

MICROPOWER SERIES

DATA AND FILE MANAGEMENT FOR THE TI-99/4A®

John P. Grillo/J. D. Robertson/Henry M. Zbyszynski



Includes 48 programs to give the more advanced user techniques for information management.

**REQUIRED FOR
THIS BOOK:
Extended Basic
Cartridge
Memory Expander
Disk Controller
and Drive**

MICROPOWER SERIES

*DATA AND FILE MANAGEMENT
FOR THE TI-99/4A[®]*

John P. Grillo J. D. Robertson Henry M. Zbyszynski

wcb

Wm. C. Brown Publishers
Dubuque, Iowa

Consulting Editor:
Edouard J. Desautels
University of Wisconsin-Madison

Cover photo by Bob Coyle

Micropower Series

Copyright © 1984 by Wm. C. Brown Publishers. All rights reserved

Library of Congress Catalog Card Number: 84-70104

ISBN 0-697-00245-4

2-00245-01

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of the publisher.

Printed in the United States of America

To Ann

Contents

Introduction vii

1 Pointers **1**

Subscripts 1
Nim, G1A, Heuristic Programming 2
Monte Carlo Methods 6
Random Text 10
Random Message Selection 13
Normal Distribution of Values 17

2 Sorting **21**

Sorting Categories 21
Brute Force Sorts 22
Exchange Sorts 24
Binary Sorts 27
Tree Sorts 31
Multikey Sorts 32
Summary 35
References 37

3 Strings **39**

Word Processing 39
Random Word Selection 40
Pattern Matching 42
Text Encoding 46
Text Reordering 47
Text Analysis 50

4 Linear and Linked Lists **55**

Stacks 55
Queues 57
Dequeues 59
Linked Lists 60
Singly Linked Lists 60
Doubly Linked Lists 61
Circularly Linked Lists 62
Circular Doubly Linked List Application 62

5	Sequential Access Files	69
	Sequential Search Techniques	70
	Sequential File Access	71
	Group Totals	75
	Sequential File Merging	78
	Index Production	81
6	Direct Access Files	89
	File Searching	90
	Binary Search	90
	Interpolation Search	92
	Hash Address Processing	93
	Sorting Large Files	96
	Disk Sort	96
	Detached Key Sort	98
	Segmented Detached Key Sort	100
	ISAM File Processing	102
	ISAM Storage Areas	103
	DOS Physical Characteristics	103
	ISAM Structuring	103
	ISAM Access	104
	Overflow Area	105
	ISAM Insertion	105
7	Trees	107
	Binary Trees	108
	Binary Sequence Search Tree	114
	In-Memory, Single-Key BSST	115
	In-Memory, Double-Key BSST	116
	In-Memory, Multi-Key BSST	120
	BSST on Disk	122
	Tree and Circularly Linked List	124
8	Inverted Files	131
	Secondary Keys	131
	Record Structure	132
	Record Contents	133
	File Access Using Pointer Table	134
	Main Program Driver	134
	Deletion and Balancing	135
	Record Insertion	135
	Random Access	138
	Sorted Order Display	141
	Physical Record Display	148
	Final Thoughts	149
	Index	151

Introduction

What is a data structure? How can the understanding of a search technique help you to write a better genealogy program? What in the world are stacks, queues, deques, and trees? Do you lose the contents of a file if you invert it?

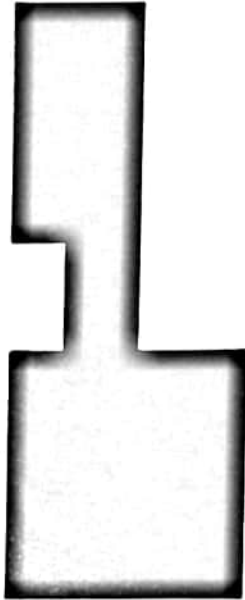
There is a reason why programmers ask these questions. The questions stem from a lack of understanding of some fundamental programming concepts that deal with data. Most programmers rely on a background of vendor manuals and perhaps one or two formal courses in BASIC. They feel confident in their ability to deal with lists, arrays, subscripts, and some sequential searches. But unfortunately this level of programming expertise, this repertoire of techniques, is not enough to be helpful in writing good, efficient software.

In the classroom and in the world of consulting, we are quick to point out the crucial importance of writing usable programs, that is, programs that benefit the user. We also emphasize that the programmer is rarely the only user of a good program, because a good program is by definition one that is used by many people. A good programmer must take pains to write programs that are easy to use. A good program has the following properties:

- It should be structured well, so that its author and all other readers of the program feel confident in being able to change parts of it — add, delete, or modify modules — without adversely affecting the untouched portions.
- It should be documented well, so that its logic can be understood easily.
- It should be written to be interactive whenever this process can benefit the user.
- It should use files if these media for storing information are appropriate.
- It should make efficient use of the computer through the proper use of algorithms that minimize sorting and searching times, minimize disk accesses, avoid excessively large memory arrays, and avoid inappropriate data storage techniques.

In most colleges and universities there exists a course called Data Structures. It is taught after two semesters of programming and its intent is to refine the students' technique and introduce the commonly used procedures that have been developed over the years to make a program run more efficiently. The course covers pretty much what this book covers, and in pretty much the same order. After mastering these techniques, these same students seem to be able to deal with masses of data, either in memory or on files, with considerable ease and success. The Data Structures course, more than any other course in their formal training, prepares them to write user-oriented programs with direct applicability in their future industrial exposure.

