPARM文件包括以下输入值：

```
RESULT
RESULT
NUL
(enter)   ┐
   ⋮      ├ 7次
(enter)   ┘
```

这个文件告诉LINK应将RESULT.OBJ与PASCAL.LIB中的程序连接起来，并产生结果文件RESULT.EXE，不利用映像文件NUL，7个空行把其它LINK参数均假定为缺省值。

当PASCAL.BAT和LINKPARM文件均存在程序软盘上时，你只要执行PASCAL.BAT就可编译任何程序。

一个Pascal个例子

NikLaus Wirth发明的Pascal告诉人们如何使用一个具有良好结构的语言来编程。实际上，良好结构的目的在于

使未编写程序的人能容易地理解程序。这就要求思路清晰并且结构简单。

Pascal程序不象BASIC程序那样容易编写，但Pascal程序非常容易阅读和理解，在数年之后，若要对一个程序进行维护和修改，对程序的理解将比编写程序更重要。

实际上，因为Pascal容易阅读和理解，所以，我们可以通过阅读一个完整的程序来考察该语言。图6—3是一个简单的Pascal程序，它不仅可用来介绍该语言，也可用来将以BASICA输入的程序转换成PASI可接受的输入文件。

图6—3的程序用来去掉程序中语句的行号，假设你用BASICA的屏幕编辑来输入一个Pascal程序。BASICA的屏幕编辑要求程序的每行必须以行号开头。例如，输入程序时可用AUTO来对输入行自动加上行号，但是，必须用Pascal语句而不用BASICA语句。

其次，使用有关编辑程序一节中已讨论过的BASICA屏幕编辑来修改有行号的Pascal程序。在程序编辑之后，将其用ASCII形式存入磁盘：

SAVE "MYPROG", A

这就使有行号的程序被作为正文文件保存。

退出BASICA解释程序，并执行图6—3中的程序，该程序去掉每行开头的行号。因此，仅留下后面可接受的Pascal语句，那么这个过程是如何进行的呢？

在图6—3中，每一行都有一个行号（从1到34），这些行号由编译程序产生并置于列表文件中，如图所示，PASI编译程序不接收行号，然而，列表中的行号可用来区分每个Paseal语句。

1行：program stripper（input，output）这是程序首部，它说明程序的名字及允许的输入和输出设备，该程序名是“stripper”，用键盘作为输入设备（input），并用屏幕作为输出设备（output）。

2—7行：Var语句列出了该程序用到的所有变量数据值在此说明。注意，Pascal中的所有变量名必须出现在单个Var语句中，事实上，由于每个变量都必须具有一定的类型，所以，在此必须定义变量的类型。

两个文件名：diskout和diskin，在第3行定义为正文文件，正文文件是一个按屏幕行组织起来的字符集，每行实际长可达127个字符。

这两个正文文件diskin和diskout分别作为由BASICA生成的stripper输入文件和由stripper产生的输出文件。diskout文件将作为pascal编译程序的输入文件。

```
                                        page    1
                                        01—31—82
                                        07：33：52
JG IC Line  Source Line     IBM Personal
                Computer Pascal Compiler V1.00
  20    1  program stripper (input,output);
  10    2   var
  10    3    diskin, diskout  :  text,
  10    4    infile, outfile  :  string(15);
  10    5    ch  :  char;
```

```
10    6    line  :  lstring(80)        ;
10    7    i  :  integer;
10    8   begin
10    9    page;
11   10    repeat
11   11     write('Enter source file name=');
12   12     readln(infile);
12   13     write('Enter destination file
              name=');
12   14     readin(outfile);
12   15       write('OK-Y/N? ');
12   16       read(ch);
11   17    until (ch='Y') or (ch'y');
11   18    reset(diskin, infile);
11   19    reset(diskin);
11   20    assign(diskout, outfile);
11   21    rewrite(diskout);
11   22    while not eof(diskin) do
11   23     begin
12   24      readln(diskin, line);
12   25      i:=1;
12   26      while line[i] in ['0'..'9']
12   27            do i:=i+1;
12   28      delete(line, 1, i);
12   29      writeln(diskout, line)
11   30    end;
```

```
11    31    close ( diskin ) ;
11    32    close ( diskout ) ;
11    33    writeln ( infile, '--> ,, outfile)
00    34  end.

Symtab  34  Offset Length  Variable
                 0   1390  Return offset,Frame
                                    length
                 2    636  DISKIN
                                  : File    Static
              1306      1  CH
                                  : Char    Static
              1390      2  I
                                  : Integer Static
              1308     82  LINE
                                  : Array   Static
               638    636  DISKOUT
                                  : File    Static
              1274     16  INFILE
                                  : Array   Static
              1290     16  OUTFILE
                                  : Array   Static

STRIPPER
```

图6—3　一个把由BASICA生成的正文转换成PAS1能接收的正文的Pascal程序

4行，我们已说明infile和outfile是字符串。字符串是由键盘字符组成的一个串。string（15）类型限定了每个串的长度为15个字符，当然，串变量infile和outfile中的字符可少于15个字符，这两个串用于保存输入和输出文件名。

第5行中打入的下一个变量是一个称为CH的字符，CH用来保存回答的Y或N。在16—17行中你可以看出这一点。

第6行说明变量行是一个“长度串”，在IBM的Pascal中，列表是一种可以用串功能进行操作的串。串功能可用于插入、删除以及连接字符，在某种意义上说，这种串是可伸缩的。在删除某行行号的期间，这个变量同时可用来存放该行。

第7行说明如何表明i是个整数，它用作行的下标，每次用来取一个字符。

Var语句定义了程序中使用的变量及其类型，但它不将值赋给变量。将值赋给变量是由Pascal程序的“可执行部分”完成的。

8—34行：在图6—2中，8行到34行是可执行部分，指令就放在这个地方。每一个Pascal程序少有一对“begin—end”界线符，它指出一个由多个执行语句组成的“复合语句”。

在第9行中你可以看到一个用PAGE过程来清屏幕（和使行印机走纸）的例子。

PAGE只不过是许多内建函数和过程的一个例子。这些内建函数和过程是PASCAL.LIB程序的基本组成部分，由于它们建立在语言中，故称之固有函数。

10—17行形成程序中的一个循环，该循环重复执行，直到输入Ch的字符是大写或小写的Y为止。该循环的目的在于捕捉正确的输入和输出文件名。

第11—12行使提示" Enter Source file name =" 出现在屏幕上。它们后面是READLN (infile)，可接收长达15个字符的串输入，该字符串存贮在串变量infile中，这是磁盘上一个文件的目录名。

18—21行打开正文文件diskin和diskout以使输入和输出，ASSIGN是固有过程，用于联接infile中的目录文件和文件diskin，在此diskin的用途是重要而微妙的，其过程如下：

每当程序存取一个文件时，只有一小部分读入RAM。这部分存入文件变量中。在本例中，文件变量是diskin和diskout。在Pascal中，有时把这称之为" 窗口" ，因为它们是通向一部分文件的窗口。

RESET固有函数使指定为diskin的文件作好输入准备（参看19行）。RESET用来复位一个已存在的文件，而REWRITE用来建立一个完整的新文件。实际上，21行在建立一个文件时删除已存在的同名文件。

18～21行的目的是为了打开已存在的由BASICA产生的正文文件，并建立和打开用于存放去行号后的正文的输入文件。此时的在图6—3中我们已准备好了每次取一行，去掉其行号，然后把去号后的行存入输出文件。

22—30行是Pascal中的While循环。只要 "not EOF" 为真，就反复执行该循环。也就是说，diskin文件中的每一行正文执行一次该循环。一旦文件结束，EOF固有函数的

值立即为TRUE。因此，每当循环执行完一次23行到30行中的“begin—end”复合语句，都要测试一下“not EOF”。一旦其值为FALSE，则终止循环并执行31行的语句。

23—30行包括一对begin和end关键字。这就是说，这对关键字中的所有语句都被看作一个语句。在Pascal中，象这样由begin—end对构成复合语句，都看作是单个语句。复合语句可以出现在简单语句可出现的任何地方。因此，每次执行While—Loop时就执行了整个复合语句。

24—29行从diskin文件中取一行正文并来其存入串变量行。然后计算行开始处的位数，并来该数字存入变量i中。

26—27行说明如何使用whileloop测试来有效地完成这一工作。测试如下：

```
line[i] in [ '0'.. '9']
```

该行的意思是将line[i]中的第i个字符与‘0’到‘9’集合中的每个字符进行比较，数字‘0’..‘9’两边的方括号表示构成一个集合。该集的元素取自从‘0’到‘9’顺序字符。两点表示“到”。

在27行中每执行一次循环就执行一次该简单语句。i的值仅增加1。

28行说明删除变长串行中的1到i个字符的串固有函数的作用，它能就在这里删去行号。

29行把去行号的行写到磁盘输出文件。它把该行中存贮的串复制到文件窗口磁盘输出文件diskout。随后diskout的输出使当前串强行从窗口输出到磁盘上。

31—32行的两个CLOSE固有函数迫使输出两个窗口中的最后一个串，并标明文件已关闭。最后，33行在屏幕上显

示输入文件名和输出文件名，因此告诉用户能知道除去行号的文件是否已放好。

34行说明如何结束一个Pascal程序，关键字“end”后要跟一个点。

图6—3还列出了与程序有关的一些统计情况。这是一个列出了所有的变量及其类型的符号表，此外，你可看出每个变量在机器中占用了多少RAM。

Pascal程序的格式

我们由图6—3的例子可以导出一些基本的概念。首先，什么是Pascal程序的一般格式？每个Pascal程序由数据段、过程与函数段和可执行指令段组成。数据段包含常数、标号、类型和变量，过程和函数段包含子程序。

```
program任意程序名（输入，输出）；
    （*任选标号*）
    （*任选常数*）
    （*任选类型*）
    （*任选变量*）
    （*任选过程和／或函数*）
begin
    （*任选可执行语句*）
end
```

出现在Pascal中的注解，都要用（* *）括起来。而且，所有任选部分都可由程序员选择，例如，下面是一个有常数和变量部分而没有可执行部分的程序。

```
    program NULL（输入，输出）；
```

```
const
  X=5;  (*整型常数*)
var
  Y:integer;  (*整型变量*)
begin
end.
```

虽然此程序未做任何事情，但它仍为一个完全正确的Pascal程序！请注意，变量和常数可以同属一种类型，但变量的取值可以在一定范围内变化，而常数只能取单一的值。

我们可以用子域符来限定一个整型变量的取值范围，用两个句点'..'把域中的最小值和最大值分隔开来。

```
var
  Y:-15..25;    (*子域*)
```

这表明Y是一个整型变量，且其取值范围仅限定从－15到＋25。

如果Pascal语言仅允许程序员使用有限的数据类型，那它就不会引起人们多大的兴趣。因此，Pascal通过对几种基本数据类型的连接、嵌套，构造出了更为复杂的数据结构。这也许是近25年来程序设计中最巧妙的思想。

Pascal的数据结构

Pascal定义了四种基本的数据类型。之所以把它们称为基本类型是因为所有其它的类型均可以从这四种类型中派生出来。

```
boolean      (*真或假值的集合*)
```

```
integer      (*整数值的集合*)
real         (*实数值的集合*)
char         (*单个字符值的集合*)
```

此外，IBM的Pascal编译程序也接受下面的非标准类型：

```
Word (*16位整数值，无类型*)
```

这是用来进行Parcal系统程序设计的数据类型。从某种意义上说，它摆脱了Pascal的严格的类型限制。对word类型进行的任何操作均产生一个16位的结果。

以下是基本类型变量的例子。

```
var A, B, C      : integer;
    X, Y         : word;
    TI, U, W     : real;
    CH, CA       : char;
    T, F         : boolean;
```

由基本类型向上一步就是结构类型。结构类型可以由基本类型构造出来。

```
array      (*相同类型的值的数组*)
record     (*不同类型的值的集合*)
set        (*作为一个集合的一组值*)
file       (*磁盘文件窗口*)
( ).       (*基本类型的标量枚举*)
```

下面是构造类型的几个例子

```
var
  A : array [1..10] of char;
  B : array [1..5, 0..10] of integer;
```

```
C : record;
        X : real;
        Y : integer;
        Z : char;
    end
```

请注意，数组可以是一维的，也可以是二维的。数组的下标允许在指定的子域内变化，如上面指定的1··10等子域。

上面例子中的记录结构包含三个值，它们分别为实型、整型和字符型。如要访问其中的某一个值，必须使用Pascal中的'.'号。

```
C.X
C.Y
C.Z
```

除此以外，为便于系统编程和字符串处理，IBM的Pascal对结构类型作了一些扩充。

```
string(n)      (*字符串*)
lstring(n)     (*字符串*)
adv            (*某一值的地址*)
ads            (*8086处理器中的段基地*)
```

IBM的Pascal还包含指针和动态存贮器分配。然而，在这里我们将不讨论Pascal的这个特性。

在IBM的Pascal中一个使人感兴趣的语言扩充是所谓的超级数组。这种类型允许将不同的数组传送给过程和函数。例如，由SR定义的超级数组可用来定义一个或多个长度不同的其它数组。

```
type
```

```
SR=SUPER ARRAY[1..*] of REAL;
var
 M : SR ( 10 ) ;  ( * 长度为10 * )
 N : SR ( 3 ) ;  ( * 长度为3 * )
```

让我们看另一完整的例子。该例包含了许多前面提及的概念。记住，Pascal是为了使程序具有更好的可读性而设计的，因此，通过仔细地推敲例题，你会更好地理解这种语言。

另一个完整的例子

图6—4和图6—5各为一个程序，综合两者即可形成一个小的数据库系统。图6—4的DATAENTRY程序用于给一个雇员文件输入信息，而图6—5的RETRIEVE是通过雇员的姓来检索这些信息的程序。你会看到这些程序是怎样工作的，并通过下面的详细分析进一步了解Pascal语言。

首先让我们研究图6—4的DATAENTRY程序。该程序从用户处获取信息，并将这些信息存入软盘上的一个文件。

第1行：这是一个通常的程序首部。

第2—31行：这是一个包含本程序和RETRIEVE程序数据结构的独立文件。如图所示，$INCLUDE元命令把以B：DATA开始的正文插入程序中。该正文从第1行一直到第31行（图中重新编号）。

常数语句说明N和NDEP是整型常数。这些常数用于下面的数据结构中。

类型语句说明为保存某些数据而构造的新类型。要记住，一种类型就是一些值的集合，类型说明语句通知编译程序什么样的值是其所需要的。一个类型说明语句是一种数据结构的“属性单元”。它本身并不容纳任何数据，但却指出以后将容纳什么样的数据。

示于 $INCLUDE 5—7 行的三种新类型分别支持实数和字符串。dates（字符串（*））类型所能容纳的字符串的最大长度为 8 个字符。names类型最多可容纳N个字符，其中N是个常数，并在上面已规定为16。

叫做employees的新类型是一个其它类型的集合体。employees的前三个元素是雇员的“名”，“姓”和“中间名”。它告诉编译程序以后用这些元素保存雇员的名字。$INCLUDE13行所示的元素实际上是另一个集合体。这是数据结构嵌套的一个例子。元素Payroll和history嵌套在employees类型中，而它们又分别含有自己的嵌套元素。

请注意，$INCLUDE17行有一个类型为（married, single, head）的元素。括号内的每一项称为有序类型的标量。也就是说，我们在访问这些项时只利用其名，而不用其值。因此，mstatus 的“值”是 married, single, 或 head。其它的“值”都是不合法的。

employees类型、names类型和其它基本类型的实际变量在 $INCLUDE第26—31行示出。为数据库设置的文件窗口称为dbase。这样，文件将按“employees”类型定义的格式来存贮数据。当然，文件将包括多个记录，而每个记录都将遵从“employees”定义的结构。

DATAENTRY程序还需要一个叫做temp的工作变量

来保存“employees”类型的值。

此外，两个程序都用到了变量fname（文件名）、I、j（工作计数器）和ms（工作字符单元）。

第3—40行：这些行是实际执行数据存取的程序语句。

第4—8行：输入文件名，并为该文件指定一个文件窗口（dbase），这时文件被打开，允许输出操作。第8行语句用写入24个空行将屏幕清除。

第9—44行是一个repeat循环，它一遍又一遍地重复执行，直到用户决定不再给数据库加入新的雇员记录。

这个程序的最重要的特征是其数据结构的访问方式，请注意“.”号。

```
temp. first
temp. last
temp. payroll.rate
```

每个“.”指明了数据结构中的一个嵌套层。所以，temp.first系指temp结构中的first元素。既然元素本身又可含有自己的元素，因此为了检索到结构中所需的层次，有必要使用两个以上的“.”号。

在10—12行中，“.”号仅用来检索temp下面第一层的信息。first，last和middle的值被获取并存入temp。

在13—14行我们做了些新的不同的工作“with”语句是为了消除过多的“.”而采用的速记方法。这样，我们便可用图6—4所示的速记方式代替下面的写法。

```
tmp. payoll. rate
tmp. payoll. hours
tmp. payoll. depends
```

tmp. payoll. mstatus

此外，你一定会注意到21—31行中所做的几件事情。首先，用repeat—until结构来获取mstatus的正确输入值。其次，要注意，因为标量不能输入或输出，所以标量（married, single, head）不能直接输入。而必须用字符编码m，s，或h（或者相应的大写字母）代替。

第23—30行：这是一个扩充的Pascal Case语句的例子。用ms的值来决定执行下面的哪一个成分语句。如果ms的值等于“m”或“M”，则执行第一个成分语句，其它所有的成分语句都被跳过。同样，如果ms的值与其它某成分的标号相符，则执行相应的语句。如果没有一个标号符合ms的值，则执行OTHERWISE子句。

第33—39行每次输入一个temp.history的元素。这里为了缩减“.”号再次使用了with语句。

第41行很有趣味、因为它示出了怎样把temp的全部内容复制到以dbase↑命名的文件窗口中去。注意dbase后面的“↑”号。它表明dbase是文件窗口而不是一个纯粹的文件。一个窗口仅驻留一个记录，而一个文件则保存多个记录。

第42行示出怎样把一个窗口中的值送到磁盘文件上。PUT固有过程输出dbase↑中的文件记录，然后将文件记录指针上移到该文件的下一个记录。这样，再次执行PUT过程时，下一记录就会顺序输出。

第45行在全部记录输入完后关闭文件。这样，employee的信息将永久地保存在软盘上。

图6—5示出怎样写一个Pascal程序，该程序能顺序地检索名为fname的文件，用比较姓字符串的方法找出一个

或多个文件记录。图6—5使用了与先前的程序相同的$INCLUDE文件来定义其数据结构。我们将跳过下面$INCLUDE 1—31行的详细说明。

第4—6行：雇员文件的名字再次被读入fname，并将该名字与叫做dbase的文件窗口相关联。7—37行的repeat—until语句引起的循环，直到用户以姓检索雇员的记录做有后方才停止。每当检索到所需的记录，程序自动关闭文件，然后再重新打开。RESET固有过程与REWRITE固有过程不同，前者用来打开一个现已存在的文件。

在顺序文件情况下（DBASE·MODE：=SEQUENTIAL这里是自动进行的），RESET过程还可以预先取出文件的第一个记录，因此，第9行打开了名为fname的文件。并检索到了它的第一个记录。

第11—32行是当固有过程EOF（文件尾）为假时进行的循环。到达文件尾时EOF过程返回真值。

第13行将存于文件窗口中的姓与用户输入到temp·last中的姓进行比较。如果相等，则由14—31行显示整个记录。注意，这里又用到“with”来缩减“·”号。

第20—24行示出了怎样使用Case语句（不带选择项的OTHERWISE子句）对存贮在payroll.mstatus中的标量值进行“译码”。

第14和32行示出Pascal编译程序怎样通过打印出警告信息来辅助检错。在第14行中，编译程序担心你会改变文件窗口中的内容（如GET（输入）或PUT(输出）不同的值）从而导致错误。但这里的用法是不会引起错误的。32行指明编译程序怎样在END后面插入了“；”号，这里程序的作

者忘记了遵守Pascal的规则——两个相邻语句之间须用“;”隔开。但编译程序有足够的智能去改正程序员的错误。

这个例子说明了Pascal的数据结构，顺序文件的输入/输出，以及case，repeat，while和if—then—else语句。当然这并未包含所有Pascal语言的处理功能。要想有个更全面的了解，读者必须翻阅更多的内容较深的Pascal方面的书籍。

参看下面的功能特性、固有函数和语句表可以了解IBM Pascal的功能及其灵活性。你会看到IBM Pascal已扩充到能够胜任系统编程、应用编程和通讯控制。

```
                                        Page    1
                                        00-00-80
                                        02:31:33
JG  IC Line#   Source Line  IBM Personal
               Computer Pascal Compiler V1.00
    20     1   Progam DATAENTRY(input,
               output);
           2    {$include:'b:data'}
           1    {Data section for DATAENIBY
               and RETRIEVE}
    10     2   const N=16 ; {string lengths}
    10     3         ndep=10 ; {maximum
               dependents}
    10     4   type
```

```
10      5  dollars=real ;
10      9  dates  =string ( 8 ) ;
10      7  names =string ( N ) ;
10      8  employees
10      9          =record
20     10             first : names;
20     11             last : names;
20     12             middle : char;
20     13             payroll : record
30     14              rate : dollars;
30     15              hours : integer;
30     16              depends : 0..ndep;
30     17              mstatus : ( married,
                      single  , head ) ;
20     18             end; { payroll }
20     19             history : record
30     20              hired  : dates;
30     21              boss   : names;
30     22              dept   : names;
30     23              title  : names;
30     24             end { hlstory }
10     25           end    ;{employees}
10     26  var
10     27   dbase : file of emoloyees;
10     28   temp  : employees;
10     29   fname : names;
```

```
10      30   I , j   : integer;      {working
                        variables}
10      31   ms      : char;
10      3   begin { enter an employee record
                    one at a time )
11      4   write(,Enter Employee Information
                    File Name : , );
11      5    readln ( fname ) ;
11      6    assign ( dbase, fname ) ;
11      7    rewrite ( dbase ) ;
11      8    for I : = 1 to 24 do writeln;
             {clear screen}
11      9   repeat
12      10   write ( 'Enter first name : ' ) ;
             readln ( temp.first ) ;
12      11   write ( 'Enter last name : ' ) ;
             readln ( temp.last ) ;
12      12   write ( 'Enter middle initial:' ) ;
             readln ( temp.middle ) ;
12      13   with temp.payroll do
22      14  begin
23      15   write ( 'Enter rate of pay : ' ) ;
             readln ( rate ) ;
```

```
JG IC Line#  Source Line  IBM Personal
             Computer Pascal Compiler V1.00
 23     16      write('Enter hours per week:');
                readln ( hours ) ;
 23     17      repeat  {get correct dependents}
 24     18        write('Enter number depen—
                  dents
                  (O••, ndep : 2 ,') : ') : readl
 24     18  n ( j ) ;
 23     19        until ( O<=j ) and ( j<=ndep ) ;
 23     20        depends : =j;   {correct sub—
                  range}
 23     21        repeat    {get correct ms}
 24     22         write ( 'Enter marital status
                   ( m
                   s, h ) : ' ) ; readln ( ms );
 25     23         case ms of
 25     24          'm','M':mstatus:=married;
 25     25          's','S'   : mstacus : =single;
 25     26           'h','H' : mstatus : =head;
 25     27          otherwise
 25     28          write ( 'Incorrect input. try
                    again.' ) ;
 25     29           ms : ='? ';
 25     30         end{case}
 25     31       until ms < > '';
```

```
22    32        end; {with}
12    33      with temp.historydo
22    34      begin
23    35        write ( 'Enter date hired ( m-
                m/dd
                /yy) : '); readln (hired );
23    36        write('Enter supervisor's last
                name : ' ) ; readln ( boss ) ;
23    37        write ( 'Enter department
                name : ' ) ; readln ( dept ) ;
23    38        write ( 'Enter position title : '
                ) ; readln ( title ) ;
22    39      end; {with}
12    40    writeln ( 'Thankyou' ) ;
12    41    dbase :=temp,{Load buffer with
              record}
12    42    put ( dbase ) :      ( write to disl
              file )
12    43    write('Are you done(Y/N)? ');
              readln ( ms ) ;
11    44  until ( ms='Y' ) or ( ms='Y' ) ;
11    45  close ( dbase ) ;
00    46 end.
Symtab   46    Offset  Length    Variable
                  0      854     Return Offset
                                 Frame Iength
                 848       2     I
```

```
                                    : Integer Static
      850          2                  J
                                    : Integer Static
      852          1                  MS
                                    : Char    Static
      734          98                 TEMP
                                    : Record  Static
        2          732                DBASE
                                    : Filc    Static
      832          13                 FNAME
                                    : Array   Sattic
   Errors   Warns  In  Pass  One
      0        0
```

图 6—4　雇员数据库的DATAENTRY程序

```
                                              Page  1
                                              00—00— 82
                                              04 : 24 : 40
JG  IC  Line#   Source Line  IBM Personal
                Computer Pascal Compiler V1.00
    20      1   Program RETRIEVE ( input,
                output ) ;
            2   { $include : 'data'}
            1   {Data section for DATAENTRY
                and RETRIEVE}
```

```
10    2  const N=16; {string lengths}
10    3     ndep=10;{maximum dependents}
10    4  type
10    5   dollars=real;
10    6   dates  =string(8);
10    7   names  =string(N);
10    8   employees
10    9          =record
20   10             first : names;
20   11             last  : names;
20   12             middle : char;
20   13             payroll : record
30   14             rate  : dollars;
30   15             hours : integer;
30   16             depends : 0..ndep;
30   17             mstatus : (married,
                    single, head);
20   18             end : {payroll}
20   19             history : record
30   20             hired   : dates;
30   21             boss    : names;
30   22             dept    : names;
30   23             title   : names;
30   24             end (history)
10   25           end;{employees}
10   26  var
```

```
10      27  dbase  : file of employees;
10      28  temp   : employees;
10      29  fname  : names;
10      30  I, J   : integer;      {working
                                    variables}
10      31  ms     : char;
10      3   begin
11      4    write ( 'Enter Employee
             Information File Name : ' ) ;
11      5    readln ( fname ) ;
11      6    assign ( dbase, fname ) ;
11      7    repeat
12      8       for I : =1 to 24 do writeln;
12      9       reset ( dbase ) ;
12      10      write ( 'Enter last name of
                employee to find : ' ) ; readln
12      10   ( temp.last ) ;
12      11     while not EOF ( dbase ) do
12      12        begin
23      13         if dbase,last=temp.last
13      14            then with dbase do
```

```
                Computer Pascal Compiler V1.00
          14    --------^warning 208 File
                Dereference Considered Ha
          14    rmful
    23    15        begin
    24    16          writeln( first, ' ',
                      middle, '', last);
    24    17          writeln('Rate$', Payroll
                      .rate : 8 : 2 );
    24    18          writeln('Hours#',Payroll.
                      hours : 8 );
    24    19          writeln('Dependents#',
                      payroll.depends : 3 );
    25    20    case payroll.mstatus of
    25    21         married : writeln('Married');
    25    22         single  : writeln('Single');
    25    23         head : writeln('Head-of-
                       household');
    24    24         end;      {case}
    24    25         with history do
    34    26          begin
    35    27            writeln('Date
                        hired : ', hired);
    35    28            writeln('Supervisor
                        : ', boss);
    35    29            writeln('Departm-
```

```
                                    en ) t : ', dept ) ;
 35      30                         writeln ( 'Position
                                    : ', title ) ;
 35      31             end{with history}
 34      32             end{with dbase}
         32  ---------^warning 164
                    Insert;
 13      33  get ( dbase ) ;
 12      34          end; {while not EOF}
 12      35          close ( dbase ) ;
 12      36  write ( 'Are you done(Y/N)? ' ) ;
             readln ( ms ) ;
 11      37       until ( ms='Y' ) or (ms='Y');
 11      38       write ( 'Thanks' ) ;
 00      39  end.
```

Symtad 39 Offset Length Variable

0	854	Return Offset, Frame Length
848	2	I : Integer Static
850	2	J : Integer Static
852	1	MS : Char Static
734	98	TEMP : Record Static
2	732	DBASE : File Static
832	16	FNAME : Array Static

```
Errors  warns  In  Pass  One
  0       2
```

图6—5　检索一数据库记录

元 命 令

元命令本身嵌入注释中并指示编译程序工作。它们被用作开关检测，错误处理，公式列表和条件编译参数。元命令后跟“+”号表示该命令要被执行，而后跟“—”号表示不需执行。下面仅列出了缺省的开关文件。

$BRAVE+	显示编译错误和警告
$DEBUG+	允许作debug检查
$ENTRY	产生编码以指明一个过程或函数的什么地方出错
$ERRORS：n	设置出错的上限为n
$GO TO—	警告go to可能会带来不好的的结果
$IF $THEN $ELSE	条件编译
$INCLUD：‘name’	将指定为‘name’的文件与当前文件合并
$INDEXCK+	检测数组下标
$INITCK—	将整数初始化为—32768
$LINE—	产生源程序行号以便调试
$LINESIZE：n	设置屏幕宽度（缺省情况下宽度为79）
$LIST+	产生源程序代码清单
$MATHCK	检查运算错误

$MESSAGE：text 在编译时显示信息
$NILCK+ 检测指针的向前引用
$OCODE+ 反汇编目标代码
$PAGE：n 设置页号为n
$PAGEIF：n 如所剩的行数少于n，则跳到下一页
$PAGESIZE：n 设置页的长度（缺省情况为53）
$PAGESIZE：n 设置／恢复元命令的值
$PUSH／$POP 保存／恢复元命令的值
$RANGECK 检查子域值
$RUNTIME 运行时间出错检查
$SKIP：n 跳过n行
$STACKCK+ 检查栈溢出
$SUBTITLE：'text' 设置页的子标题
$SYMTAB+ 列出程序的变量、类型等
$TITLE：'text' 设置页标题
$WARN+ 给出警告信息

数据类型

boolean 真或假
char 字符
real 浮点标
integer −32768··32767
word 16位量
byte 8位量
address ADR，ADS指针
super arrays 字符串用STRING，LSTRING，作为

	参数传递的数组用SUPER ARRAY
pointers	↑，NEW，DISPOSE支持链表处理
set	集合
file	I／O设备
text	专用文本文件
enumerated	标量，如（RED，WHITE，BLUE）
subrange	子域，如100··200

杂项固有过程

ABORT	异常程序终止
BYWORD	将两个字节拼成一字
DECODE	字符—→ASCII码转换
EVCODE	ASCII—→码字符转换
EVAL	仅完成过程的参数求值
LOBYTE，HIBYTE	返回字的一个字节
LOWER，UPPER	一个结构的上界／下界
RESULT	返回函数的结果而不是函数求值
SIZEOF	返回结构的长度
RETYPE	改变结构的类型
MOVEL，MOVER	从一个串的左端或右端移出字符
FILLC	用字符装填一个串

串固有过程

CONCAT	附加串
DELETE	删除子串
INSERT	插入子串

COPYLST	复制串
SCANEQ	检索串（判相等）
SCANNE	检索串（判不等）
COPYSTR	复制串
POSITN	子串定位

时钟固有过程

TIME	赋给一时刻
DATE	赋给一日期
TICS	返回时钟值，该时钟值以每秒18.2（0.055秒）的速率递增

文件固有过程

GET／PUT	文件输入一输出（二进制）
RESET／REWRITE	打开文件
EOF／EOLN	文件结束，行结束读出器
PAGE	下一页
READ／READIN	输入正文
WRITE／WRITEIN	输出正文
ASSIGN	给某文件名指派一个文件窗口
CLOSE／DISCARD	关闭和取消一个文件
SEEK	直接访问

文件方式

TERMINAL	人机对话，显示／键盘方式
SEQUENTIAL	顺序访问方式

DIRECT	随机访问方式

专用控制结构

BREAK	从循环（或嵌套循环）中退出
CYCLE	退出循环并返回到循环的开始
RETURN	退出过程／函数
OTHERWISE	Case语句的缺省子句
AND THEN	AND的延时计算
OR ELSE	OR 的延时计算

以上控制结构需作进一步解释，因为它们均属Pascal的非规则扩充（在有关Pascal的书中找不到）。

下面是一个循环控制的例子。

```
OUT : for I : = 1 to N do
          for J : = 1 to M do
              begin
              ...
              if...then BREAK OUT
              end
```

这个程序说明了怎样用BREAK语句来结束双重循环。

下面是一个使用了cycle语句的相同的例子。

```
OUT : for I : = 1 to N do
          for J : = 1 to M do
              begin
              ...
              if...then CYCLE OUT
              end
```

这次终止了两重循环后控制又转回OUT，使双重循环重新开始。

下面是AND THEN测试的例子。

```
while ( Y<=N ) AND  THEN ( S[Y]<>KEY )
                    do
```

首先计算表达式Y<=N，如果为真，再计算下一表达式；如果为假，下一表达式不再计算。这种控制是很有用的，因为在Y>N时计算S[Y]<>KEY的值可能会引起越域错。有AND THEN操作就可避免这种麻烦。

常见问题

问：写一个程序时可以使用哪几种编辑程序？

答：EDLIN和BASIC EDIT。

问：当存贮一个BASICA文件时，为什么选择ASCII码非常重要？

答：只有ASCII码文件可以合并归类，也只有ASCII码文件才能被EDLIN编辑程序识别。

问：为什么IBM个人计算机使用了BASICA和Pascal两种语言？

答：对快速开发新的程序来说，BASICA是理想的，而对于那些要求速度特别高且内存容量相对较大的程序，则Pascal更为合适。

问：变量I%和I的区别在哪儿？

答：I%为整型变量，只取整数值，I为实型变量，仅取实数值（带十进制小数点）。

问：怎样对一个行式打印机输出？

答：BASCIA中用LPRINT实现，而Pascal则使用WRITELN（PRINTER，…），其中PRINTER是一个TEXT文件，用ASSIGN（PRINTER，LPT'）将PRINTER定义给系统。

问：什么是串？

答：串是一行字符而不是一个数字。

问：怎样将一数字存入一个BASICA随机文件？

答：首先必须用函数MKI $，MKS $，或MKD $将该数字变为一个串，然后用LSET或RSET过程将串装入串缓冲区，最后，则可用PUT过程输出之。

问：怎样用Pascal表示“IF A=0 THEN1010”语句？

答：IF A=0 THEN BEGIN…END ELSE BEGIN…END。

问：在Pascal中怎样打开一个新文件？

答：用REWRITE而不是RSET建立和打开一个新文件。

问：什么是数组？

答：一些有相同类型的元素或值的集合。如，一个串数组是一些串的集合。

问：Pascal中需要行号吗？

答：一般不需要，只有当GOTO或CYCLE过程访问一行时才使用。

问：为什么使用IBM个人计算机做同一件事情可有很多途径？

答：每一个子系统，如DOS,BASIC等等都是独立开发的，因此，每个子系统都能够作一些其它子系统也能做的相同的事情。

问：编译程序的伪指令应该出现在程序的什么地方？

答：Pascal中：LIST--等伪指令（IBM中叫元命令）必须写在源文件开头的注释部分。

问：BASICA中的LSET和RSET有何区别？

答：LSET——向左对齐串

RSET——向右对齐串。

问：为什么一个程序中有不同的数据类型？

答：设置不同的数据类型旨在提高系统的可靠性，灵活性和速度。拿算术运算来说，用整数就比用实数速度快。

第 七 章

数 据 库 处 理 概 述

一切事情都想到了：系统“砰”的一声开始了，然后继续发展。很快出现一大堆问题，有的部分工作太多了，有的太少了。然后解决了一些问题，整个系统开始工作。

事情就是如此，我们要苦于管理系统，当系统的一部分一经建立，系统的其它部分所用的工具可能会出现问题，如果只想有一种简明的管理方法，也许数据库管理系统是我们所需要的……

什么是数据库管理系统（DBMS）？

在第一章我们把数据库管理系统定义为能输入、检索、修改和输出磁盘上信息的计算机软件系统。数据库管理系统的主要作用是简化用户对数据库信息的存取，其方法是通过假定的简化用户数据视图。

第一章说明了三种用户视图，它们是：

网状型；

层次型；

关系型。

上面的第二种和第三种视图是最常用的，本章只描述关系模

型，实际上，我们在这里描述的是如何建立和使用最简单的关系数据库系统。本章第三节列出了几个建立和使用这种简单的关系数据库管理系统的BASIC程序。

关系数据库管理系统中设定该系统的所有数据都可作为表的集合存贮。每个表含有若干行和若干列，如下所示：

Qty	Part No.	Price	Date
5	NC-875	5.95	9/1/81
3	NC-501	0.35	9/30/81
0	NC-450	1.95	9/9/81