In this book we have made a sincere effort to simplify and demystify the subject of data structures. We call the book *Data and File Management* because we are trying to make a point: The topic of data structures is useful only as long as it relates to the practical aspects of how to manage data. The programs we include as examples should serve to show you how these techniques can be useful in many common applications. We sincerely hope that some of these intrigue you enough that you will adapt them in some novel fashion and that you have as much fun using them as we had writing them.



Pointers

BASIC has become the popular problem-solving language for microcomputers for a wide variety of reasons, not the least of which is its inherent simplicity. It is easy to learn and to use. Consider the long list of high schools and colleges that teach BASIC as a way to introduce the computer to novices.

One important reason for BASIC's popularity is often overlooked: The language is highly flexible. By this we mean that BASIC will allow programming constructs that are difficult or impossible to manage in another language. A case in point is an array's subscript. In BASIC, the only rule is that the subscript be a numeric expression, while in many versions of FORTRAN, for example, the subscript form is limited to but a few very simple variations.

Subscripts

The subscript is the programmer's pointer into an array, and as such must be capable of as much variation as possible.

The subscript as a pointer in its most elemental form is simply a way to access a given array element. For example, consider this segment of code:

```
100 DIM D(50)
110 P = 17
120 V = 4
130 D(P) = V
```

The variable P assumes the role of pointer to the array D in line 130 when it is used as a subscript. The overall effect is to store the value 4 into the 17th location of D.

Suppose this line were added:

```
140 D(D(P)) = 237.8
```

This time, the pointer to the array D is the variable D(P), which in itself uses a pointer. P = 17, as defined in line 110 above.

D(P) = D(17) = 4, from lines 120 and 130. D(D(P)) = D(4), so define D(4) as 237.8. Thus the value 237.8 is stored in D(4). This form of addressing an array is called *indirect addressing*, because the computer determines the final destination by proceeding through an intermediate location that points to the value to be transferred.

In order to complete this introduction to pointers, we should point out that most versions of FORTRAN don't allow subscripted variables to have subscripted variables as subscripts. Try saying that ten times fast!

The programs which we have selected to include in this chapter to exemplify the use of pointers all have a common concern: Where do the array pointers come from? You will discover that array pointers can be selected from a pool, generated at random, or calculated. This differs from the more common source of array pointers, such as a FOR-NEXT loop index or a counter.

Nim, G1A, and Heuristic Programming

The first program, G1A, is an interesting variant of the game of Nim. The two Nim players remove from one to three objects from a starting set of 13 objects. The player who removes the last object loses. The game can always be won by the second player if that player remembers one rule: Always leave a pile with the number of objects left equal to 9, 5, or 1.

In the early days of computers, much discussion centered on how to program these machines to assume the characteristics that would make them seem intelligent. The area of interest, called *Artificial Intelligence*, or AI, was born. One of the techniques for simulating intelligence was given the name *heuristic programming*, or programming with the intent to discover or reveal an underlying principle.

In 1965, H. D. Block in an article in *American Scientist* described a machine that would "learn" to play the game of Nim with a winning strategy. The machine, originally called G1, did a pretty fair job of imitating the way we humans learn. At first, it would play in seemingly random fashion. After many games it would give up the current game as lost before the game was over. It was as if it had discovered the futility of continuing that game. Then, many games later, it became unbeatable. It had "learned" the way to win.

To speed up the learning process, Block altered G1 and created G1A. We present this machine to you now as a program. The program sets up all possible moves for itself in four cups. Think of each cup containing three slips of paper, marked 1, 2, and 3. As a game proceeds, G1A "draws" its move from the appropriate cup at random.

When the flow of the game determines that G1A has lost, the last "draw" that G1A made before losing is marked with a -1. From that point on in the series of games, this move will not be made. Eventually, all cups' moves contain a -1 except those that lead to wins, and those will be the only plays G1A makes.

Part of the fun of this program is to trace G1A's progressively better play. We leave this as an exercise for the reader.

Some features of the program are worthy of special mention.

- All user inputs are programmed with the CALL KEY function. Study the portion of the program that asks for the user's initials. This section uses CALL KEY to build an input string three characters long without the use of the ENTER key.
- The DISPLAY instruction is used extensively to display the game's status as it changes.
- Graphic characters are used to display the chips. Note that a special effort is made to remove the chips randomly from the pile.
- The messages that G1A displays give it a personality. If G1A loses, it responds in modest lower case. If G1A wins, it responds in an obnoxious and pretentious upper case.

```

10 REM filename: "g1a"
20 REM purpose: To play the game of "NIM" heuristically
30 REM author: jpg & jdr 9/82
40 REM -----
50 RANDOMIZE :: CALL CLEAR
60 DIM CUP(4,3),CY(13),CX(13)
70 DIM PERM(6,3)!6 permutations of 3 digits.
80 ! His win or our win messages.
90 DIM H1$(6),H2$(6),O1$(13),O2$(13),O3$(13)
100 ! Flash the title scree.
110 DISPLAY AT(5,1):RPT$("=",28)
120 DISPLAY AT(7,5):"N I M W I T H G 1 A"
130 DISPLAY AT(9,1):RPT$("=",28)
140 ' Fill each set of 3 cups with digits 1,2,3.
150 FOR I=1 TO 4
160 FOR J=1 TO 3 :: CUP(I,J)=J :: NEXT J
170 NEXT I
180 'Fill the PERM array with all permutations of 1,2,3.
190 FOR I=1 TO 6
200 FOR J=1 TO 3 :: READ PERM(I,J):: NEXT J
210 NEXT I
220 ! Fill his win message with gracious pap.
230 FOR I=1 TO 6 :: READ H1$(I),H2$(I):: NEXT I
240 ! Fill our win message with scathing scorn.
250 FOR I=1 TO 13 :: READ O1$(I),O2$(I),O3$(I):: NEXT I
260 DISPLAY AT(12,1):RPT$("-",16):: WHO$="/"
270 ! Set up the player's name.
280 DISPLAY AT(12,3)BEEP:"Enter your initials";
282 CALL KEY(0,T,S):: IF S=0 THEN 282
284 T%=CHR$(T)
290 IF T=13 THEN 320
300 WHO%=WHO%&T% :: DISPLAY AT(12,22):WHO%
310 IF LEN(WHO%)<>6 THEN 270