表中的列称为关系**定义域**。行称为**元组**。例如上面PARTS关系的一个元组为：

3 Nc-501 0.35 9/30/81

它包含PARTS关系中每个域的值。PARTS关系中的一个定义域如下：

Part No

NC-875

NC—501

NC-450

它含有PARTS关系中每个元组的值。

由上面两个例子看到，要从PARTS关系中选出某个特定的行或列，必须先指定一个或多个定义域为这一关系的**存取键**（access key）。于是，当我们指定“Part No”为PARTS关系中的存取键时，只要确定该存取键的一个值，便可实现对数据库的检索。例如，要检索“Part No”为

NC-501的所有元组，那么这一检索要求检索到的元组如下：

Part No＝NC-501的元组

3 NC-501 0.35 9/30/81

若检索要求较复杂，则可能检索出一个以上的元组，例如，我们要检索出所有大于“NC-500”的“Part No”的值，此时，可得到以下两个元组。

“Part No”＞NC-500的元组

5 NC-875 5.95 9/1/81

3 NC-501 0.35 9/30/81

上面两个元组的集合我们是熟悉的，因为它也是一个关系。因此，在关系数据库系统中，所有询问（要求提供信息）是对一组关系进行运算，而运算结果也是一组关系。关系数据库管理系统的设计思想是通过其它关系产生的关系来简化数据的用户视图，这一思想便叫做关系代数。在关系数据库管理系统中所执行的运算是由高级询问语言所写的关系代数语句实现的。然而，在大多数的微型计算机系统中，数据库管理系统所使用的语言不是以关系代数为基础的高级询问语言，而是菜单式驱动询问语言。因此，在IBM个人计算机中，所使用的数据库系统与本章其它各节完全相同。

数据库管理系统的组成部分

大多数数据库管理系统是由完成下述功能的一组软件构成的：

物理数据库管理　数据库管理系统必须管理一组统一的文件，它可响应用户的每次要求对这组文件进行插入、修改和删除数据。

逻辑数据库管理　数据库管理系统必须能对用户给出一个简明的物理数据库视图，因此，在关系数据库管理系统中，用户只把数据库看为表的集合，而每个表以某种较复杂的文件结构存贮起来的，用户不需知道。这样，这些表、询问代数、菜单等等给了用户一个逻辑上的数据库视图。

询问语言处理器　数据库管理系统必须要使用户拥有浏览、检索、修改、处理和删除信息的能力，一般地，我们为数据库管理系统提供了菜单式询问语言来支持用户的询问。

输入/输出控制　数据库管理系统必须要有一组报告生成程序，这组程序可提供用户把检索到的信息格式化和在屏幕上显示。

我们在下一节讨论本书中的简单关系数据库管理系统时，将说明数据库管理系统的上述各种功能。在IBM个人计算机中可以输入这种简单的数据库管理系统程序，並直接由BASIC*解释。本章最后一节描述了由简单的关系数据库管理系统所采用的物理层的文件结构。

---

* 花10美元便可从作者处买到含有这种简单的DBMS原码表的软盘，若想了解整个数据库的应用，可以购买具有强功能特性的完整的DBMS系统。若需了解实用的关系数据库管理系统，作者在此给你推荐Insoft公司的Portland OR（503）244-4181的数据库设计。

简单的关系型数据库管理系统

这里描述的简单关系型数据库管理系统具有以下特性：

物理数据库管理　系统用二叉树索引文件子结构以建立一组统一的文件，用于物理数据库管理的程序可以在用户的每次要求下扦入、修改和删除一个关系中的元组，並有后备设备维护这组统一的文件。

逻辑数据库管理　系统用CRT屏幕形式定义的关系简化了系统的用法，CRT形式可由用户输入，它可以定义每个关系的定义域，而且可用于输入、检索和输出按元组存贮的信息。

询问语言处理器　系统使用的是很简单的询问菜单，元组用单个存取键（定义域的值）便可检索到。一旦检索到元组，用户可删除它、修改它或把它搁置起来。

输入／输出控制　要在打印机上打印元组和它的屏幕形式，读者可以修改简单的DBMS系统。但这种简单的DBMS系统没有为任何其它输出格式提供格式化的能力。屏幕显示7－1说明了简单的DBMS的功能。当运行BASIC时，装入和运行DBMENU·BAS文件，便可得到这张菜单。

```
OK
LOAD "DBMENU", R
```

这时，主菜单就会出现如屏幕显示7－1。

屏幕显示7－2示出了构成一个关系时必须执行的第一项任务。下面是屏幕显示7－2的说明：

BTREE.DAT　　这是在检索元组的询问中所用到的索引文件名。参看本章最后一节，这一节说明信息较多时，如何使用索引文件来描述物理层。

```
( 1 ).Create a database system.or
( 2 ).Add new records.or
( 3 ).Lookup/Modify existing records.or
( 4 ).Backup file and index.or
( 5 ).Build screen form (for input).or
( 6 ).Quick dump index file.or
( 7 ).Stop.Exit DBMS system.

Enter a number : ? 1
```

屏幕显示 7－1。数据库检索系统的主菜单

```
                Create Index File Header
Enter index file name : BTREE.DAT
      Correct ( Y/N ) ? Y
How many characters per search key ? 16
      Correct ( Y/N ) ? Y
Enter screen form file name : SCREEN.DAT
      Correct ( Y/N ) ? Y
Enter number screen lines in form : ? 6
      Correct ( Y/N ) ? Y
Enter data file name : MASTER.DAT
      Correct ( Y/N ) ? Y
Enter number fields in screen form : ? 8
      Correct ( Y/N ) ? Y
```

幕屏显示 7－2　建立数据库文件系统的示例

16　用一个定义域作为存取键，这个定义域存贮在数据库和索引文件BTREE·DAT中，所以它的长度必须先知道，这里我们限定一个存取键的值为16或小于16个字符。

SCREEN·DAT　定义一个关系（包括定义域）的CRT形式保存在这一文件中。

6　SCREEN·DAT中的CRT形式有6行。

MASTER·DAT

这是关系存贮的位置，每个元组对应于这个文件的一个记录。

8　SCREEN·DAT文件中定义了一个有8个定义域的关系，每个域的值在DAT关系文件中占有一个单一的字段，所以MASTETR的每个记录有8个值。

一旦选择主菜单1建立一个关系，便可设计CRT的显示形式，然后可以输入、修改、删除和检索你所需要的关系中的任何元组。现在我们来看一个实例。

定义一个关系的屏幕形式

屏幕显示7－3和7－4说明如何用输入屏幕形式来定义一个关系。当主菜单选择5输入时，DBMS所要做的工作如屏幕显示7－4所示。

一个**关系**定义为元组（行）的集合，而每个**元组**定义为定义域值的集合，因此，每个定义域值在屏幕显示7－4上为一段有一专门字符开始的白杆（减号）。

*　本字段含有一个存取定义域值

:　本字段含有一个常规定义域值

— 这是值的符号位

对应于屏幕显示7－4的关系如下所示：

ACCOUNTS

Id	Last	First	Street	City	State	Zip	Balance
Lol	Lewis	Ted	Oak	Corv	OR	97330	10.50
Sol	Smith	Joe	Main	SanF	CA	97560	0.0
So2	Smith	Mary	Park	SanF	CA	97660	3.50

图7－1说明在屏幕显示7－4中回答“Y”时所得到的打印输出格式。注意，要改变行时，可输入重新定义的行号，这里不改变行，所以在屏幕显示7－4上输入的是0（零）。

CRT显示形式定义了存贮在关系文件MASTER.DAT中的关系，所有参照这个关系的关系都必须使用CRT的这种显示形式。因此，数据输入、信息检索和DBMS处理都使用这种形式。

```
( 1 ).Create a database system.or
( 2 ).Add new records.or
( 3 ).Lookup/Modify existing records.or
( 4 ).Backup file and index.or
( 5 ).Build screen form (for input).or
( 6 ).Quick dump index file.or
( 7 ).Stop.Exit DBMS system.

        Enter a number: ? 5
```

屏幕显示7－3　选择屏幕格式建立程序

```
Enter form aline at a time.remember :
          .=keyed value
          : =begins a field
          — =field designation
          6 =lines per screen
1 Id : ————
2 Last  .  ———————— First———————————
3 Street :  ————————————————————————
4 City  :  ———————— State : ————————
5                    Zip  : ——————————
6 Balance  : ——————————
Enter 0 to stop.or line number to change : ? 0
Do you want hard copy ?  y__
```

屏幕显示 7－4　数据库检索系统的屏幕显示格式示例

```
                Screen Form SCREEN.DAT
1 Id  : ————
2 Last *  ——————————First : ————————
3 Street  : ————————————————————————
4 City    : ——————————State : ———
5                    Zip : ——————————
6 Balance : ——————————
```

图 7－1　定义屏幕格式的关系

数据输入到一个关系

这一例子中的下一步是把数据输入到关系，该关系的每个元组对应于一幅屏幕的全部值。

当然，屏幕的全部值是由CRT形式得到的。

屏幕显示7－5和7－6说明了如何把信息输入到关系，当提供每个元组的定义域值时，屏幕形式重新显示如屏幕显示7－6所示。

屏幕显示7－6中的每个字段是在屏幕形式提示控制下输入的，例如第4行包含两条提示：CITY和STATE。注意，在第2行输入Last名时会导致一索引操作：

Indexing by LEWIS

这说明元组被索引，因此，元组在下一次可以由它的存取值检索。LAST定义域是一个存取域，因为它是以屏幕显示7－4的形式定义的。（参看屏幕显示7－4第2行中的*）

在元组的所有字段输入到关系后，这个元组便存贮在数据库中，此时，数据库管理一个含有关系的文件（MASTER·DAT）和管理另一个含有存取域（LEWIS等）的值的文件（BTREE.DAT）。下一步的要求是用存取域找出一个或多个元组，並显示它的CRT形式和它的域值。

数据库的询问处理

现在，假定已把几百个元组输入到了上面所述的关系中，这时要想检索其中的一个元组和修改域的值，便要用到查找程序，它使用一个如“LEWIS”的检索键值便可找出一个元组。CRT显示形式及其相应的值如屏幕显示7－7所示。

一旦找到元组，便可立即对它进行以下的操作：

M　用改变一个或多个域值来修改元组。

L　查找另一个元组。

D　删除现有的元组，这就使得元组不能存取了，用后备操作就把所有删除的元组清除掉。

E　查找操作结束后返回主菜单级。

```
1.Create a database system.or
2.Add new records.or
3.Lookup/Modify existing records.or
4.Backup file and index.or
5.Build screen form (for input).or
6.Quick dump indexfile.or
7.Stop.Exit DBMS system.

 Enter a number:? 2
```

屏幕显示7－5．数据输入选择示例

```
1.    Id  :LO1
2.    Last.LEWIS
Indexing by LEWIS
First:TED
3.    Street  :2812 MONTEREY
4.    City    :CORVALLIS
  State :OR
5.    Zip  :9733O
6.    Balance  :1O.95
Inputs stored in file:MASTER.DAT
Do you want to enter more(Y/N)? Y__
```

屏幕显示7－6。屏幕格式控制下的数据输入示例

屏幕显示7－7说明选择操作M时会出现的情况。当输入零时，修改方式终止，若要修改定义域6，则输入6。询

问处理程序有以下提示：

Change OR to ?

示例中的ORE是定义域6（STATE）的新值，这个新值要存在关系中。若没有其它的修改要求，则询问处理程序再次请求执行修改、查找、删除或退出命令。

备用数据库

数据库在经多次修改和删除后，应将这时的数据库复制下来，这样在系统出错时有备用的数据库。这种复制的备用数据库一直要保存到下次再复制备用数据库时为止。上述复制备用数据库的过程保证我们在另一个软盘上总是存有“最大的数据库”。

```
Enter search key value : LEWIS
    Correct (Y/N)? Y
1. Id : LO1
2. Last LEWIs          First : TED
3. Street : 2812 MONTEREY
4. City : CORVALLIS Stae : OR
5. Zip : 9733O
6. Balance : 1O.95

Enter M=modify.L=lookup.D=delete.
      E=exit M
Enter Q=quit.=field to change : ? 6
Change OR to? ORE
Are you sure? Y
Enter Q=quit.=field to change : ? O
Enter M=modify. L=lookup. D=delete.
exit L__
```

屏幕显示7－7　用屏幕格式检索信息示例

```
    ( 1 ). Create a database system. or
    ( 2 ). Add new records. or
    ( 3 ). Lookup/Modify existing records. or
    ( 4 ). Backup file and index. or
    ( 5 ). Build screen form ( for input ). or
    ( 6 ). Quick dump index file. or
    ( 7 ). Stop.Exit DBMS system.
    Enter a number : ? 4

Enter name of backup data file : MASTER.CPy
      Correct ? Y
Enter name of backup index file : BTREE.CPY
      Correct ? Y_
```

屏幕显示7－8　复制备用数据库操作示例

周期性复制备用数据库的第二个原因是消去那些“无用单元”，即，不再需要的信息。删除操作是把元组标为不再需要的信息。因此，数据库的备用付本上除删除的元组外包含所有的元组。我们也可在备用复制操作下重写信息，把它压缩到较小的文件中去，

屏幕显示7－8说明如何把本节所述例子中的关系复制到备用文件中去的。把MASTER.DAT拷贝到MASTER.CPY中，BTREE.DAT拷贝到BTREE.CPY中。在执行复制操作时，要跳过所有标有删除标志的元组。

上述例子说明了DBMS的四个主要组成部分。简单的关系数据库管理系统用CRT屏幕形式和单一的存取域支持一个关系，在较复杂的关系数据库管理系统的多关系中，每个关系有多个存取域，能支持强功能的询问处理。

简单关系数据库管理系统的程序列表

图7－2包含了本章示例的简单关系数据库管理系统的整个BASIC程序表。系统如何工作将在下一节举例说明。

这里列出的是用早期BASIC编制的程序，要使它在你现有的IBM个人计算机上按BASICA执行，还需稍加修改它。这种修改是必要的，因为在BASICA中还存有一点问题，例如在BASIC中，保留关键字是不能用作变量的，不幸的是KEY$与BASICA中的关键字KEY是冲突的，为克服BASICA中的这一缺陷，在下述程序中用KEYS$代替KEY$。（BASICA中的上述缺陷将以“语法错误”显示出来。）

```
100  REM ----------------------
105  REM
110  REM                dbmenu.bas
115  REM    Root program of the DBMS
            demonstration program
120  REM
125  REM ----------------------
130  REM
131  COLOR 11, 1, 4
135  NMAX% = 8
140  CLS : LOCATE 2 : GOSUB 300 : LOCATE 5
145  PRINT SPACE$(10); " [1] . Create a
```

```
      database system, or" : PRINT
150   PRINT SPACE$(10); "[2] : Add new
      records, or" : PRINT
155   PRINT SPACE$(10);"[3] .Lookup/Modify
      exiting records, or" : PRINT
160   PRINT SPACE$(10); "[4] .Backup file
      and index, or" : PRINT
165   PRINT SPACE$(10); "[5] . Build screen
      form(for input), or" : PRINT
170   PRINT SPACE$(10); "[6] . Quick dump
      index file, or" : PRINT
175   PRINT SPACE$(10); "[7] . Change
      current database name, or" : PRINT
176   PRINT SPACE$(10); "[8] . Stop. Exit
      DBMS system." : PRINT
180   PRINT : PRINT SPACE$(25); "Enter a
      number : "; : INPUT N%
190   IF N% >NMAX% OR N%<1 THEN 140
195   ON N% GOTO 200, 205, 210, 215, 220,225,
      230, 400
200           RUN"CREATE"
205           RUN"INSERT"
210           RUN"LOOKUP"
215           RUN"BACKUP"
220           RUN"SCREEN"
225           RUN"DUMP"
```

```
229 REM ------ Change databases-----
230 CLS : LOCATE 10
235 LINE INPUT"Enter new database name : ";
    DBNAME$
240 I% = INSTR(DBNAME$, ".")
245 IF I% <> 0 THEN 255
250 DBNAME$=DBNAME$+".DAT"
255 OPEN"0", 2, "CURRENT.DAT"
260 PRINT#2, DBNAME$
265 CLOSE 2
270 GOTO 140
299 REM ----Print current database----
300 ON ERROR GOTO 335
301 OPEN "I", 2, "CURRENT. DAT",
302 ON ERROR GOTO 0
305 INPUT #2, HEADER$
310 CLOSE 2
315 I% = INSTR(HEADER$, ".") - 1
320 DBNAME$ = LEFT$ (HEADER$, I%)
325 PRINT SPACE$(20); "Current database
    name is"; DBNAME$
330 RETURN
335 IF ERR <>53 THEN 355
340 OPEN "0", 2, "CURRENT.DAT"
345 PRINT #2, "<none>."; ", "
350 CLOSE 2 : GOTO 360
```

```
355 ON ERROR GOTO 0
360 RESUME
399 REM ---------- Finish up -----
400 COLOR 14, 9, 2 : CLS
500 END
```

图7.2a. DBMS程序主菜单

```
100 REM ---------------------
105 REM
110 REM            create.bas
115 REM   Create a database from information
          about the files
120 REM
125 REM ---------------------
130 GOTO 340
140 REM ----------------------
150 REM   Write( p%, flag%, keys$, arc%,
          link% )
160 REM ---------------------
170 REC$ = SPACE$ ( 127 )
180 FOR INDEX% = 1 TO N%
190   CH% = SIZE% *( INDEX% - 1 )
200   ON FLAG% ( INDEX% ) + 1 GOTO 210,
      230, 250
210       FLAG$ = "E" : GOTO 260
220       REM
```

```
230         FLAG$ = "F" : GOTO 260
240         REM
250         FLAG$ = "D"
260     MID$(REC$, CH% + 1, 1) = FLAG$
270     MID$(REC$, CH%+2, SIZE%-3)
        =KEYS$(FNDEX%)
280     MID$(REC$, CH%+SIZE%-1, 2)
        MKI$(ARC%(INDEX% ))
290   NEXT INDEX%
300   MID$(REC$, 126, 2) =
      MKI$(LINK%)
310   LSET R$ = REC$
320   PUT 1, P%
330   RETURN
340   REM ----------------------
350   REM    Create : Make header file for
             b-tree index
360   REM ----------------------
370   CLS : LOCATE 2
375   PRINT SPACE$(30); "Create new
      database" : LOCATE 5
380   LINE INPUT"Enter new database name :
      "; HEADER$
385   LINE INPUT"   Correct(Y/N)? "; Y$
390     IF Y$ < > "Y" AND Y$ < > "Y" THEN
      380
```

```
395 LINE INPUT"Enter index file name :
    "; INDEX$
400 LINE INPUT"    Correct( Y/N )? "; Y$
410     IF Y$ < > "Y" AND Y$ < >  "Y"
        THEN 395
411 I% = INSTR( INDEX$, "." )
412 IF I% < > 0 THEN 420
413 INDEX$ = INDEX $+ ".DAT"
420 PRINT "How many characters per search
    key"; : INPUT SIZE%
430 LINE INPUT"    Correct( Y/N )? "; Y$
440     IF Y$  "Y" AND Y$  "Y" THEN 420
450 LINE INPUT"Enter screen form file name
    : "; FSCREEN$
460 LINE INPUT"    Correct( Y/N )? "; Y$
461 I% = INSTR( FSCREEN$, "." )
462 IF I% < > 0 THEN 475
463 FSCREEN$ = FSCREEN$ + ".DAT"
470       IF Y$ < > "Y" AND Y$ < >  "Y"
          THEN 450
475 PRINT "Enter number screen lines in form
    : "; : INPUT LINS%
480 LINE INPUT"  COrrect( Y/N )?"; Y$
485   IF Y$ < > "Y" AND Y$ < > "Y" THEN
      475
490 LINE INPUT "Enter date file name : " :
```

```
    MAST$
495 LINE INPUT"    Correct (Y/N)?": Y$
496 I%=INSTR(MAST$, ".")
497 IF I% <> 0 THEN 505
498 MAST$=MAST$+".DAT"
500         IF Y$ <> "Y" AND Y$ <> "Y"
             THEN 490
505 PRINT "Enter number fields in screen form
    : "; : INPUT AN%
510 LINE INPUT"    Correct(Y/N)?"; Y$
515 IF Y$ <> "Y"AND Y$ <> "Y" THEN 505
520 REN----------------------
525 REM            Write header file
530 REM----------------------
535 SIZE%=SIZE%+3
536 N%=INT(126/SIZE%)
537 I%=INSTR(HEADER$,".")
538 IF I% <> 0 THEN 541
540 HEADER$=HEADER$+".DAT"
541 OPEN "0", 2, "CURRENT.DAT"
542 PRINT #2, HEADER$
543 CLOSE 2
545 OPEN "0", 2, HEADER$
550 ROOT%=1
555 LNF%=1 : LNG%=0
560 P% = ROOT%
```

```
565 PRINT #2, FSCREEN$; \", "; ROOT%;
    LNG%; LNF%; AN%; LINS%; SIZE%;
    INDEX$; \", "; MAST$
570 CLOSE 2
575 REM ----------------------------
580 REM          Write first root node
585 REM ----------------------------
590 DIM FLAG%(N%+1), KEYS$(N%+1),
    ARC%(N%+1)
595 ZERO$=SPACE$(SIZE% -3); LSET
    ZERO$ = \"0"
600 FOR IO% = 1 TO N%
605   KEYS$(IO%)=SPACE$(SIZE% 3)
610   FLAG%(IO%) = O; KEYS$(IO%)
      = ZERO$ : ARC%  (IO%)=O
615 NEXT IO%
620 LINK% = 0
625 OPEN \"R", 1, INDEX$
630 FIELD 1, 127 AS R$
635 GOSUB 140
640 CLOSE 1
645 REM ----------------------------
650 REM          All done, return to dbmenu.bas
655 REM ----------------------------
660 RUN \"DBMENU"
665 END
```

图 7.2b. 新建数据库的初始化程序

```
1     REM ----------------------
2     REM
3     REM                insert.bas
4     REM        Add to the contents of a date
                 file thru its index
5     REM
6     REM ----------------------
10    REM
15    S0%=20 : DIM STACK%(S0%) : GOTO
      1100
20    REM
25    REM        Subroutines used :
30    REM             100, 150 : read, write a
      node of b-tree
35    REM             200, 250 : save,restore copy
                      of b-tree node
40    REM             300, 350, 395 : push, pop,
                      init the stack
45    REM             400 : shift items in node for
                      splitting node
50    REM             500 : search down b-tree
55    REM             600 : allocate more space fo
                      rb-tree
60    REM             700 : split b-tree node into
                      left and right nodes
65    REM             800 : over flow b-tree node
```

```
                    to root node
70  REM             900 : insert a new item into
                    b-tree
75  REM             1000 : close all files and
                    finish up
80  REM
85  REM
90  REM
100 REM ----------------------------
105 REM
110 REM                 read.bas
115 REM         Input a b-tree node from disk
                file #1
120 REM
125 REM ----------------------------
130 GET 1, P% : LSET REC$ = R$
131 FOR INDEX% = 1 TO N%
132         CH% = SIZE% * ( INDEX% - 1 )
133         FLAG$ = MID$ ( REC$. CH% +
            1, 1 )
134             IF FLAG$ = "E" THEN
                 FLAG% ( INDEX% ) = 0
135             IF FLAG$ = "F" THEN
                FLAG% ( INDEX% ) = 1
136             IF FLAG$ = "D" THEN
                FLAG% ( INDEX% ) = 2
```

```
137         KEYS$(INDEX%) =
            MID$(REC$, CH%+2, SIZE%-3)
138         ARC%(INDEX%) = CVI(MID$
            (REC$, CH% + SIZE%-1, 2))
139 NEXT INDEX%
140 ARC%(N% + 1) = CVI(MID$(REC$,
    126, 2))
145 RETURN
149 REM
150 REM ---------------------
155 REM
160 REM                write.bas
165 REM    Output a b-tree node to file #1
170 REM
175 REM ---------------------
177 REC$ = STRING$(127, " ")
180 FOR INDEX% = 1 TO N%
181         CH% = SIZE% * (INDEX%-1)
182         ON FLAG%(INDEX%) + 1 GOTO
            183, 184, 185
183           FLAG$ = "E" :GOTO 186
184           FLAG$ = "F" :GOTO 186
185           FLAG$ = "D"
186         MID$(REC$, CH% + 1, 1) =
            FLAG$
187         MID$(REC$, CH% + 2, SIZE%
```

```
        -3) = KEYS$
        (INDEX%)
188     MID$(REC$, CH% + SIZE% -
        1, 2)=MKI$(ARC%(INDEX%))
189 NEXT INDEX%
190 MID$(REC$, 126, 2)= MKI$(ARC%
    (N% + 1))
195 LSET R$= REC$ : PUT 1, P%
199 RETURN
200 REM ----------
201 REM
202 REM              Save a b-tree node
203 REM ----------
210 FOR INDEX% = 1 TO N% + 1
212     SFLAG%(INDEX%) = FLAG%
        (INDEX%)
214     SKEYS$(INDEX%)= KEYS$(I
        NDEX%)
216     SARC%(INEEX%) = ARC%
        (INDEX%)
218 NEXT INDEX%
220 RETURN
250 REM ----------
251 REM
252 REM              Restore a b-tree node
253 REM ----------
```

```
260 FOR INDEX%=1 TO N% + 1
262         FLAG% ( INDEX% ) = SFLAG%
            ( INDEX% )
264         KEYS$ ( INDEX% ) = SKEYS$
            ( INDEX% )
266         ARC% ( INDEX% ) = SARC%
            ( INDEX% )
268 NEXT INDEX%
270 RETURN
300 REM ------------------------
301 REM
302 REM                 Push
303 REM ------------------------
304 REM       S
310 IF TS% <=S0% THEN 330
315       D$ = "Stack overflow"
320       RETURN
330 STACK% ( TS% ) =A% : TS%=TS%+1
340 D$ = " " : RETURN
350 REM ------------------------
351 REM
352 REM                 Pop
353 REM ------------------------
360 TS% = TS% - 1
365 IF TS% > 0 THEN 380
370       D$ = "Stack underflow"
```

```
375       RETURN
380  A% = STACK%( TS% )
385  D$ = "" : RETURN
395  REM ----------------------------
396  REM
397  REM                    Initialize stack
398  REM ----------------------------
399  TS% = 1 : RETURN
400  REM ----------------------------
401  REM
402  REM                Shift b-tree node
403  REM
044  REM ----------------------------
405  REM
410  SPLIT% = INT( ( N%+1 )/2 )
415  I% = 1
420  IF  SPLIT%+I% = N% THEN 425 ELSE 450
425       ARC%(I% ) = ARC%( SPLIT%+I% )
430       KEYS$(I%) = KEYS$(SPLIT%+I%)
435    FLAG%( I% ) = FLAG%( SPLIT%+I% )
40        I% = I% + 1
4445  GOTO  420
450  ARC%( I% ) = TEMP%
455  KEYS$( I% ) = ZERO$
460  FLAG%( I% ) = 0
465  REM ----------------------------
```

```
466 REM                    Zero out remaining
                           items in node
467 REM ----------------------------
470 FOR I%=I%+1 TO N%
475     ARC%(I%)=0
477     KEYS$(I%)=ZERO$
480     FLAG%(I%)=0
485 NEXT I%
490 GOSUB 600              'allocate dish space at
                           p2%
495 SWAP P%, P2%
496 GOSUB 150              'write right son to dish
497 SWAP P%, P2%
499 RETURN
500 REM ----------------------------
501 REM
502 REM                    Search b-tree for k$
503 REM
504 REM ----------------------------
505 REM
506 D$ = " "               'message
510 GOSUB 395              'initialize stack
515 P% = ROOT%
520 REM Repeat until found or not-in-file
525     I% = 1
530     GOSUB 100
```

```
535       IF KEYS$(I%)=ZERO$ THEN 545
540       IF KEYS$(I%)<K$ THEN 542
          ELSE 545
542       I%=I%+1                    :GOTO 535
545       A%=P% :GOSUB300            'push node
                                     number
550       A%=I% :GOSUB 300           'push item
                                     number
555  P% = ARC%(I%) : IF P%<=0 THEN
     RETURN
560  GOTO 520
600  REM ----------------------
601  REM
602  REM                  Allocate more disk
                          space for b-tree
603  REM
604  REM ----------------------
605  REM
610  D$=""+LNF%=LNF%+1
620  P2% = LNF%
630  RETURN
700  REM ----------------------
701  REM
702  REM                  Split a b-tree node into
                          left and right nodes
703  REM
```

```
704 REM ----------------------
705 REM
710 GOSUB 200
715 GOSUB 400
720 GOSUB 250
725 K$ = KEYS$(SPLIT%)
730 FOR I% = SPLIT% + 1 TO N%
731       KEYS$(I%) = ZERO$
732       FLAG%(I%) = 0
733       ARC%(I%) = 0
740 NEXT I%
745 ARC%(N% + 1) = P2%
750 GOSUB 150
790 RETURN
800 REM ----------------------
801 REM
802 REM                Overflow
803 REM
804 REM----------------------
805 REM
810 GOSUB 700 : P0% = P%
820 GOSUB 350 : ITEM% = A%
825 GOSUB 350 : P% = A%
830 IF D$ = "Stack underflow" THEN 835
    ELSE 880
835       FLAG%(1) = 1 : KEYS$(1) =
```

```
            K$ : ARC%(1) = PO%
840         FLAG%(2) = 0 : KEYS$(2) =
            ZERO$ : ARC%(2) = P2%
845         FOR I% = 3 TO N%
850             FLAG%(I%) = 0
851             KEYS%(I%) = ZERO$
852             ARC%(I%) = 0
855         NEXT I%
860         ARC%(N% + 1) = 0
865         GOSUB 600 : P% = P2%
870         GOSUB 150 : ROOT% = P%
875         D$ = "Done" : RETURN
880 REM ------------------------------
885 GOSUB 100              'read parent node
890 ARC%(ITEM%) = P2%
895 D$ = "Not done"
899 RETURN
900 REM ------------------------------
901 REM
902 REM                    Insert new item in
                           b-tree
903 REM
904 REM ------------------------------
905 REM
910 GOSUB 500          'search
920 GOSUB 350 : ITEM% = A%      'pop
```

```
925 GOSUB 350 : P% = A%          'pop
930 IF K$ = KEYS$(ITEM%) THEN 931
    ELSE 940
931     D$ = "Found" : PRINT Already
    indexed"
932     LINE INPUT "Strike return to
        continue"; Y$
933     RETURN
940 REM ----------------------
945 TEMP% = ARC%(N%)
950 FOR I%=N% TO ITEM%+1 STEP(-1)
955     ARC%(I%) = ARC%(I%-1)
960     KEYS$(I%)=KEYS$(I%-1)
965     FLAG%(I%)-FLAG%(I%-1)
970 NEXT I%
975 ARC% (ITEM%)=PO%
976 KEYS$(ITEM%)=K$
977 FLAG% (ITEM%)=1
978 REM --------insert done--------
980 IF KEYS$(N%) = ZERO$ THEN 990
    ELSE 995
990     GOSUB 150 : RETURN  'rewrite node
995 GOSUB 800 : IF D$ <> "done" THEN
    940 'ascend b-tree
999 RETURN
1000 REM ----------------------
```

```
1001 REM
1002 REM                    Finish up
1003 REM
1004 REM  ----------------------------
1005 REM
1010 CLS
1015 CLOSE 1, 2
1020 OPEN "O", 2, HEADER$
1025 PRINT #2, FSCREEN$; "."; ROOT%;
     LNG%; LNF%; AN%; LINS%; N%;
     SIZE%; INDEX$; ", "; MAST$
1030 CLOSE 2
1035 RETURN
1100 REM  ----------------------------
1101 REM
1102 REM                    Capture date from
                            screen form
1103 REM
1104 REM  ----------------------------
1105 REM
1106 CLS
1107 OPEN "I", 2, "CURRENT.DAT"
1108 INPUT #2, HEADER$
1109 CLOSE 2
1110 OPEN "I", 2, HEADER$
1115 INPUT #2, FSCREEN$, ROOT%,LNG%,
```

```
    LNF%, AN%, LINS%, N%, SIZE%,
    INDEX$, MAST$
1120 CLOSE 2
1125 NO%=NO%+1 : DIM FLAG%(NO%),
    KEYS$(NO%), ARC%(NO%)
1130          DIM SFLAG%(NO%),
              SKEYS$(NO%), SARC%
              (NO%)
1135 OPEN "I", 2, FSCREEN$
1140 FOR L% = 1 TO LINS% : INPUT #2,
    RW$(L%) : NEXT L%
1145 CLOSE 2
1150 OPEN "R", 1, INDEX$
1155 FIELD 1, 127 AS R$
1160 REC$=SPACE$(128) : ZERO$=SPACE$
    (SIZE%-3) : LSET ZERO$ = "0"
1165 K$ = SPCAE$(SIZE% - 3)
1170 OPEN "R", 2, MAST$
1175 FIELD 2, 127 AS MR$
1179 REM --------forms input--------
1200 DIM AN$(AN%)          'answers in an$
1210 K% = 0
1215 PRINT
1220 FOR L% = 1 TO LINS%
1225       SRW$ = RW$(L%) 'save form,
           prompt
```

```
1230    PRINT USING "##"; L%; :
        PRINT".";
1235    IF INSTR(LEFT$(RW$(L%),
        ϕ1), "-")= 1 THEN 1240
1236    IF INSTR(LEFT$(RW$(L%),
        1), " ")= 0 THEN 1250
1240        RW$(L%) = RIGHT$(RW$
            (L%), LEN(RW$(L%))-1)
1245        GOTO 1235
1250    STAR%=INSTR(RW$(L%), "*")
1255    J% = INSTR(RW$(L%), ":")
1260    IF STAR% = 0 THEN 1270
1265        IF STAR% < J% THEN 1295
1270    IF J% = 0 THEN 1295
1275        PRINT " "; LEFT$(RW$
            (L%), J%);
1280        K% = K% + 1
1281        RW$(L%)=RIGHT$(RW$
            (L%), LEN(RW$(L%))-J%)
1285        LINE INPUT AN$(K%)
1290        GOTO 1235
1295    J%=INSTR(RW$(L%), "*")
1300    IF J%=0 THEN 1340
1305        PRINT " "; LEFT$(RW$
            (L%), J%);
1310        K%=K%+1
```

```
1311          RW%(L%)=RIGHT$(RW$
              (L%), LEN(RW$)L%))-J%)
1315          LINE INPUT AN$(K%) :
              K$=""
1320        K$ = LEFT$(AN$(K%), SI
            ZE% - 3)
1325        LNG%=LNG%+1 : P0%=-LNC%
1330        PRINT "Indexing by"; K$
1335        GOSUB 900 : GOTO 1235 'insert K$
            P0% into b-tree
1340        RW$(L%)=SRW$ 'restore rw$
1342        IF D$="Found" THEN 1230 'try
            again
1345 NEXT L%
1350 TR$ = STRING$(127, ":") : I1% = 1
1355 FOR I% = 1 TO AN%
1360        I2%=I1%+LEN(AN$(I%)) - 1
1365        MID$(TR$,I1%,I2%)=AN $(I%)
1370        I1% = I2% + 2
1375 NEXT I%          'pack answers into tr$
1380 LSET MR$ = TR$
1385 PUT 2, LNG%    'write random record
1390 PRINT "Inputs stored in file : "; MAST$
1395 REM ------DO IT AGAIN ?------
1400 LINE INPUT "Do you want to enter more
     (Y/N)?"; Y$
```

```
1405 IF Y$ = "Y" OR Y$ = "y" THEN 1210
1410 GOSUB 1000
1415 RUN "DBMENU"
1420 END
```