```

```

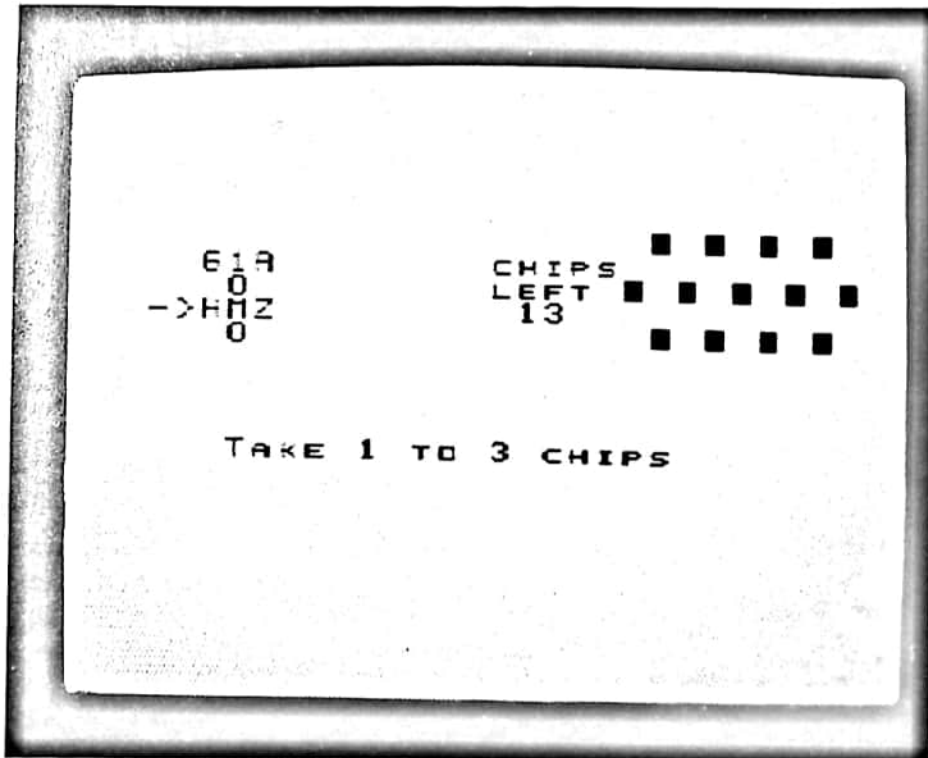
320 GOSUB 5000 :: WHO%=SEG$(WHO$,LEN(WHO$)-2,3):: CALL CLEAR
330 DISPLAY AT(7,4):"G1A";'The computer's name is G1A.
340 DISPLAY AT(9,4):WHO$;'Print the player's name.
350 DISPLAY AT(8,4):WIN;'Print number of player's losses.
360 DISPLAY AT(10,4):LOSE;'Print number of player's wins.
370 GOSUB 1000 'Blank out previous chips.
390 !           Position the chips.
400 DISPLAY AT(7,2):" ";
410 DISPLAY AT(9,2):"->";
420 !           Get player's response.
430 DISPLAY AT(15,5)BEEP:"Take 1 to 3 chips";
432 CALL KEY(0,T,S)
434 IF S=0 THEN 432
436 T%=CHR$(T)
440 IF T%="" THEN 420 ELSE TAKE=VAL(T%): DISPLAY AT(15,25):"/";T% :: GOSUB 5000
450 IF TAKE<1 OR TAKE>3 OR TAKE>CHIPS THEN GOSUB 6000 :: DISPLAY AT(15,5):"impro
per move";:: GOSUB 5000 :: GOTO 420
460 GOSUB 2000
470 IF CHIPS<=0 THEN GOSUB 4000 :: GOTO 330
480 ! G1A's move
490 GOSUB 6000 :: DISPLAY AT(9,2):" ";
500 DISPLAY AT(7,2):"->";
510 !           A is the random permutation selector.
520 GOSUB 5000 :: A=INT(RND*6+1)
530 FOR I=1 TO 3
540 !           Check the Ath. cup.
550 !           B is the remainder: Chips modulo 4.
560 K=PERM(A,I):: B=CHIPS-4*INT(CHIPS/4)
570 IF B=0 THEN B=4
580 !           If this cup is not empty check others.
590 !           If it is empty, check others.
600 IF CUP(B,K)>0 THEN 610
605 NEXT I :: CUP(BCUP,KCARD)=-1 :: HOW=1 :: GOSUB 3000 :: GOTO 330
610 TAKE=CUP(B,K)
620 !           If this cup is not empty, check others.
630 IF TAKE>CHIPS THEN CUP(B,K)=-1 :: HOW=2 :: GOSUB 3000 :: GOTO 330
640 !           Note proper grammatical display. 'chip' or 'chips'.
650 DISPLAY AT(15,5):"G1A takes";TAKE;SEG$("chips",1,4+INT(TAKE/2))
660 GOSUB 5000 :: GOSUB 5000
670 BCUP=N :: KCARD=K
680 GOSUB 2000 :: GOTO 390
1000 !***** Subroutine to set up chips. *****
1010 N=0
1020 FOR I=-1 TO 1
1030 L=ABS(I):: M=46+L
1040 FOR J=1 TO 5-L
1050 N=N+1 :: CX(N)=M+J+J :: CY(N)=I+I+8
1060 DISPLAY AT(CY(N),CX(N)):CHR$(30)
1070 NEXT J
1080 NEXT I
1090 CHIPS=13 :: DISPLAY AT(7,15):"chips";:: DISPLAY AT(8,15):"left";