图 7.2c. 在一个数据库关系(表)
中插入一新行的程序

```
1    REM ----------------------------
2    REM
3    REM                lookup.bas
4    REM        Retrieve/Change information
                in database
5    REM
6    REM ----------------------------
7    REM
10   GOTO 1100
20   REM -------- Subroutines --------
30   REM
100  REM ----------------------------
105  REM
110  REM                read.bas
115  REM        Input a b-tree node from
                disk file #1
120  REM
125  REM ----------------------------
```

```
130 GET 1, P% : LSET REC$ = R$
131 FOR INDEX% = 1 TO N%
132       CH% = SIZE% * ( INDEX% - 1 )
133       FLAG$ = MID$ ( REC$, CH% + 1, 1 )
134          IF FLAG$ = "E" THEN FLAG%
             ( INDEX% ) = 0
135          IF FLAG$ = "F" THEN FLAG%
             ( INDEX% ) = 1
136          IF FLAG$ = "D" THEN FLAG%
             ( INDEX% ) = 2
137       KEYS$(INDEX%) = MID$ ( REC$,
          CH% + 2, SIZE% - 3 )
138       ARC% ( INDEX% ) = CVI ( MID$
          ( REC$, CH% + SIZE% - 1, 2))
139 NEXT INDEX%
140 ARC% ( N% + 1 ) = CVI ( MID$ ( REC$,
    126, 2 ) )
145 RETURN
149 REM
150 REM  ----------------------
155 REM
160 REM                    write.bas
165 REM        Output a b-tree node to file #1
170 REM
175 REM  ----------------------
177 REC$ = STRING$ ( 127, " " )
```

```
180 FOR INDEX% = 1 TO N%
181       CH% = SIZE%* (INDEX% - 1)
182       ON FL AG%(INDEX%) + 1 GOTO
          183, 184, 185
183          FLAG$ = "E" : GOTO 186
184          FLAG$ = "F" : GOTO 186
185          FLAG$ = "D"
186       MID$(REC$, CH%+1,1)=FLAG$
187       MID$(REC$, CH%+2, SIZE%-3)
          = KEYS$(INDEX%)
188       MID$(REC$, CH%+SIZE%-1, 2)
          = MKI$(ARC%  (INDEX%))
189 NEXT INDEX%
190 MID$(REC$, 126, 2)=MKI$(ARC%
    (N% + 1))
195 LSET R$ = REC$ : PUT 1, P%
199 RETURN
500 REM ----------------------
501 REM
502 REM          Search for k$ in b-tree
503 REM
504 REM ----------------------
505 REM
515 P% = ROOT% : D$ = ""
520 REM --repeat until found or not in file-
525       I% = 1
```

```
530       GOSUB 100          read node
535       IF KEYS$(I%) = ZERO$ THEN
          545
540       IF KEYS$(I%) < K$ THEN 542
          ELSE 545
542          I% = I% + 1 : GOTO 535
545       A% = P%
550       ITEM% = I%
552       P% = ARC%(I%)
555  IF P% < = 0 THEN 556 ELSE 520
556       IF KEYS$(ITEM%) < > K$ THEN
          565
557        F FLAG%(ITEM%) < > 2 THEN
            563    'may be deleted
558           PRINT "Key was delated..cannot
              retrieve it."
559           LINE INPUT "Do you want to
              restore it?"; Y$
560              IF Y$ < > "Y" AND Y$ < >
                 "y" THEN 567
561       FLAG%(ITEM%) = 1
562       P% = A% : GOSUB 150 : RETURN
563       P% = A% : RETURN , founb it !!!
565       PRINT "Key not found..cannot
          retrieve it."
567       D$ = "Not found"
300
```

```
750 RETURN
800 REM ------------------------------------
801 REM
802 REM        get and unpack data file
803 REM
804 REM ------------------------------------
805 REM
810 GET 2, - ARC%(ITEM%)
840 LSET TR$ = MR$ : I1% = 1
850 FOR I% = 1 TO AN%
860       I2% = INSTR(TR$, ":")
865       AN$(I%) = SPACE$(I2% -
          I1%)
870       LSET AN$(I%) = MID$(TR$,
          I1%, I2% - 1)
880       MID$(TR$, I1%, I2%) =
          STRING$(I2% - I1% + 1, " ")
890       I1% = I2% + 1
895 NEXT I%
899 RETURN
900 REM ------------------------------------
901 REM
902 REM    pack and rewrite data file record
903 REM
904 REM ------------------------------------
905 REM
```

```
920  TR$ = STRING$ ( 127, \:" ) : I1% = 1
925  FOR I% = 1 TO AN%
930       I2% = I1% + LEN ( AN$ ( I% ) )
          - 1
935       MID$ ( TR$, I1%, I2% ) = AN$
          ( I% )
940       I1% = I2% + 2
945  NEXT I%
950  LSET MR$ = TR$
955  PUT 2, - ARC% ( ITEM% )
960  RETURN
1000 REM ----------  ----------
1001 REM                finish
1002 REM ----------  ----------
1003 REM
1010 CLOSE 1, 2
1015 OPEN \0", 2, HEADER$
1020 PRINT #2, FSCREEN$; \, "; ROOT%;
     LNG%; LNF%; AN%; LINS%; N%;
     SIZE%; INDEX$; \, "; MAST$
1025 CLOSE 2
1030 REM \DBMENU.BAS"        'bail out.
1100 REM ----------  ----------
1101 REM
1102 REM    Retrieve data using Screen form
1103 REM
```

```
1104 REM ----------------------------
1105 REM
1106 CLS
1107 OPEN "I", 2, "CURRENT.DAT"
1108 INPUT #2, HEADER$
1109 CLOSE 2
1110 OPEN "I", 2, HEADER$
1115 INPUT #2, FSCREEN$,ROOT%,LNG%,
     LNF%, AN%, LINS%, N%, SIZE%,
     INDEX$, MAST$
1120 CLOSE 2
1125 NO% = N% + 1 : DIM FLAG%(NO%),
                    KEYS$(NO%),
                    ARC%(NO%)
1130                DIM SFLAG%(NO%),
                    SKEYS$(NO%),
                    SARC%(NO%)
1135 DIM AN$(AN$)
1137 OPEN "I", 2, FSCREEN$
1140 FOR L% = 1 TO LINS% : INPUT #2,
     RW$(L%) : NEXT L%
1145 CLOSE 2
1150 OPEN "R", 1, INDEX$
1155 FIELD 1, 127 AS R$
1160 REC$ = SPACE$(127) : ZERO$ =
     SPACE$(SIZE% - 3) : LSET ZERO$
```

```
    = "0"
1165 K$ = SPACE$(SIZE% - 3) : TR$ =
    SPACE$(128)
1170 OPEN "R", 2, MAST$
1175 FIELD 2, 127 AS MR$
1180 LINE INPUT "Enter search key value : ";
    KINP$ : LSET K$ = KINP$
1185 LINE INPUT "       Correct(Y/N)?";
    Y$
1190 IF Y$ = "y" OR Y$ = "Y" THEN 1192
    ELSE 1180
1192 IF LEN(KINP$) = 0 THEN 1000
1195 GOSUB 500                  'search
1196 IF D$ <> "" THEN 1180
1197 GOSUB 800
1199 REM -------forms display-----------
1200 REM
1210 K% = 0
1220 FOR I% = 1 TO LINS%
1225        SRW$ = RW$(I%)
1230        PRINT USING "##"; I%;   :
            PRINT ".";
1235        IF INSTR(LEFT$(RW$(I%),
            1), "-") = 1 THEN 1237
1236        IF INSTR(LEFT$(RW$(I%),
            1), " ") = 0 THEN 1240
```

```
1237        RW$(I%) = RIGHT$(RW$
            (I%), LEN(RW$(I%)) -
            1)
1238        PRINT " "; : GOTO 1235
1240      J% = INSTR(RW$(I%), ":")
1242      JSTAR% = INSTR(RW$(I%),
          "*")
1243        IF JSTAR% = 0 THEN 1250
1245        IF JSTAR% < J% THEN 1300
1250      IF J% = 0 THEN 1300
1260        PRINT LEFT$(RW$(I%),
            J%);
1270        K% = K% + 1
1271        RW$(I%) = MID$(RW$
            (I%), J% + LEN(AN$
            (K%)) + 1)
1280        PRINT AN$(K%);
1290        GOTO 1235
1300      J% = INSTR(RW$(I%), "*")
1310      IF J% = 0 THEN 1340
1311      K% = K% + 1 : PRINT LEFT$
          (RW$(I%), J%);
1312      RW$(I%) = MID$(RW$(I%),
          J% + LEN(AN$(K%)) + 1)
1313      PRINT AN$(K%);
1338      GOTO 1235
```

```
1340        PRINT : RW$(I%) = SRW$
1345 NEXT I%
1400 REM ----modify, delete or what---
1401 REM
1405 PRINT : PRINT
1410 LINE INPUT "Enter M=modify,L=lookup,
     D=delete, E=exit : "; C$
1420 IF C$ = "E" OR C$ = "e" THEN 1000
1430 IF C$ = "D" OR C$ = "d" THEN 1450
1440 IF C$ = "M" OR C$ = "m" THEN 1500
1445 GOTO 1180
1450 REM --------delete data--------
1455 LINE INPUT "Are you sure you want to
     delete this information ?"; Y$
1460 IF Y$ <> "Y" and Y$ <> "y" THEN
     1410
1465 FLAG%(ITEM%) = 2
1470 GOSUB 150          'rewrite b-tree node
1475 GOTO 1410
1500 REM -------change AN$--------
1505 Y$ = "N"
1510 PRINT "Enter 0=quit, #=field to change
     : "; : INPUT L%
1515 IF L% <= 0 OR L% > AN% THEN
     1545
1520 PRINT "Change"; AN$(L%); "to"; :
```

```
    INPUT C$
1525 LINE INPUT "Are you sure ? "; Y$
1530 IF Y$ <> "Y" AND Y$ <> "y" THEN
     1510
1535 AN$(L%) = C$
1540 GOTO 1510
1545 IF Y$ = "N" THEN 1410
1550 GOSUB 900 : GOTO 1410    'rewrite data
1599 END
```

图 7.2d. 从表中检索行的程序

```
1    REM ----------  ----------
2    REM
3    REM            backup.bas
4    REM    Copy b-tree index file and
            master data file
5    REM     Remove deleted items in the
            database
6    REM
7    REM --------  ------------
8    REM
10   GOTO 1100        'branch to main program
100  REM ----------------  -----
105  REM
110  REM            read.bas
```

```
115 REM     input a b-tree node from disk
            file #1
120 REM
125 REM ------------------------------------
130 GET 1, P0% : LSET REC$ = R$
131 FOR INDEX% = 1 TO N%
132     CH% = SIZE% *( INDEX% - 1 )
133     FLAG$ = MID$ ( REC$, CH%
        + 1, 1 )
134         IF FLAG$ = "E" THEN FLAG%
            ( INDEX% ) = 0
135         IF FLAG$ = "F" THEN FLAG%
            ( INDEX% ) = 1
136         IF FLAG$ = "D" THEN FLAG%
            ( INDEX% ) = 2
137     KEYS$ (INDEX% ) = MID$(REC$,
        CH% + 2, SIZE% - 3 )
138     ARC% ( INDEX% ) = CVI( MID$
        ( REC$, CH% + SIZE% -1, 2 ) )
139 NEXT INDEX%
140 ARC% ( N% + 1 ) = CVI( MID$ (REC$,
    126, 2 ) )
145 LINE% = ARC% ( N% + 1 ) : RETURN
149 REM
150 REM ------------------------------------
160 REM
```

```
160 REM          write.bas
165 REM     Output a b-tree node to file #1
170 REM
175 REM ---------------------------------
180 FOR INDEX% = 1 TO N%
181       CH% = SIZE% * (INDEX% - 1)
182       ON FLAG%(INDEX%) + 1 GOTO
          183, 184, 185
183           FLAG$ = "E" : GOTO 186
184           FLAG$ = "F" : GOTO 186
185           FLAG$ = "D"
186       MID$(REC$, CH% + 1, 1) =
          FLAG$
187       MID$(REC$, CH% +2, SIZE%
          - 3) = KEYS$(INDEX%)
188       MID$(REC$, CH% + SIZE% -
          1, 2) = MKI$(ARC%(INDEX%))
189 NEXT INDEX%
190 MID$(REC$, 126, 2) = MKI$(ARC%
    (N% + 1))
195 LSET NR$ = REC$ : PUT G%, LNF%
199 RETURN
250 REM ---------------------------
251 REM
252 REM          Restore a b-tree node
253 REM ---------------------------------
```

```
260  FOR INDEX% = 1 TO N% + 1
262       FLAG%(INDEX%) = SFLAG%
          (INDEX%)
264       KEYS$(INDEX%) = SKEYS$
          (INDEX%)
266       ARC%(INDEX%) = SARC%
          (INDEX%)
268  NEXT INDEX%
270  RETURN
500  REM ------------------------
501  REM
502  REM  Search b-tree for left-most item,
          only
503  REM
504  REM ------------------------
505  REM
510  D$ = "Found" : P0% = ROOT%
515  GOSUB 100        'read a node
520  IF ARC%(1) = 0 THEN 525 ELSE 535
525       LINE INPUT "File is empty. Strike
          RETURN"; Y$
530       D$ = "Not Found" : RETURN
535  IF ARC%(1)<0 THEN 540 ELSE 545
540       ITEM% = 1 : RETURN
545  P0% = ARC%(1)
550  GOTO 515
```

```
700 REM ----------------------------
701 REM
702 REM          Read next sequential node
703 REM
704 REM ----------------------------
710 D$ = "" : PO% = LINK%
720 IF PO% = 0 THEN 725 ELSE 740
725       D$ = "Done" : RETURN
740 GOSUB 100 : ITEM% = 1 : RETURN
750 REM ----------------------------
751 REM
752 REM          Fill new index file node
753 REM
754 REM ----------------------------
755 FOR NI% = NI% TO N%
760       SKEYS$(NI%) = ZERO$
765       SFLAG%(NI%) = 0
770       SARC%(NI%) = 0
775 NEXT NI%
780 IF D$ = "Done" THEN SARC%(N% +
    1) = 0 ELSE SARC%(N% + 1) =
    LNF% + 1
795 RETURN
900 REM ----------------------------
901 REM
902 REM          Change file names
```

```
903 REM
904 REM ----------------------------
909 ON ERROR GOTO 975
910 NEWNAME$ = LEFT$(INDEX$,
    INSTR(INDEX$, ".")) + "BAK"
915 NAME INDEX$ AS NEWNAME$
920 NEWNAME$ = LEFT$(MAST$,
    INSTR(MAST$, ".")) + "BAK"
925 NAME MAST$ AS NEWNAME$
930 NEWNAME$ = LEFT(OUTDEX$,
    INSTR(OUTDEX$, ".")) + "DAT"
935 NAME OUTDEX$ AS NEWNAME$
940 NEWNAME$ = LEFT$(NW$, INSTR
    (NW$, ".")) + "DAT"
945 NAME NW$ AS NEWNAME$
950 NEWNAME$ = LEFT$(HEADER$,
    INSTR(HEADER$, ".")) + "BAK"
955 NAME HEADER$ AS NEWNAME$
960 ON ERROR GOTO 0
965 GOTO 999
970 REM ---Error handling routine-----
975 IF ERR<>58 THEN 990
980       KILL NEWNAME$
985       RESUME
990 ON ERROR GOTO 0
995 RESUME
```

```
999 RETURN
1000 REM ---------- - ---------
1001 REM
1002 REM            Finish up
1003 REM
1004 REM ------------- - ------
1005 REM
1015 CLOSE 1, 2
1019 HEADER$ = LEFT$(HEADER$,
     INSTR(HEADER$, "," ) ) + "DAT"
1020 OPEN "O", 2, HEADER$
1025 PRINT #2, FSCREEN$; ", "; ROOT%;
     NPTR%-1; LNF%-1; AN%; LINS%;
     N%; SIZE%;
1026 PRINT #2, INDEX$; ",", MAST$
1030 CLOSE 2
1035 RETURN
1100 REM ---------- - ---------
1101 REM
1102 REM        Copy and garbage collect
1103 REM
1104 REM ------------ - -------
1105 REM
1109 CLS : LOCATE 2
1110 PRINT SPACE $(30); "Backup Database"
     : LOCATE 5
```

```
1111 OPEN "I", 2, "CURRENT.DAT"
1112 INPUT #2, HEADER$
1113 CLOSE 2
1114 OPEN "I", 2, HEADER$
1115 INPUT #2, FSCREEN$, ROOT%,
     LNF%, AN%, LINS%, N%, LNG%,
     SIZE%, INDEX$, MAST$
1120 CLOSE 2
1125 N0% = N% + 1 : DIM FLAG%(N0%),
     KEYS$(N0%), ARC%(N0%)
1130                    DIM SFLAG%(N0%),
                        SKEY$(N0%),
                        SARC%(N0%)
1150 OPEN "R", 1, INDEX$
1155 FIELD 1, 127 AS R$
1160 REC$ = SPACE$(127) : ZERO$ =
     SPACE$(SIZE% - 3) : LSET ZERO$
     = "0"
1170 OPEN "R", 2, MAST$
1175 FIELD 2, 127 AS MR$
1200 REM ------------------------
1201 REM
1202 REM    Now that the files are open, etc
1203 REM    create backup copies.....
1204 REM
1205 REM ------------------------
```

```
1210 REM
1300 OUTDEX$ = LEFT$(INDEX$, INSTR
     (INDEX$, ".")) + "TMP"
1310 NW$ = LEFT$(MAST$, INSTR
     (MAST$, ".")) + "TMP"
1320 PRINT "Busy working...."
1330 OPEN "R", 3, OUTDEX$
1340 FIELD 3, 127 AS NR$
1350 G% = 3
1360 OPEN "R", 4, NW$
1370 FIELD 4, 127 AS RR$
1380 GOSUB 500
1390 IF D$ = "Not Found" THEN 1395 ELSE
     1400
1395 CLOSE 1, 2, 3, 4 : RUN"DBMENU"
1400 REM --Copy from old master to new--
1470 NPTR% = 1 : LNF% = 1 : ITEM% =
     1
1480 REM -------loop----------
1490        FOR NI% = 1 TO N% - 1
1492           IF ITEM% = N% THEN GOSUB
               700
1496           IF D$ = "Done" THEN 1760
1500           MPTR% = -ARC%(ITEM%)
1505           IF T%MPR = 0 THEN 1506
               ELSE 1510
```

```
1506              ITEM% = ITEM% + 1
1507              GOTO 1492
1510           IF FLAG%(ITEM%)= 2 THEN
               1506
1550           GET 2, MPTR%
1560           LSET RR$ = MR$
1570           PUT 4, NPTR%
1580 REM ------copy index info-------
1590           SKEYS$(NI%) = KEYS$
               (ITEM%)
1600           SFLAG%(NI%) = 1
1610           SARC%(NI%) = -NPTR%
1630 REM ---update new master pointer---
1650           NPTR% = NPTR% + 1
1700 REM -----update old index info-----
1720           ITEM% = ITEM% + 1
1750        NEXT NI%
1751 PRINT "Calm down, I'm still working on
     it...."
1755 IF(KEYS$(ITEM%) = ZERO$) AND
     (LINK% = 0) THEN 1756 ELSE 1760
1756        D$ = "Done" : SARC%(N% +
     1) = 0
1760 GOSUB 750 : GOSUB 250 : GOSUB 150
1770 LNF% = LNF% + 1
1780 IF D$ = "Done" THEN 1850
```

```
1785 IF ITEM% <> N% THEN GOSUB 100
1790 GOTO 1490
1850 REM ----------
1855 REM  Close files and redefine header file
1860 REM ----------
1880 CLOSE 1, 2, 3, 4
1885 GOSUB 900        'Rename files
1890 OUTDEX$ = INDEX$
1900 NW$ = MAST$
1910 PRINT "Data file reorganized. Now for
     the index file...."
1930 REM -----Do b-tree  tier by tier-----
1950  P0% = 1 : ROOT% = LNF%
1960  OPEN "R ",1, OUTDEX$
1965 G% = 1 : FIELD 1, 127 AS NR$
1980 REM ---Find last and  move it up---
2000 KOUNT% = 1 : D$ = " "
2010 FOR ITEM% = 1 TO N% - 1
2020        GOSUB 100 : SFLAG% (ITEM% )
            = 1
2030        I% = 0
2031            I% = I% + 1
2032            SKEYS$ (ITEM% ) = KSYS$
                (N% - I% )
2033            IF KEYS$ (N% - I% ) =
                ZERO% THEN 2031
```

```
2060        SARC%(ITEM%) = PO%
2070        PO% = LINK%
2080        IFPO% = 0 THEN 2100
2090 NEXT ITEM%
2100 REM ------Finish off node-----
2105 PRINT "You are being so patient···· "
2130 IFPO% = 0 THEN 2140 ELSE 2250
2140        D$ = "Done ": NI% = ITEM%
            + 1 : GOSUB 750
2150        GOSUB 250 : GOSUB 150
2155        PO%= ROOT% : ROOT%=LNF%
            + 1: LNF% = ROOT%
2160        GOTO 2330
2200 REM -----More still to come----
2250        KOUNT% = KOUNT% + 1
2255        NI% = N%
2260        GOSUB 750 : GOSUB 250 : GOSUB
            150
2270        LNF% = LNF% + 1 : GOTO 2010
2330 IF KOUNT% = 1 THEN 2340 ELSE 2000
2340 PRINT "Done at last!! "
2390 ROOT% = ROOT% - 1 : GOSUB 1000
2400 RUN "DBMENU "
2500 END
```