1100 DISPLAY AT(9,15):CHIPS;:: R=INT(RND*13+1):: P=1+INT(RND*11+1)
1110 RETURN
2000 !***** Subroutine to remove 1 to 3 chips.*****
2010 FOR I=1 TO TAKE
2020 R=R+P
2030 IF R>13 THEN R=R-13
2040 DISPLAY AT(CY(R),CX(R)): " ";
2050 NEXT I
2060 CHIPS=CHIPS-TAKE :: DISPLAY AT(9,15):CHIPS
2070 RETURN
3000 !***** Subroutine to print loss message.*****
3010 GOSUB 5000 :: GOSUB 5000 :: CALL CLEAR
3020 A=INT(RND*6+1):: DISPLAY AT(7,1):RPT$("~",28)
3030 IF HOW=2 THEN DISPLAY AT(9,1):"G1A ";H2$(A);" acknowledges defeat";
3040 IF HOW=1 THEN DISPLAY AT(9,5):"G1A ";H1$(A);" concedes the game"
3050 DISPLAY AT(12,1):RPT$("~",28):: LOSE=LOSE+1
3060 GOSUB 5000 :: GOSUB 5000 :: GOSUB 5000 :: CALL CLEAR
3070 RETURN

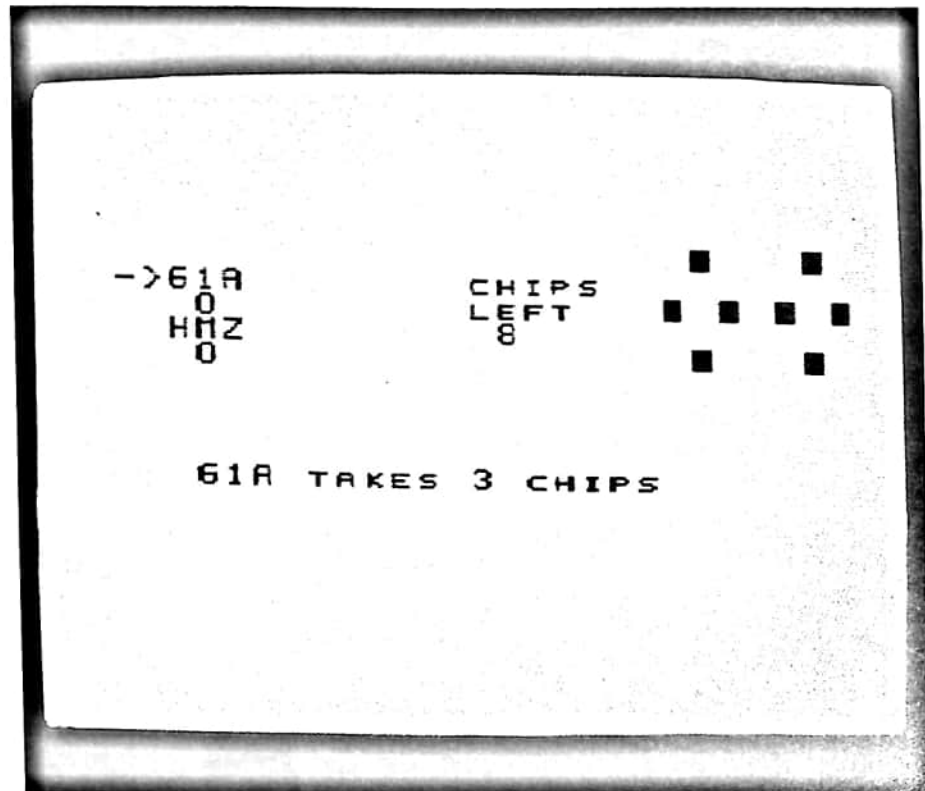
```

```

4000 !***** Subroutine to print win by G1A.*****
4010 CALL CLEAR :: A=INT(RND*13+1):: B=INT(RND*13+1):: C=INT(RND*13+1)
4020 DISPLAY AT(7,1):RPT$("#",28)
4030 DISPLAY AT(9,1):"THE ";O1$(A); " G1A HAS ";O2$(B); " THE ";O3$(C); " ";WHO$
4040 DISPLAY AT(11,1):RPT$("#",28)
4050 WIN=WIN+1
4060 GOSUB 5000 :: GOSUB 5000:GOSUB 5000 :: CALL CLEAR
4070 RETURN
5000 !***** Subroutine to mark time *****
5010 FOR I=1 TO 500 :: NEXT I
5020 RETURN
6000 !***** Subroutine for blanking out *****
6010 DISPLAY AT(15,1):RPT$(" ",28);
6020 RETURN
7000 !***** Data Statements *****
7010 !
7020 !
7030 DATA 1,2,3,1,3,2,2,1,3,2,3,1,3,1,2,3,2,1
7040 ! Data for G1A losses
7050 DATA cordially,respectfully,graciously,politely
7060 DATA affably,humbly,cogenially,modestly
7070 DATA meekly,amicably,courteously,agreeably
7080 ! Data for G1A wins
7090 DATA AWESOME,ANNIHILATED,PROSAIC,DREDED
7100 DATA EXTERMINATED,VAPID,PUISSANT,OBLITERATED,SLUGGISH
7110 DATA EMINENT,DEMOLISHED,DOLTISH,EXALTED,CONQUERED,ORTUSE
7120 DATA INTREPID,VANQUISHED,INFERIOR,SPLENDID,DEVASTATED
7130 DATA INSIPID,SAPIENT,EXTIRPATED,MAWKISH,ERUDITE,SUBJUGATED
7140 DATA BUNGLING,FORMIDABLE,CRUSHED,FLACCID,REDOUBTABLE
7150 DATA FLATTENED,INEPT,BRILLIANT,STOPMED,IGNORANT
7160 DATA MAGNIFICENT,DESTROYED,STUPID
9999 END