图7．2e．　复制数据库的程序

```
100 REM ----------  ----------
105 REM
110 REM              screen.bas
115 REM Build a screen format for data entry
120 REM
125 REM ---------------  -----
130 REM
131 OPEN "I ", 2, "CURRENT.DAT "
132 INPUT #2, HEADER$
133 CLOSE 2
135 OPEN "I ", 2, HEADER$
140 INPUT #2, FSCREEN$,ROOT%, LNG%,
    LNF, AN%, LINS%, N%, SIZE%,
    INDEX$, MAST$
145 CLOSE 2
150 CLS
155 PRINT "Enter form a line at a time,
    remember : "
160 PRINT "        * = keyed value "
165 PRINT "        : = begins a field "
170 PRINT "        - = field designation "
175 PRINT "         "; LINS%; "= lines per
                    screen "
176 PRINT : PRINT
180 DIM RW$ ( LINS% )
185 FOR L% = 1 TO LINS%
```

```
190         PRINT USING "## "; L%;
195         LINE INPUT RW$(L%)
200   NEXT L%
205   REM ----------  ---------
210   REM                edit it
215   REM ----------  ---------
220   PRINT
225   INPUT "Enter 0 to stop, or line number
      to change : "; L%
230   IF L% <= 0 OR L%>LINS% THEN 275
235   PRINT USING "## "; L%;
240   PRINT RW$(L%)
245   PRINT USING "## "; L%;
250   LINE INPUT RW$(L%)
255   GOTO 220
260   REM --------  ----------
265   REM          File it in fscreen$
270   REM ----------  --------
275   REM
290   PRINT : LINE INPUT "Do you want hard
      copy ? "; Y$
295   IF Y$ <> "y "AND Y$ <> "Y "THEN
      310
300   GOSUB 360 : GOTO 290
305   REM ------------------
310   OPEN "0 ", 1, FSCREEN$
320
```

```
315 FOR L% = 1 TO LINS%
320       PRINT #1, RW#(L%)
325 NEXT L%
330 CLOSE 1
335 PRINT : PRINT "Form saved in ";
    FSCREEN$
340 RUN "DBMENU "
345 REM ------------------------------
350 REM              Hard copy
355 REM ------------------------------
360 LPRINT : LPRINT "    Screen Form ";
    FSCREEN$
365 LPRINT : LPRINT
370 FOR L% = 1 TO LINS%
375       LPRINT USING "## "; L%; :
    LPRINT RW$(L%)
380 NEXT L%
385 LPRINT : LPRINT : LPRINT
390 RETURN
400 END
```

图7．2f．　建立屏幕显示格式程序

```
1   REM ------------------------------
2   REM
3   REM              dump.bas
4   REM  Print contents of the entire b-tree
```

```
5   REM
6   REM ---------- ---------
7   REM
10  GOTO 1100
100 REM ---------- ----------
105 REM
110 REM                read.bas
115 REM Input a b-tree node from disk file #1
120 REM
125 REM ---------- ---------
130 GET 1, P% : LSET REC$ = R$
131 FOR INDEX% = 1 TO N%
132     CH% = SIZE% *( INDEX% - 1 )
133     FLAG$ = MID$ (REC$, CH% +
        1, 1 )
134       IF FLAG$ = "E "THEN
          FLAG% ( INDEX% ) = 0
135       IF FLAG$ = "F "THEN
          FLAG% ( INDEX% ) = 1
136       IF FLAG$ = "D "THEN
          FLAG% ( INDEX% ) = 2
137     KEYS$ ( INDEX% ) = MID$
        (REC$, CH% + 2, SIZE% - 3)
138     ARC% ( INDEX% ) = CVI ( MID$
        (REC$, CH% + SIZE% - 1, 2))
139 NEXT INDEX%
```

```
140  ARC%(N% + 1) = CVI(MID$(REC$,
         126, 2))
145  RETURN
149  REM
1100 REM ------------ ----------
1101 REM
1102 REM  Quick read-out of b-tree index file
1103 REM
1104 REM ------------ ----------
1105 REM
1110 CLS : LOCATE 12
1111 OPEN "I", 2, "CURRENT.DAT"
1112 INPUT #2, HEADER$
1113 CLOSE 2
1115 OPEN "I", 2, HEADER$
1120 INPUT #2, FSCREEN$, ROOT%,LNG%,
     LNF%, AN%, LINS%, N%, SIZE%,
     INDEX$, MAST$
1125 CLOSE 2
1130 N0% = N% + 1 : DIM FLAG%(N0%),
     KEYS$(N0%), ARC%(N0%)
1135 OPEN "R", 1, INDEX$
1140 FIELD 1, 127 AS R$
1145 REC$ = SPACE$(128) : LSET ZERO$
     = "0"
1150 K$ = SPACE$(SIZE% - 3)
```

```
1151 PRINT "Turn on printer, strike RETURN
     : "
1152 PRINT "Printer ready ( RETURN if it
     is ) "; : INPUT Y$
1153 LPRINT "Quick dump of b-tree index
     file : "
1154 LPRINT : LPRINT
1155 P% = 0
1160 FOR II% = 1 TO LNF%
1165        P% = P% + 1 : GOSUB 100
            'read a node
1170        LPRINT "--Node = ", P%,
            "---------------------- "
1175        FOR I% = 1 TO N%
1180            IF FLAG% ( I% ) = 0 THEN
                C$ = "Empty "
1185            IF FLAG% ( I% ) = 1 THEN
                C$ = "Full "
1190            IF FLAG% ( I% ) = 2 THEN
                C$ = "Deleted "
1195            LPRINT "Flag = "; C$
1200            LPRINT "Key = "; KEYS$
                ( I% ) ;
1205            LPRINT "Arc = "; ARC%
                ( I% )
1210        NEXT I%
```

```
1215          LPRINT"Link =  "; ARC%(N0%)
1220 NEXT II%
1225 CLOSE 1
1230 REM ---------Other stuff---------
1240 LPRINT : LPRINT
1250 LPRINT "Root node =  ", ROOT%
1260 LPRINT "Number items indexed =  ",
     LNG%
1270 LPRINT "Number nodes (sectors) = ",
     LNF%
1280 LPRINT "Number fields per input form
     = ", AN%
1290 LPRINT "Number lines per input form =
      ", LINS%
1300 LPRINT "Number items per node (sector)
     = ", N%
1310 LPRINT "Size (chars) of each node item
     = ", SIZE%
1320 LPRINT "Name of master (data) file = ",
     MAST$
1330 LPRINT "Name of index (b-tree) file = ",
     INDEX$
1400 RUN "DBMENU "
1499 END
```

图 7．2g．打印数据库的二叉树索引程序

索引文件怎样工作

在上述例子中，主菜单选择6时，将使索引文件中的全部内容转送到打印机上输出。索引文件有一种灵巧的文件结构，称为二叉树，它是任何高速数据管理系统中的核心，因为在存贮和检索软盘上的信息时，二叉树是最好的结构。不需了解二叉树结构便可使用DBMS，但是，掌握二叉树结构的有关知识对设计出较好的数据库管理系统是有益的。

任何设计和实现DBMS的人会遇到以下问题："当信息由一个或多个索引值索引时，怎样才能迅速地从磁盘中检索出信息呢？"。事实上，大多数设计的DBMS，为了迅速检索到信息，总是按字顺把信息存贮起来，这样便可以用二叉树文件结构解决快速检索信息的问题。

二叉树结构

索引文件是含有索引键和记录号的文件，例如，一个关系可以存贮在主文件中。

索引文件的作用是存贮从主文件中抽取的键值，在存取记录时，先从索引文件中找出该记录的键，然后用索引文件中的记录号存取主记录，所以索引文件可以加快存取主文件记录的速度。图7—3所示为用两个索引文件存取主记录的情况，其中假定（Name）索引文件存的是主文件中的所有人名，另一个是年令（Age）索引文件，它存的是主文件中的年令。若用人名存取主文件中的记录，则先查找Name索引文件，再由该人名键中的记录号找出主记录。图7—3说

明了如何由人名索引文件查找人名为“Adams”的记录，这个人名索引文件中含有人名键值为“Adams”，记录号为（4）。同样，主文件记录中最小的年令为18，这样，年令索引查找的是与年令（18）对应的值，与这一值对应的主记录号为（5），从而，找到了年令为18的主记录。

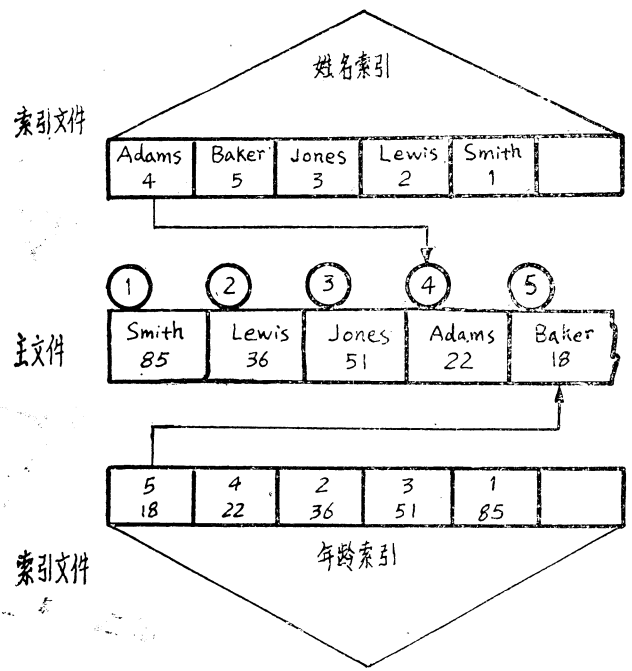


图7—3　用人名或年令存取的两个索引文件。箭头表示怎样找到Adams和年令18。

注意：在图7—3中，两个索引文件中的记录项是按字母上升顺序存贮的，这就是说，我们可用人名索引文件的人

名字顺和年令索引文件中的年令排列次序检索到所有的主记录。两个索引文件只管记录存取键的顺序而不管在主文件上存贮记录的顺序。

图7—3的示例阐述了索引文件的两个重要特性：

1．索引文件的结构必须能使查找操作迅速执行。

2．索引文件的结构必须能按序检索项目。

上面两点实质上就是二叉树结构要做的事情。

图7—4说明如何构造一个保存文字字母的二叉树。图7—4中文字字母是按反序输入的，即Z先输入，然后输入Y、X、W等等。

图7—4中的框相应于软盘上的一个区段（128字节），每个框中保存4个键和5个记录号，负记录号用于指示主文件记录的位置，正记录号用于指示二叉树索引文件中的另一个区段。

图7—4（α）表示Z存入二叉树以后的二叉树索引文件的内容，（—1）指向主文件的记录（十1）（图中未画出），因此，要检索含键值“Z”的记录，先查找二叉树，若找到Z，则（—1）记录号可作为主文件的存取记录号（十1）。

图7—4（b）表示在二叉树索引文件中增加X、Y以后的二叉树内容，含Y的主记录存贮在记录号(十2)中，含X的主记录存贮在记录号（十3）中。

图7—4（C）为索引文件区段由于有多余的项目号而发生溢出的情况。索引文件的每个区段存有4个键和5个指向其它区段（记录）的指针，所以，当把4个键放入节点（节点与区段相同）时，节点便“溢出”了，这标志索引文

件（二叉树）分配了较多的节点。

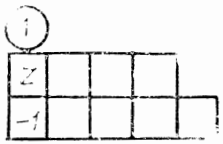
二叉树节点放满键时，可分为两个节点。注意图7—4（C）出现分叉节点的情况，节点1中的两个键转移到在区段2存贮的一个新节点中。注意，这两个键用指向节点2的第5个记录号保持字母顺序。

现在看看两个节点由“父”节点（含有每个“子”节点的最大键）“管理”的情况。在图7—4（C）中，最大的键为X，它存在节点1中，所以父节点存有X和指向节点1的指针。其它子节点的最大键为Z，所以父节点也指向节点2。父节点不需要存Z，因为在整个二叉树中没有比Z还大的键。

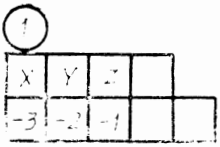
现在我们若要查找在图7—4（C）中含有键“Y”的记录，先从区段3中的父节点开始查找，把检索键Y与存在区段3的父节点上的每个键相比较，由于Y大于X，则第二个指针（+2）用于检索存在二叉树2区段中的键，在2区段中把N与其区段中的项目相比较，结果发现第一项目是匹配的，（-2）指针可检索主文件中的记录2。

图7—4（d）表示继续向二叉树加放键的情况，进一步再分区段1将会产生另一个子节点，水平链继续把二叉树中的项目按字母顺序连接起来。

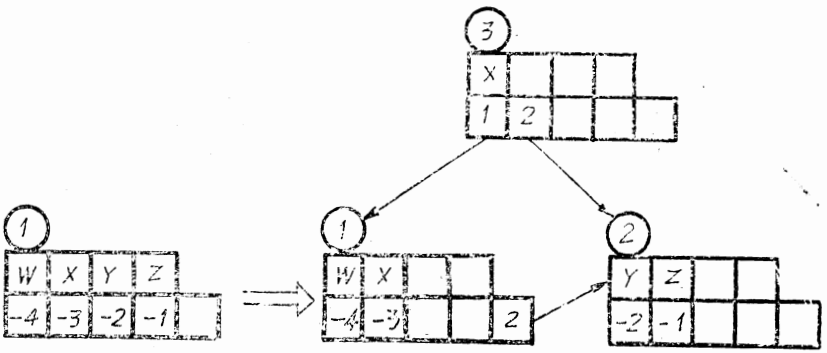
图7—4（e）表示二叉树在连续生成的过程中分离为更多区段的情况。可以看到，内部区段（3）分为区段3和7，还可看到二叉树向上生成和需要时加放新父节点的情况。



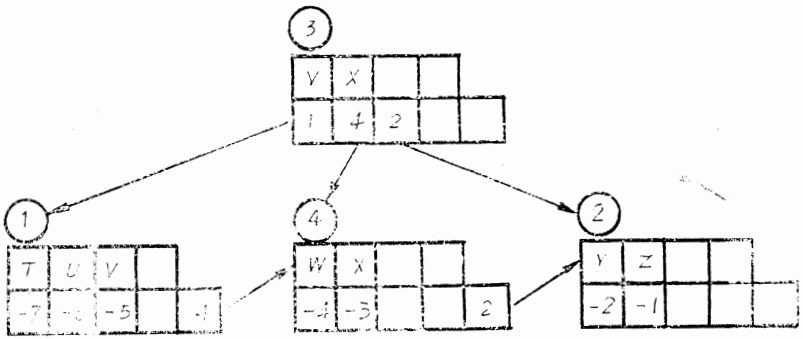
（a） 在二叉树根上存Z



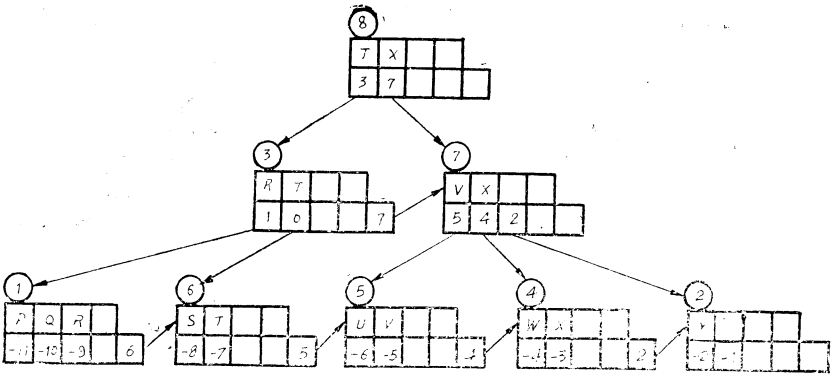
（c）对二叉树加W和将它分为二叉树



（b） 对二叉树加Y和X



（d） 对二叉树加V，U和T再分离节点!