```





Monte Carlo Methods

The next program, `JOBSTEPS`, demonstrates the use of random numbers as pointers to distribute an array's contents. The function of the program is to determine two sequences of hypothetical machining operations. Each of the sequences is to be assigned to one of two workers, with a sense of fairness requiring that the total time for the machining operations each is assigned be as closely matched as possible. There are as many as 25 various operations the two workers are qualified to do, and each worker can be assigned any number of these operations, as long as they both work the same total amount of time. Their boss, the user of this program, inputs the amount of time allotted, for example four hours. The program prints out two work schedules, each containing machining operations, or tasks. No task on one list appears on the other.

The technique of successively scrambling an array's contents, then checking to see if this order produces a better solution than a previous one, is an example of the *Monte Carlo Technique*, named after the famous casino at Monaco.

The solution of this problem is not trivial. To come up with such a schedule with paper and pencil takes the better portion of an hour. What the computer does in the program `JOBSTEPS` is to select at random a set from 25 possible operations and sum their times. As soon as the set exceeds the total time the boss dictates, for example four hours, the total time is displayed. The boss can elect to have the computer program select another set closer to the four hours, or accept that one.

When the first worker's schedule has been determined in this manner, the program uses that amount of time as a target and randomly selects from the unused tasks another schedule for the second worker. The computer displays its first try, and the boss can accept or reject it. A rejection forces the computer to come up with a better schedule. Each try that is closer to the target (worker 1's schedule) is displayed for the boss to accept or reject. When the boss finally accepts that run, both workers' schedules are displayed, with each total time shown.

```

10 REM filename jobsteps
20 REM purpose: Monte Carlo selection of job operations
30 REM author: jpg & jdr 9/82
40 REM -----
50 DIM T$(25),T(25),K(25)
60 REM T$ = operation          T = time per operation, min.
70 REM K$ = selected pointer
80 RANDOMIZE :: CALL CLEAR
90 REM                          Read tasks, times.
100 INPUT "What is target time (in minutes)":TT
110 FOR I=1 TO 25
120 READ T$(I),T(I):: K(I)=0
130 NEXT I
140 S=0
150 T1=10 !Set difference between time S and TT
160 GOSUB 1000
170 PRINT "Suggested time is";S
180 INPUT "Is this acceptable? (y=yes)":A$
190 IF A$<>"y" THEN T1=ABS(S-TT):: GOTO 160
200 FOR I=1 TO 25
210 IF K(I)>0 THEN K(I)=-K(I)!Mark this the first pass.
220 NEXT I
230 GOSUB 1000
240 PRINT "Suggested time is ";S
250 INPUT "Is this acceptable? (y=yes)":A$
260 IF A$<>"y" THEN T1=ABS(S-TT):: GOTO 230
270 PRINT "Schedule for both workers"
280 S1=0 :: S2=0
290 FOR I=1 TO 25
300 IF K(I)<0 THEN P=ABS(K(I)):: PRINT T$(P),T(P):: S1=S1+T(P)
310 NEXT I
320 PRINT "Sum, Worker 1: ";S1
330 FOR I=1 TO 25
340 IF K(I)>0 THEN PRINT T$(K(I)),T(K(I)):: S2=S2+T(K(I))
350 NEXT I
360 PRINT "Sum, Worker 2: ";S2
370 STOP
500 DATA "stamping",31.7,"spooling",42.0,"flanging",25.4
510 DATA "milling",40.1,"cutting",32.5,"degreasing",24.7
520 DATA "pithing",34.8,"polarizing",30.3,"rolling",31.7
530 DATA "cascading",22.2,"waafting",44.8,"leveling",15.0
540 DATA "plating",29.1,"chafing",38.2,"fluting",38.5
550 DATA "sanding",53.9,"bending",26.5,"stressing",27.7
560 DATA "testing",51.4,"polishing",20.1,"packing",44.2
570 DATA "Blunting",32.2,"merging",37.8,"gnashing",23.4
580 DATA "flushing",25.0
1000 REM ***** Sunroutine to return S, sum within T1 of TT. *****
****