（e） 加S、R、Q、P和分离内部节点

图 7—4 二叉树生成结构图

假若我们经由 B—树查找路径来找到与键值“X”相关的记录号，则查找的第一个节点在 B—树的顶部，查找节点 8。直到找到键X，这时查找被引向B—树的第七区段。在7区段再查找X，这次把查找引向 B—树的 4 区段。最后查找 4 区段的内容，直到找到X，知道主记录的存取记录号为（十3）。

为查找含X的主文件中的记录，对“X”要进行四次检索，这看来是过多了一点，其实不然，因为1）检索次数比文件系统中存贮的总记录数（11）的一半还少；2）按键的排列次序进行检索。后一点对数据库系统是非常重要的，因为一旦找到含 X的记录，则可按X到Z的次序迅速地检索到所有其它记录，检索途径是水平地沿存在 B—树叶上第5个记录号读 B—树区段，因此，对区段 2 只存取一次便可

检索到Y和Z键。

在上面的例子中，每个B—树区段只存4个键，而区段的大小完全依赖于每个键的长度。在大多数情况下，区段含128字节，它能存8个或10个键，存的键多可大大提高检索程序的性能，因为对磁盘存取的次数可近似地由下式算出：

$$LOG_{N/2}(K/2)$$

其中，N为每个区段上所存键的个数，

K为存在整个B—树中不相同键的总个数。由于在图7—4中，有N＝4，K＝11，则存取的次数为：

$$LOG_2(5)$$

它近似地等于2.5。

B—树索引文件的特性

B—树结构有一些优点也有一些缺点，我们在这里强调的是它的优点，当然也应适当地了解它的一些缺点。

优点 在B—树结构中，由于使用“各个击破”（divide—and—conquer）的检索方法，所以用B—树结构可以很快地检索到记录。

B—树结构可以使索引键按字母顺序排列保存，这样可容易检索到按序排列的信息。

B—树结构可容易增加新键，而重新编排键的工作却非常少，从而可相当快地扦入信息。

缺点 B—树结构是用冗余的信息来提高检索速度的，它的许多键不止存贮一次。事实上，B—树结构以下面两种方式占有磁盘空间：1）B—树的内部节点上存的是冗余信息；2）B—树的叶节点上几乎没有存放信息。

由于B—树结构在磁盘上是以两种形式存贮信息的，因

此检索时，访问这种磁盘次数比访问以一种形式存贮信息的磁盘次数要多，根据每个节点的大小，查找一个记录对磁盘要访问5到6次。

执行图7—2所给定的简单关系数据库管理系统的“Quik Dump”操作，可查看B—树文件中的实际内容。这个程序可转存B—树中所有区段的内容，每个区段含有一个FLAG字段，该字段指明键是否在使用（FULL）、未用（EMPTY）或已被删除（DELETED）。若记录号指向另外的B—树区段，则ARC存正值。若ARC值指向主文件记录，则记录号为负。LINK指针按序指出下一个叶节点。

常见问题

问. DBMS是什么？

答. DBMS是一个有能增加、删除和修改信息的软件的计算机系统。典型的DBMS由物理文件系统、逻辑型用户数据视图、查询语言和处理报表的输入／输出控制子系统组成。

问. DBMS的三种逻辑用户模型是什么？

答. 网状型、层次型、关系模型。

问. 关系数据库管理系统给用户提供了什么简单存贮方法？

答. 关系数据库管理系统的所有数据可按一张张的表存贮。

问．什么是关系中的行和列？

答．一个行叫做一个元组，而一个列叫做一个定义域。

问．CRT 显示格式如何与关系发生联系？

答．CRT 显示格式定义关系的域，然后用显示格式一次输入一个元组。

问．关系代数是什么？

答．关系代数是对关系运算的总和，一些关系与另一些关系运算后可产生新的关系。

问．“关系”这个词不是为表专门设定的吗？

答．在非正式情况下关系不是为表设定的，但在用于专业方面，关系是作为描述表的特性的，在用关系代数查询处理时，关系所描述的表就像关系一样。

问．说明本章描述的简单的 DBMS中，物理和逻辑层的区别。

答．物理层由含有关系的主文件、含有 B— 树结构的索引文件和含有显示格式及各种参数的其它文件组成。而逻辑层只含有关系。

问．在简单的 DBMS 中所讨论的查询语言看来是原始语言，它真的是关系查询语言吗？

答．它不是关系查询语言，它是一种特别简单的问答查询语言，它仅用于说明想法。

问．备用数据库的两个原因是什么？

答．1）当正在工作的数据库出错时，可生成备用的数据库付本；2）为节省磁盘空间和处理时间删除

那些无用的记录。

问．一个主文件可有多少个与之相关的B—树索引文件？

答．需要多少个可建多少个。

问．请看图7—4（e）的B—树，真正用作存贮信息的空间占磁盘总空间的百分比为多少？

答．若把信息占有的所有空间和空槽都考虑为使用空间，则磁盘空间的利用率为56%，若只把叶节点上的键作为有用信息，则磁盘空间的利用率下降为15%。

第 八 章

计 算 机 怎 样 工 作

“盒子里装有什么？”

“一台计算机。”

“计算机是什么？”她兴致勃勃地问道。

“看！”他打开盒子。

“里面不过又是一只盒子罢了。”她觉得失望。

“再打开它。”他建议道。

她接连打开一个又一个盒子，但是每个盒子里都装有一只更小的盒子。最后，出现了一只很小的盒子，她又将它打开。

“这是什么？”她觉得有点沮丧。

“一个比特”他嘻嘻笑道。

人脑与计算机

几十年前，人们将自己的大脑作为构造第一台电子计算机的原型，这是可以理解的。事实上，今天的小型计算机与第一台巨大的电子机相比仍然有着很相似的操作方法—那就是模仿人的大脑。但是你如果建造过一台具有人脑功能的机电装置的话，那么，你是从何处着手的呢？人脑又是怎样工作的呢？

在这最后一章中，我们将要更加仔细地看一看计算机是如何工作的。不过仅仅为了使用计算机而去了解计算机如何工作，那就没有必要。但是，好奇的读者总是想知道更多的东西。因此，还是让我们来探讨一下作为人脑模型的计算机吧。

事实上，电子计算机只是极其肤浅地接近人脑。要使计算机能模拟人脑还必须对人进行更深入的了解。但是初看起来，这两部“机器”又是何等相似啊！

人脑和计算机两者都要接受输入：人要接受光、声、触觉等输入，而计算机则要接受表示数据的电信号输入。但两者都能以电荷的形式将信息保存在存储器中（短期保存或长期保存），並且，又能以语言、动作等形式产生输出。

人脑由意识（肤浅的理解）来控制，而计算机则由软件（程序）来控制。在这两种情况下，都由“程序”来处理输入的信息和来自长期存储器中的信息，並且产生输出。图8—1示出了这一输入—处理—输出的简单模型。

在图8—1（a）中，人接受视觉输入，利用长期记忆力和人的意识（处理器）处理该输入，于是，作出奔跑（害怕）的决定。这里的输出就是向腿发出奔跑的信号。与此类似，图8—1（b）中的计算机模型利用输入的有关一位职工的数据以及计算机程序（软件）和长期存储器（文件）来处理一份工资表。工资表借助打印机输出。

显然，这里所说的计算机模型必须具备三个基本部分：

1）输入装置。如键盘、磁盘驱动器等等。

2）处理器。如微处理器芯片。

3）输出装置。如显示屏、磁盘驱动器和打印机。

现在，让我们逐一步考察以上各个部分，看看它们是如何工作的。

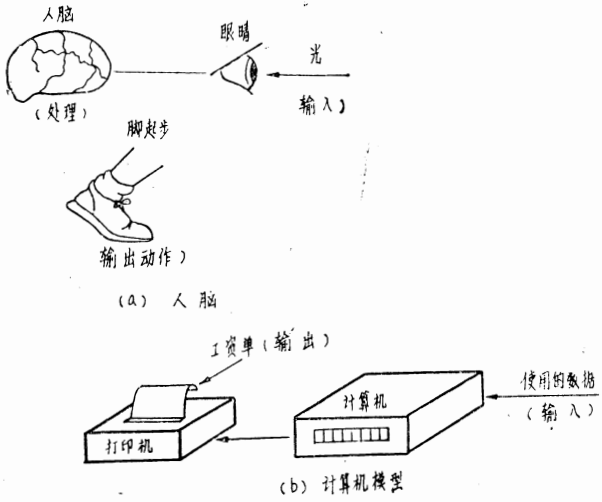


图 8—1．输入—处理—输出

机电存储器

人们利用短期记忆力来暂时地记住一个电话号码或者正在发生的事情，而用长期记忆力来记住好朋友的名字、你的家庭地址，等等。人脑实际上管理着各种层次的记忆，从容易回忆的“短期”事件直到难以回忆的“长期”事件。分级存储这一概念在计算中也同样得到应用。

寄存器

在一台电子计算机中，最容易回忆或存取的存储单元是寄存器。寄存器是能够存储一个信息字的若干电路的集合。在小型计算机中，一个字通常是8位或16位的信息。每一位都对应一个置于“开”或“关”的电路。如果电路处于“开”状态，则有电流流过，处于“关”状态则没有电流流过，正如家中利用开关管理台灯的开或关一样。

举个例子说，一个8位的字具有256种可能的开和关的组合方式。每一种方式都对应一种特定的意义。例如，英文字母就对应着26种组合的方式。这叫做编码，它是计算机之所以强有力的秘密之一。

主存储器

在计算机中，下一级存储就是“短期存储”部分。RAM（随机存取存储器）是成千上万个字节的集合（通常，每一字节有8位）。一个64K字节的存储器含有64×1024个字节，K(读作Kay)是计算机术语，它等于1024。一台8位计算机中的每个字就是一个8位的字节。一台16位计算机中的每个字就需要两个字节。

RAM主存储器中存有计算机运行所需要的程序和数据。如果程序和数据太多，本级存储器容纳不下，那么，必须采用下一级存储器。磁盘文件就是计算机的一种“长期”存储器。

磁盘文件

磁盘是一种廉价的存储媒介，里面存有大量字节的信

息。事实上，磁盘可以保存几兆而不是几千字节。因此，磁盘的容量是以兆字节来计量的。一兆字节或1M，等于1024×1024亦即是1,048,576个字节信息。

然而，对计算机来说，控制和存取磁盘是相当困难的。磁盘上的信息是以一个个磁点而不是一个个的开关电路的形式存储的。在被读／写电路检测之前，这一个个“点”必须动起来。因此，为了读出盘上的信息，磁盘驱动器首先使磁盘转动，使磁盘表面上的磁点通过一个读／写磁头的下面。当然，要使予想的磁区定位到磁头下是需要时间的，因而，计算机也就必须等待。

其次，存储在磁盘上的信息通常不是以一种很容易被计算机利用的形式存在的。例如为存储某人的姓名、地址和电话号码而将几百个字节组成“簇”，我们称之为记录。集中在一起的记录就组成了文件，文件的集合有时又称作数据库或程序库。

为了克服磁盘存储器和RAM存储器之间的不相容性，人们在文件结构上作了大量的努力並提出了很多的设想。例如，计算机程序员面临的困难之一就是为某一给定的应用选择一种最合适的文件结构。不过，由于软盘存储器具有存储容量大和成本低的优点，作为长期存储器，在将来很长一段时间内它肯定还会得到应用。

存储器的层次

让我们来考察一下存储器的层次，看看怎样利用这种结构使计算机工作。首先，最直接的一级存储器是寄存器。典型的计算机有2到10个寄存器，在计算机处理信息时，每个

寄存器都存有8位或16位的信息，有的是和数；有的是字符等等，而处理器对这些寄存器的内容进行某种运算。

如果计算机同时要存储大量数据的话，寄存器就不够用，所以有些数据就被存储在RAM中，余下的数据则被存放在磁盘文件中。另外，还必须告诉计算机下一步该做什么，因而，其中一个寄存器还必须用来存放称为指令的字。指令是一个字，它告诉处理器要执行什么运算（加、减、比较、传送、复制），以及根据什么操作码来执行（修改哪一个寄存器）。

同样，由于不可能有足够的寄存器来存放程序的所有指令，所以除现行指令外的其它指令都存放在RAM和磁盘文件中。这些含有指令信息的一些字的集合就是程序。

RAM级的存储器既存有数据又存有指令，它们等待着被取走和送往寄存器。因为指令和数据一样驻留在RAM中，这样，就有可能发生错误，本打算取指令却错取了数据，或者相反。因此有些存储器具有一种标记，它指出信息的类型：究竟是数据还还是指令。然而，计算机是怎样知道这一点的呢？

编　码

计算机存储器几乎没有“智能”。事实上，就计算机的存储器来说，所有信息都以各种开和关的组合方式存储起来的。那么，计算机是怎样区别例如字母A和数字5之间的不同之处呢？

计算机系统用编码的方式来表示存储器中的信息。这是计算机如何工作的最大秘密之一。因为代码只是一种利用开

关状态来表示信息的简单方法，所以关键在于如何表示。在存储器中的每一个数字、字母和特殊符号都是以一个或多个字节的开关位形式存储的。然而，数字的编码与字母的编码不同，这种不同的唯一表现是编码和译码的方法，这是由计算机的硬件和软件来实现的。

大多数小型计算机都采用以下信息类型，每种类型都对应一种编码方式，它们都是利用一个或多个开关状态来表示给定类型的信息。

信息类型	编　码
1．二进制位	一个开关（1或0）
2．布尔值	真（开），假（关）
3．整　数	2的补码表示的字节或字
4．实　数	浮点格式（字）
5．字　符	ASCII 代码
6．指　令	八进制数

让我们更详细地考察一下这些类型及其编码方式。搞懂这些代码将有助于你对几乎所有计算概念的理解。

一个比特就一个二进制的数字，也就是说，它是来自二进制数系统的一个数字。二进制数字仅仅用两个值：0或1，而不是用0，1，2，3，4，5，6，7，8，9十个值。电存储器的开关状态是由0（关）或1（开）来表示的。因此，一个比特就是计算机能够存储的最小信息块。

有时我们将信息编码为布尔值，就是说，将真或假的值存入存储器中。例如，某一位职工上月加班，我们可以存储

一个布尔值：真（TRUE）；相反，如果他或她未加班，则存储一个布尔值：假（FALSE）。正如你所看到的，真的编码为1（开），假的编码为0（关）。

整数是用字节或字来编码的。通常，在存储器中可用16位字编码－32768至＋32767的值。这里有几种方法，但在微计算机中最好的方法是2的补码表示法。

在2的补码表示法中，整数从0到＋32767都可编码成二进制数，也就是：

十进制数	二进制数
0	0 0 0 0
1	0 0 0 1
2	0 0 1 0
3	0 0 1 1
4	0 1 0 0
5	0 1 0 1
6	0 1 1 0
7	0 1 1 1
8	1 0 0 0
9	1 0 0 1
10	1 0 1 0
11	1 0 1 1
12	1 1 0 0
⋮	⋮
32,767	（15个1）

我们只要在一张纸上写出从0到最大（15个1）的所有十进制数，然后划去不仅仅包含0或1的数，就可以得到上表。例如；头几个数是：

把余下的二进制数重新编排就得到上面的换算表。记

十进制数字	被保留的数
0	0
1	1
2	
3	
4	
5	
6	
7	
8	
9	
10	10
11	11
12	
13	
⋮	⋮

住，计算机存储器仅仅能存储开关值（即0，1）。

这是得到正整数值的方法，但是负整数又怎么办呢？为了用2的补码形式存储负数，我们只需将所有的1转换为0，所有0转换为1即可。假设我们要将（－14）用8位2的补码形式编码，其步骤如下：

第1步，首先取正值＋14

第2步，将＋14编码成二进制数（该例中用8位）。即00001110。

第3步，将最右边的一个1以及这个1后面的所有的0作上标记，它们都是该数的不变部分。即00001110（最后两个数字10是不变部分）。

第4步，除不变部分外，将其余的所有1转换为0，所有的0转换为1。即11110010

第5步，结果就成了2的补码。即－14：11110010。

为了证实2的补码整数的编码正确性，我们可以计算（＋14）＋（－14），结果应为0：

```
  00001110        (+14)
+ 11110010
----------
 100000000
```

注意这里有进位1，这个1被舍弃而结果就成为0。记住，因为只有0和1是唯一许可的数字，所以加法（1＋1）＝（10）。

实数编码问题要比2的补码法更加复杂。这里，我们不讨论这个问题，但要提一下，这种编码包括对实数的两个部分编码。第一部分是指数（2的幂），第二部分是小数或尾数。例如：14.5实际上也可以是0.145乘以10的2次幂。我们将小数0.145用一种方式编码，又将10的2次幂用另一种方式编码。

字符信息又有另一种编码法，被称作ASCII(读作“ask—key”)，每当你在计算机键盘上按下一键，它就会送出一个ASCII代码，这是由8个开关位组成的一个字节，我们且以两个键为例来看看它们的ASCII代码。

A是十进制65或二进制01000001

a是十进制97或二进制01010001

计算机可能将字母A错误地当作一个整数、实数或指令来对待，因为它只是一个简单的8位字节的开关信息。然而，如果计算机通过编程解释出这是字母A的的代码，那么，它就会将这代码与其它字母进行比较，再输出给打印

机，等等。

现在，我们再来研究一下当键盘上的键5被按下时送入计算机的代码是什么，以及计算机是怎样来解释这个符号的。

数值5的ASCII编码如下：

5是十进制53或二进制00110101

奇怪的是数字5作为字符输入后不是以整数5的形式存储，计算机必须通过编程将5转换成整数5或实数5.0。

结构类型

我们已经看到，一个简单的二进制位、布尔值、整数、实数和字符信息是怎样存储在机电存储器中的。这些不同类型的信息都是以成组的开关形式存储的，但它们的编码方法不同。确实，这是一件要做的工作，否则，程序员必须将信息从一种类型翻译成另一种类型。有一种设想可以减轻程序员的工作，这就是将信息组成一定的结构。数据结构就是数据的集合，它包含信息和结构两重意义。我们来看看两种很重要的结构。

数组是一种数据结构，它由一个或多个具有同样类型的信息块组成。例如，整数数组就是一种字的序列，它们是以2的补码方式编码的数值。

字符组又称作字符串，它是包含字母和/或数字的序列。数组中的每个字都存放在存储器内的有序字列中，就好象一组邮局的信箱。

第二种数据是文件，它在计算中起很重要的作用。文件是一组记录，在典型的情况下，它被存放在软盘上。一个文

件的记录可能包含不同类型的信息。例如：字符、整数和布尔值。因此，一个文件可以是不同类型信息的混合。

文件结构的主要目的是允许我们使用磁盘作为一种成本低、容量大的存储媒介。然而，磁盘存储的方法与RAM（主存储器）存储的方法很不相同。一般说来，磁盘上的信息是以文件结构而不是以数组结构来存储的。这就需要程序员从一种存储方法翻译到另一种存储方法，然后再翻回。

芯片的威力

计算机的威力来自一个小型电路，它被装在一个芯片里，只有指甲大小。这就是微处理器，它能够根据指令的要求来执行操作。

在这一节中，我们来讨论处理器操作的基本概念。这将要求理解两件事：1）排序，2）计算机运算。这些功能在所有的计算机中都是一样的，不过实现这些功能的细节因计算机不同而有所不同。

排　序

排序的思想很简单，在计算中也很有用。这是微处理器如何工作的决窍。

重复下面步骤直到关闭电源为止。

步骤1．从RAM中取出一条指令，

步骤2．从寄存器中取出该指令将要用到的数据，

步骤3．根据指令的说明对数据作指定的运算。

步骤4．计算出按步骤1要取出的下一条指令在存储器

中的地址。

只要微处理器加上电，时序发生器就开始工作。程序中第一条指令的地址必须是已知的，同时，一旦执行完第一个程序，微处理器就必须得到另一个程序以继续运行。

序列的头三步很直观而且容易实现。微处理器只要简单地取出一条指令和数据进行处理（加、减、复制、传送、比较），然后做一些予定的操作。例如，取出一条指令，要求将两个整数相加。执行这条指令首先要取出两个数，然后再以2的补码形式做加法。运算结果得到一个和数，存储在寄存器中。另外再有一条指令将这个和数送回RAM的某些单元中。

第四步要稍微麻烦些，但是这里就显示了排序微处理器的威力。它有三种基本的方法来计算出下条指令的地址。

1．顺序执行下条指令

2．利用程序中的跳转指令转移到程序中的其它地址

3．暂时挂起现行程序，转而执行另一个程序。当该程序被执行完成后再回到被中止的程序上並顺序执行下面一条。

第一种方法称作顺序执行，因为微处理器只是顺序取出下条指令来执行。

第二种方法被称作转移，因为这时待执行的下一条指令取决于布尔值，是真还是假。

这种转移有两种基本控制方案。

1．如果—则—否则转移。由此方式作出决定。

2．为了反复执行某段程序而采取循环的方式。

如果—则—否则转移的方案是用来有条件地执行某段程

序否则执行另一段程序，这都取决于测试的结果。例如，如果一位职工在一周内工作超过40小时，那么就按加班工资计算，否则就按正常工资计算。注意，这里只有一种选择，而不是两种。

循环是用来反复执行某段程序的一种方法。这种方法很有用，它可以对某段程序反复执行从而减少程序的长度。例如我们可以重复一段打印程序，在一份工资表上分别打印出100位职工的工资清单。重复了予定的次数以后或者当某一条件满足以后（例如，职工全部打印完），循环就结束。

第三种方法被称作子程序，它又包含两种方法：

1．程序的子程序调用

2．中断服务子程序调用

在第一种方法中，程序员写出一段称之为过程（或子程序）的程序，该程序从属于主程序并由主程序利用一条特殊的调用子程序指令来调用。这是一种“各个击破”的思想，也就是说，先将程序分成若干个较小（且较简单）的过程，然后在需要的时候一次一个地调用。

第二种方法是由计算机系统本身用来处理一些实时性较强的事件。比如，从磁盘里读取信息，回答电话的呼叫等。中断就是一个非予定的事件，它通常由输入／输出设备发出信号而引起的。由于我们总不能予测什么时候执行中断服务子程序，因而调用这种子程序只能由微处理器本身来完成。简言之，一个中断服务子程序是一个由硬件调用的过程而不能由主程序中的专门指令来调用。

多处理器

大多数小型计算机系统都含有很多个微处理器，而不仅仅只是一个。例如，可以用微处理器来控制磁盘、打印机和屏幕／键盘的存取。一个计算机系统包括有由若干微处理器来存取的多层次的存储器。

在大多数情况下，微处理器需要在一个固定程序的控制之下。也就是说，一部打印机可以由一个微处理器控制，这个微处理器控制打印轮(或别的什么部件)、打印格式等等。而打印机的微处理器本身又是由厂家提供的固定程序来控制的。这个程序永久保持不变並且仅仅供打印机的微处理器执行。因此它被存储在称之为ROM（只读存储器）的一类特别的存储器中。

ROM不能被改变，就是说，一旦程序被存入ROM中，它就永久地驻留在那里了。即使ROM的电源被切断，程序依然不变。

计算机运算

正如我们上面所讨论的那样，微处理器通过每次执行一条指令的办法来顺序地完成一道程序，但是要更准确地说，微处理器指令又做些什么事呢？在这一节中，我们大致来看看，一个典型的微处理器执行某些操作的情况。它们都执行下面的操作：

- 运算
- 转移（布尔值）
- 输入／输出操作

我们已经讨论了有关整数编码的二进制运算，正整数和

负整数都用2的补码来表示，实数则用浮点来表示。

假定我们要进行下面的工资计算

工资问题：将工资率乘以工时。

如果工时超过40，则加上工资率的50%与加班工时的乘积。

这个问题包含了所有三个基本操作。首先，我们必须输入工资率和工时；其次，必须执行必要的运算；最后必须输出答案。这几步就是：

步骤1.输入工资率和工时

步骤2.计算：工资＝工资率×工时

步骤3.如果工时超过40

则　3a）加班工时＝工时－40

加班工资＝（工资率／2）×加班工时

否则　3b）加班工资＝0

步骤4.修正工资值：工资＝工资＋加班工资

步骤5.输出工资数

执行该程序步骤的微处理器指令列在下面（不过这里是以语言而不是用开关编码来表达）。

步骤1

a.等待键盘输入

b.输入一个字符

c.将输入的数转换成整数或实数

d.按上面的步骤输入工资率和工时

e.将工资率和工时数存储在RAM中

步骤2

a.将工时和工资率送入寄存器

b.做乘法：工资率×工时

c.将结果送入RAM中工资数所对应的单元

步骤3

a.将工时送入寄存器

b.将工时与40比较

c.如果比较值为真则转至3a）

d.如果比较值为假则转至3b）

则3a

a.做减法：工时－40　结果存入加班工时单元

b.将工资率送入寄存器並除以2

c.做乘法：（工资率／2）×加班工时

d.将结果存入RAM中加班工资所对应的单元

否则3b）

a.将0存入RAM中加班工资所对应的单元

步骤4

a.将工资数送入一个寄存器

b.将加班工资送入另一个寄存器

c.将上两个寄存器的内容相加，並将结果存入RAM中工资数的单元

步骤5

a.将工资数从RAM中取出送入寄存器

b.将工资数的整数值或实数值换成字符串

c.重复上面的步骤直到字符串全部在显示屏上显示出来：

c.i.　等待显示屏就绪

c.ii.　将字符输出给显示屏

这些操作大致上与微处理器实际执行情况相符。正如你所看到的那样，具体地做起来工作量很大，而上面的例子还算是很简单。要想成功地做好上面这些琐碎的事很困难，除非我们有一个强有力的软件工具。这就是高级语言像BASIC、Pascal和COBOL产生的原因。这些工具给了我们一个“杠杆”，加速了软件的生成，从而免除很多繁琐的工作。

计算机的运算与我们在学校中所学过的那种运算大不相同。计算机的运算包含以下内容：它将信息在不同级别的存储之间从一个地方传送另一个地方；将开关量编码和译码；比较接着就转移。真正的加法等运算只是计算机运算中的很小一部分，在计算工资的例子中只不过占16%。事实上，微处理器所执行的多数公共的操作是传送。就是说，大约占微处理器所执行的操作的40%是将信息在存储器中从一个地方传送另一个地方。

芯片技术

我们已经讨论了计算机是如何工作的，但是工程师又怎样制造一个实现上述功能的微处理器芯片呢？为了进一步讨论这个问题，我们必须学习某些硬件知识。

制造微处理器和RAM存贮器的芯片技术是照相术、化学，电子学和数理逻辑的高度结合。我们将在最基本的方面讨论这一综合技术。

简单地说，制造一块芯片须经过下述步骤

1、电路设计师用制图计算机作为辅助手段绘制出存贮

器或处理器的电路图。

2、用其他计算机测试该电路以确保他们工作正确。

3、把测试过的电路绘制在照相介质上，然后用光缩工艺加以缩小，在这一光缩过程中，电路被缩小几千倍，从而使“大”电路变成微电路。

4、经连续若干次光敏化学蚀刻，把微电路“显影”到称之为衬底的硅基片上。

5、清洗和测试蚀刻板以拣出合格品。然后把“产品”封装到双列直插芯片中。这些芯片将作为处理器、存贮器等等用于计算机中。

无论是微处理器芯片或是RAM存贮器芯片，其基本电路都是一个能“记忆”开一关状态的电路。为此我们采用了晶体管微电路。其概念很简单，当晶体管“导通”时允许电流流通，而当其“断开”时阻止电子流动。

图8—2是一个晶体管的简图。晶体管的基片或衬底是带有过量负电荷(过多电子)的硅化合物。图中，我们用N(负)电荷表示。图中的P(正)电荷或缺少电子)区是在衬底上刻蚀出的沟道。同时腐蚀成晶体管的还有在P区上方的金属控制端，这个控制端通过轻微改变电势来控制越过沟道的电子流。

只要沟道中缺少电子，源区电子将流进沟道中。若漏区电位比沟道的电位低，电子将继续流入漏区(称为沉)。倘若在金属控制端加上电压，沟道的电位就发生改变，使电子受到“位垒”的阻挡。。这个“位垒”阻止电子的流动从而使晶体管断开。

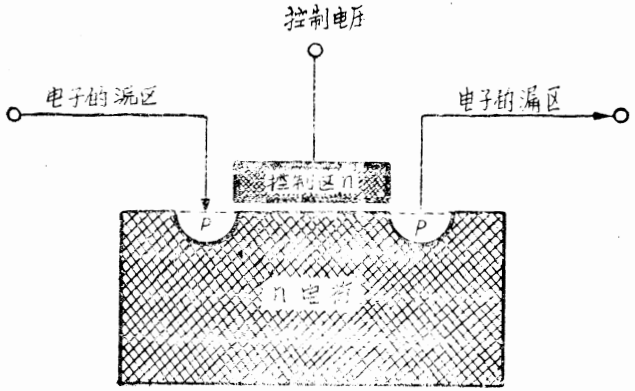
我们必须为计算机存贮器的每一位构造一个或多个不是

接通（1）就是断开（0）的晶体管。微电子学的技术革命有可能在指甲大小的晶片上刻印数百万个微小的晶体管。下一步是模仿计算机运算的方法把这些晶体管连到一起。

布尔连接法

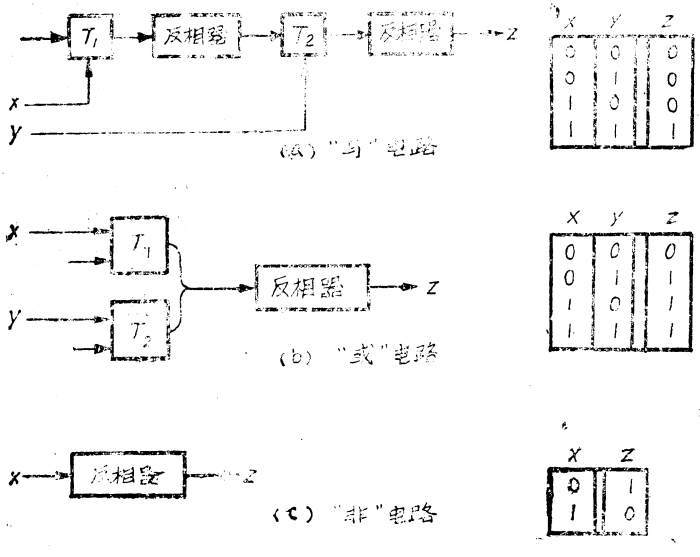
假如要计算机作加法，我们就必须按一定的逻辑规律把晶体管连接起来。乔治·布尔发明了这些逻辑规律。人们称这些规律为布尔代数以向他表示敬意。为看出这些规律与晶体管之间怎样联系，我们考察下述例子。

布尔系统地阐述了一组定律，这些定律特别适用于通一断开关，它们能支配用晶体管实现的与、或和非电路的代数运算。例如，串联连接的二个晶体管构成与电路，见图8—3（a）。并联连接的二个晶体管构成或电路，见图8—3（b）



晶体管的简单模型　2—8图

在图8—3中我们把x和y的输入连接到每个晶体管的金属控制端，使得加上电压时（由1来表示），电子不能从源流到漏。没有电子流是0，有电子流是1。反相电路把1变换为0，把0变换为1。



x	y	z
0	0	0
0	1	0
1	0	0
1	1	1

x	y	z
0	0	0
0	1	1
1	0	1
1	1	1

x	z
0	1
1	0

图8—3　布尔组件块

表中给出了对各种输入值的计算结果。例如，

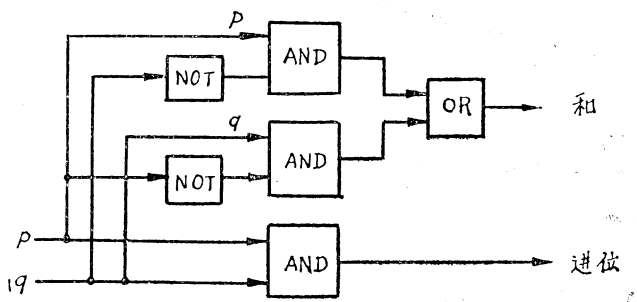
$$1 \times 1 = 1$$

$$1 + 0 = 1$$

因为这些输入允许电子流动（即为1）。

布尔电路的最大特点是它能模拟计算机的运算。例如，

图8—4示出的布尔电路能把两个二进制位相加得到和及进位。



p	q	和	进位
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

图8—4　简单（半）加法器

电路设计师应用八个象图8—4那样的简单加法器模拟两个八位字节的加法。每个加法器计算其输入位的和及产生一个进位。把进位加到上一位的一个输入位上，然后上一对输入位相加，如此等等

这个例子说明如何应用布尔代数设计刻蚀在芯片基片上的晶体管电路。把晶体管连在一起，就构成了一个符合布尔表达式的电路。布尔表达式又模拟我们在学校学过的普通的运算。

当然，芯片计算机的布尔电路比这里给出的电路要复杂得多。然而其基本原理是相同的，只要给出足够的时间并经过努力，任何人都能设计出他所希望的计算电路。例如，图8—4中和的布尔表达式是：

$$SUM=(p\times\bar{q})+(\bar{p}\times q)$$

进位是：

$$CARRY=p\times q$$

总　结

现在让我们归纳一下，看看计算机是如何工作的。首先，要设计一组电路，该电路以开关晶体管作为基本的存贮单元。这些晶体管连成的网络构成布尔电路，这些电路遵守与、或、非逻辑定律。如果连接正确，那么这些布尔电路就能做几乎所有的计算机运算。

一个开关晶体管电路保存单个信息位。这些位的集合可用来编码整数、实数、字符等等。此外，还有一些位是用于存贮指令字的。为执行指令字，微处理器电路采用了一种序列发生器。指令字修改存贮器中的数据字并使计算机“计算”。

程序员设计数据结构和称为程序的指令序列，程序则控制微处理器的每一步操作。这些程序都必须从一种编码翻译成另一种编码，能够控制输入和输出及程序中发生的转移。一般说来，这是软件的任务，且由此也产生了对软件生产工具的需要。

程序员借助文本编辑、高级语言翻译器和操作系统这些专用程序来开发其他程序。然而对于这些软件系统的运行都

依赖于低位。因此，该位以及怎样用该位编码在计算机存贮器内的信息视计算机是如何工作的情况。

常 见 问 题

问　为什么必需要有不同于RAM存贮器的寄存器？

答　把RAM的每个字都与微处理器相连太浪费，因此代之以用少量称为寄存器的特殊字来连到微处理器。所以，处理在RAM中的每个字首先必须传送到寄存器，完成操作后再传回到RAM。

问　为什么要用ASCII方法编码字符？例如，为什么数字“5”的字符编码是53而不是5？

答　ASCII码是用于电传打字机而不是用于计算机的标准码。因而，这种码是为了简化机械式打印机结构而设计的。这种机械打印机早已被人们遗忘。

问　计算机是以象晶体管接通—断开的开关规则这样的简单机理为基础的，那为什么它们显得如此复杂呢？

答　有两个因素使计算机看起来显得复杂：（1）抽象；（2）规模。高级语言的编码方案、软件和计算机程序员使用的符号造成了极度的抽象。这样层层抽象使得初学者难以从一些枯燥的编码中想象出“实在”的机器。其次，计算机系统由许多部分组成，如可以把大量信息装到64K的存贮

器中。

问　字节和字的的区别是什么？

答　一个字节信息通常对应于8位，而一个字通常对应于一个以上的字节，如二个或四个字节。

问　RAM与ROM的区别是什么？

答　RAM是可改变的存贮器，能对它进行读和写。而ROM是不可改变的存贮器，仅能从ROM中读出，不能对其写入。

问　计算机系统为什么一定使用分级结构存贮器？

答　这是考虑成本。一般来说存贮器的速度越快，其构造费用就越高。因此要利用体积小，速度快的器件，如用RAM作主存贮器，而用磁盘那样大的、中速的存贮器作大容量存贮器。之所以这样其原因是磁盘存贮器的每位成本较之RAM的低得多。

问　计算机怎样把它的存贮器中的数据和指令区分开来？

答　在存贮器中指令和数据字二者的存贮方式没有差别。因此，程序员要进行人工区分。另方面，程序员可能把数据当作程序一样执行。这是控制计算机的一个非常危险的方法。

问　为什么在计算机中应用如此多的编码方案？

答　这部分地是由于效能，部分是因为历史原因。例如，若以整数与实数编码相比较，二的补码用于整数而浮点用于实数则硬件电路更易于实现。

问　为什么计算机使用二进制数系统而不用十进制数

系统？

答　这是效率问题。过去曾制成过少量十进制计算机，但最有效的编码方案是用二进制表示法。第二个优点是它们必须用电子设备构成，例如，用晶体管构成一个接通一断开开关比构成一个十种方式的开关就容易得多。

问　怎样按二进制数写57？

答　二进制：32＋16＋8＋1＝111001

问　二进制111010的八位二的补码是什么？

答　把最后（最右边）为1的位之前的各位变反。因此，111010是00111010，其补码形式是11000110。

问　什么是串？

答　串是一组字符。

问　数组是什么？

答　数组是值的序列（全部同一类型的信息），在存贮器中它们通常是存放在一起的。

问　程序是什么？

答　程序是控制微处理器操作的指令序列。

问　例行子程序是什么？

答　例行子程序有时称为子程序、过程或函数。子程序是由另一程序调用的程序。

问　晶体管如何编码1位信息？

答　通过控制接通一断开电子的流动编码1位信息，接通为1，断开为0。

问　计算机怎样处理象文字那样的字母信息？

答　通过把字母文字编码为数字码（ASCII），然后

象任何数字一样处理该数字代码。

问　什么是数据结构？

答　数据结构是数据和信息的集合，该信息告诉程序如何访问存贮在数据结构中的值。例如，数组是值的集合，数组中的值是依次存取的。

问　计算机如何执行运算，如怎样作加法？

答　用与、或和非操作的布尔电路模拟加法。不管它们是整数还是实数，加法电路用编码来工作。

问　计算机的工作情况如何？

答　工作很正常，谢谢。

3.60元