```

```

1010 S=0
1020 FOR I=1 TO 25
1030 IF K(I)>0 THEN K(I)=0
1040 NEXT I
1050 R=INT(RND*25+1)!Generate a random number.
1060 REM Find out if it has been used.
1070 IF K(R)<>0 THEN 1050 !It has -- get another.
1080 S=S+T(R):: K(R)=R !Sum this random time.
1090 IF S>TT THEN IF S-TT<T1 THEN RETURN ELSE 1010 ELSE IF TT-S<T1 THEN RETURN E
LSE 1050
9999 END

```

```

What is target time (in minutes) 120
Suggested time is 113.4
Is this acceptable? (y=yes)n
Suggested time is 125.9
Is this acceptable? (y=yes)n
Suggested time is 124.8
Is this acceptable? (y=yes)y
Suggested time is 123.8
Is this acceptable? (y=yes)y
Schedule for both workers
packing 44.2
Blunting 32.2
gnashing 23.4
flushing 25
Sum, Worker 1: 124.8
pithing 34.8
leveling 15
sanding 53.9
polishing 20.1
Sum, Worker 2: 123.8

```

An alternative problem could be to distribute an estate equally between two heirs. Program INHERIT is modeled after a similar program in Chapter 2 of this series' BASIC book. That program distributed songs equally in time over two sides of a record.

In this program, the 25 various items in the deceased's estate are described and their estimated monetary value listed in the DATA statements. The program scrambles pointers to these items as many times as the estate's executor deems necessary. Then the program lists two equally valuable shares of the estate.

```

10 REM filename: "inherit"
20 REM purpose: Monte Carlo selection of equal inheritances
30 REM author: jpg & jdr 9/82
40 REM -----
50 DIM T$(25),V(25),K(25),L(25)
60 REM T$ = item description V = monetary value
70 REM K$ = random pointer L = second random pointer
80 RANDOMIZE :: CALL CLEAR
85 OPEN #1:"RS232"
90 REM Read the descriptions and values
100 REM N = total number of items, both inheritors
110 INPUT "How many of the 25 items to be distributed?":N
120 PRINT #1:"This distribution has";N;"of the 25 items."
130 FOR I=1 TO 25
140 READ T$(I),V(I):: K(I)=I
150 NEXT I

```



```

160 IMAGE Smallest difference is $##### in try ###
170 C=6000 ! Set difference between inheritors very high.
180 INPUT "How many scrambles?":N5
190 PRINT #1:"Selected number of scrambles = ";N5
200 FOR Q=1 TO N5
210 FOR I=1 TO N
220 J=INT(RND*N+1):: Z=K(I):: K(I)=K(J):: K(J)=Z
230 NEXT I
240 REM      Sum the values for the two inheritors.
250 Z1=0 :: Z2=0 :: N2=INT(N/2)
260 FOR I=1 TO N2
270 J=K(I):: Z1=Z1+V(J):: J=K(I+N2):: Z2=Z2+V(J)
280 NEXT I
290 IF N/2<>INT(N/2) THEN Z2=Z2+V(N)
300 B=ABS(Z1-Z2)! B = diff. in sum between inheritors.
310 IF B>C THEN 370 !C = previously smallest diff.
320 C=B :: C1=Z1 :: C2=Z2
330 FOR I=1 TO N :: L(I)=K(I):: NEXT I
340 PRINT USING 160:C,Q
350 PRINT #1,USING 160:C,Q
360 IF C=0 THEN 380
370 NEXT Q
380 PRINT :: PRINT
390 PRINT #1 :: PRINT #1
400 IMAGE "      Estate#:"
410 IMAGE "##### $#####.##"
420 F3$="      ====="
430 IMAGE "      Total      $#####.##"
440 PRINT USING 400:1 :: PRINT #1,USING 400:1
450 FOR I=1 TO N2 :: J=L(I)
460 PRINT USING 410:T$(J),V(J):: PRINT #1,USING 410:T$(J),V(J)
470 NEXT I
480 PRINT F3$ :: PRINT #1:F3$
490 PRINT USING 430:C1 :: PRINT #1,USING 430:C1
500 PRINT :: PRINT :: PRINT #1 :: PRINT #1
510 PRINT USING 400:2 :: PRINT #1,USING 400:2
520 FOR I=N2+1 TO N :: J=L(I)
530 PRINT USING 410:T$(J),V(J):: PRINT #1,USING 410:T$(J),V(J)
540 NEXT I
550 PRINT F3$ :: PRINT #1:F3$
560 PRINT USING 430:C2 :: PRINT #1,USING 430:C2
570 CLOSE #1 :: STOP
575 REM ***** DATA *****
580 DATA "Stamp collection",5200,"China hutch",4300
590 DATA "Sterling silverware",2450,"Crystal chandelier",2400
600 DATA "Coin collection",6000,"China",3200
610 DATA "Packard coupe",6200,"Diamond solitaire",2400
620 DATA "Sail boat",2200,"Gold statuette",8200
630 DATA "Overstuffed chairs",2100,"Pearl necklace",3300
640 DATA "Bedroom suite",5300,"Diamond pendant",3300
650 DATA "Grand piano",3400,"Oboe",1200,"Spinnet",2500
660 DATA "Movie camera",2400,"Station wagon",4250
670 DATA "Convertible",3800,"Dining room suite",3850
680 DATA "Living room suite",3400,"Motocycle",2200
690 DATA "Jade figurine",2350,"Painting",3450
9999 END

```

This distribution has 18 of the 25 items.
 Selected number of scrambles = 25
 Smallest difference is \$ 2850 in try 1
 Smallest difference is \$ 650 in try 2
 Smallest difference is \$ 150 in try 4

Estate1:
 Sterling silverware \$2450.00
 China hutch \$4300.00
 Pearl necklace \$3300.00
 Movie camera \$2400.00
 Gold statuette \$8200.00
 Packard coupe \$6200.00
 Oboe \$1200.00
 Crystal chandelier \$2400.00
 Spinet \$2500.00
 =====
 Total \$32950.00

Estate2:
 Diamond solitaire \$2400.00
 Sail boat \$2200.00
 Stamp collection \$5200.00
 Overstuffed chairs \$2100.00
 Bedroom suite \$5300.00
 Coin collection \$6000.00
 China \$3200.00
 Diamond pendant \$3300.00
 Grand piano \$3400.00
 =====
 Total \$33100.00

Random Text

Another way to use random printers is to generate text. The two programs that follow have appeared in literature before. The first, BLIP, was written by one of us (JDR). It appeared in the August 1979 issue of *Creative Computing* in the article "Blip is the Blap of Bleep." The second program, SIMP, has appeared in various journals in several forms, but most popularly as the program "SIMP" in the form we present here.

Program BLIP generates aphorisms, or time worn saws, cliches, and philosophical truisms of the form "----- is the ----- of -----", for example "Obesity is the result of gluttony". But perhaps a slight modification like "Obesity is the result of success" or "Obesity is the brother of success" would produce equally palatable phrases of food for thought. The list of such aphorisms is endless, and the underlying "truth" is up to the interpretation of the reader.

```
10 REM filename: "blip"
20 REM purpose: Aphorism generator
30 REM author: jdr & jpg 9/82
40 REM Reference: Creative Computing, Aug. 1979, p. 116
50 REM -----
60 RANDOMIZE :: CALL CLEAR
70 REM The point is to make "Blip is the blap of bleep."
80 REM X#=Blip Y#=Blap z#=Bleep.
90 DIM X$(25),Y$(25),Z$(25)
100 FOR I=1 TO 25 :: READ X$(I),Y$(I),Z$(I):: NEXT I
110 INPUT "How many aphorisms would you like?":N
120 FOR I=1 TO N
```

```

130 J=INT(RND*25+1):: K=(RND*25+1):: L=INT(RND*25+1)
140 PRINT X$(J);" is the ";Y$(K);" of ";Z$(L)
150 NEXT I
160 REM ***** DATA *****
170 DATA Sanity,anathema,love.,Agony,quagmire,ardor.
180 DATA Distrust,blight,beauty.,Politics,harlot,hate.
190 DATA Greed,friend,filth.,Gluttony,mother,evil.
200 DATA Health,father,excess.,Hysteria,sister,fantasy.
210 DATA Brevity,brother,racism.,Charity,enemy,laughter.
220 DATA Security,bandit,naivete.,Science,house,apoplexy.
230 DATA Celibacy,kidney,sadism.,Decency,harpy,lust.
240 DATA Abuse,laxative,sex.,Weakness,follicle,leprosy.
250 DATA Death,heart,suicide.,Bravery,reward,deceit.
260 DATA Genius,genesis,loyalty.,Alcohol,liver,success..
270 DATA Violence,curator,ideocy.,Apathy,eunuch,progress.
280 DATA Money,result,failure.,Garbage,apex,release.
290 DATA Obesity,bladder,avrice.
9999 END

```

```

Agony is the kidney of failure.
Distrust is the house of suicide.
Science is the brother of suicide.
Money is the heart of sadism.
Apathy is the father of racism.
Apathy is the result of ideocy.
Charity is the eunuch of release.
Decency is the brother of success..
Brevity is the harpy of laughter.
Politics is the blight of filth.
Agony is the liver of love.
Security is the apex of ardor.
Decency is the anathema of evil.
Death is the eunuch of beauty.
Celibacy is the follicle of ideocy.
Abuse is the harpy of success..
Alcohol is the sister of fantasy.
Sanity is the bladder of release.
Weakness is the bladder of failure.
Decency is the sister of racism.
Charity is the sister of laughter.
Security is the bandit of lust.
Death is the curator of loyalty.
Alcohol is the father of leprosy.
Genius is the anathema of ardor.
Security is the friend of excess.
Distrust is the harlot of lust.
Death is the bandit of progress.
Distrust is the genesis of laughter.
Sanity is the eunuch of suicide.
Brevity is the result of beauty.
Death is the enemy of progress.
Gluttony is the quagmire of beauty.
Sanity is the quagmire of suicide.
Death is the bladder of avrice.
Health is the follicle of suicide.
Apathy is the eunuch of progress.
Bravery is the eunuch of deceit.
Hysteria is the result of ardor.
Agony is the result of racism.
Hysteria is the bladder of progress.
Distrust is the curator of filth.
Weakness is the brother of apoplexy.
Agony is the kidney of apoplexy.
Science is the apex of release.

```

Program SIMP is an anonymous contribution to the field of computing. Its purpose, not altogether serious, is to have the computer produce text material that resembles the technical jargon so often found in systems analysis reports and in progress reports. The fascination with this program lies in its highly readable output. It looks and sounds so good that one is tempted to insert several of its paragraphs in some report to see if it could be detected as computer-generated text. Read the output of this program aloud and you will be hard pressed to distinguish it from the real thing.

```

10 REM filename: "simp"
20 REM purpose: To generate gobbledigook
30 REM author: jpg & jdr (after other like efforts) 9/82
40 REM -----
50 RANDOMIZE :: CALL CLEAR
60 DIM X$(40)
70 FOR I=1 TO 40 :: READ X*(I):: NEXT I
80 INPUT "How many paragraphs":P
90 INPUT "How many sentences per paragraph":S
100 CALL CLEAR
110 FOR I=1 TO P
120 PRINT " ";
130 FOR J=1 TO S
140 A=INT(RND*10+1):: B=INT(RND*10+11):: C=INT(RND*10+21):: D=INT(RND*10+31)
150 PRINT X$(A);X$(B);X$(C);X$(D);" "
160 NEXT J :: PRINT
170 NEXT I
500 REM *****
510 REM data statements
520 DATA "In particular, ","On the other hand, "
530 DATA "However, ","Similarly, ","In this regard, "
540 DATA "As a resultant implication, ","For example, "
550 DATA "Based on integral subsystem considerations, "
560 DATA "Thus, ","With respect to specific goals, "
570 REM *****
580 DATA "a large portion of the interface coordination communication "
590 DATA "a constant flow of effective information "
600 DATA "the characterization of specific criteria "
610 DATA "initialization of critical subsystem development "
620 DATA "the product configuration baseline "
630 DATA "the fully intergrated test program "
640 DATA "any associated supporting element "
650 DATA "the incorporation of additional mission constraints "
660 DATA "the independent functional principle "
670 DATA "a primary interrelationship between system and/or subsystem technollog
ies "
680 REM *****
690 DATA "must utilize and be functionally interwoven with "
700 DATA "maximizes the probability of project success and minimizes the cost an
d time required for "
710 DATA "adds explicit performance limits to "
720 DATA "necessitates that urgent consideration be applied to "
730 DATA "requires considerable systems analysis and trade-off studies to arrive
at "
740 DATA "is further compounded, when taking into account "
750 DATA "presents extremely interesting challenges to "
760 DATA "recognizes the importance of other systems and the necessity for "
770 DATA "effects a significant implementation to "
780 DATA "adds overriding performance constraints to "
790 REM *****

```

```

800 DATA "the sophisticated hardware."
810 DATA "the anticipated fifth generation equipment."
820 DATA "the subsystem compatibility testing environment."
830 DATA "the structural design, based on system engineering concepts."
840 DATA "the preliminary qualification limits."
850 DATA "the philosophy of commonality and standardization."
860 DATA "the the evolution of specifications over a given time period."
870 DATA "the greater flight-worthiness concept."
880 DATA "any discrete configuration mode."
890 DATA "the total system rational."
999 END

```

Based on integral subsystem considerations, a constant flow of effective information must utilize and be functionally interwoven with any discrete configuration mode. However, initialization of critical subsystem development recognizes the importance of other systems and the necessity for the greater flight-worthiness concept.

On the other hand, the fully intergrated test program is further compounded, when taking into account the total system rational. In this regard, the incorporation of additional mission constraints effects a significant implementation to the structural design, based on system engineering concepts.

Random Message Selection

The next program, ACDC, plays a typical game of Acey-ducey. It selects a random message from the DATA statements in a slightly different fashion from those exemplified in the previous programs. Here the program reads the messages in the DATA statements up to a randomly selected position, then prints that element. The advantage is that the program does not have a DIM statement to make room for the array of messages. In some programs with a lot of messages, the savings in memory used is considerable, making this alternative technique worthwhile.

Acey-ducey is a popular game of cards played between dealer (computer) and player. This game is very generous: The player starts with \$100, the ante is only \$2, and there is no bet limit, except that it must be less than the total amount in the player's possession. The point of the game is to draw a card between the two cards drawn by the computer for the dealer. A drawn card outside of the dealer's range, or a card equal to either of the two dealer cards, constitutes a loss. All cards "in between" (another name for this game) constitute wins.

This version of the game is quite elementary. It can be embellished nicely with better graphics. We leave that up to you.

```

10 REM filename: "acdc"
20 REM purpose: To play the game "Acey-Ducey" (AC-DC)
30 REM author: jpg & jdr 10/82
40 !-----
50 RANDOMIZE :: CALL CLEAR
60 DIM C2(52)
70 FOR I=1 TO 4
80 FOR J=1 TO 13 :: C2(J+(I-1)*13)=J+1 :: NEXT J
90 NEXT I

```

```

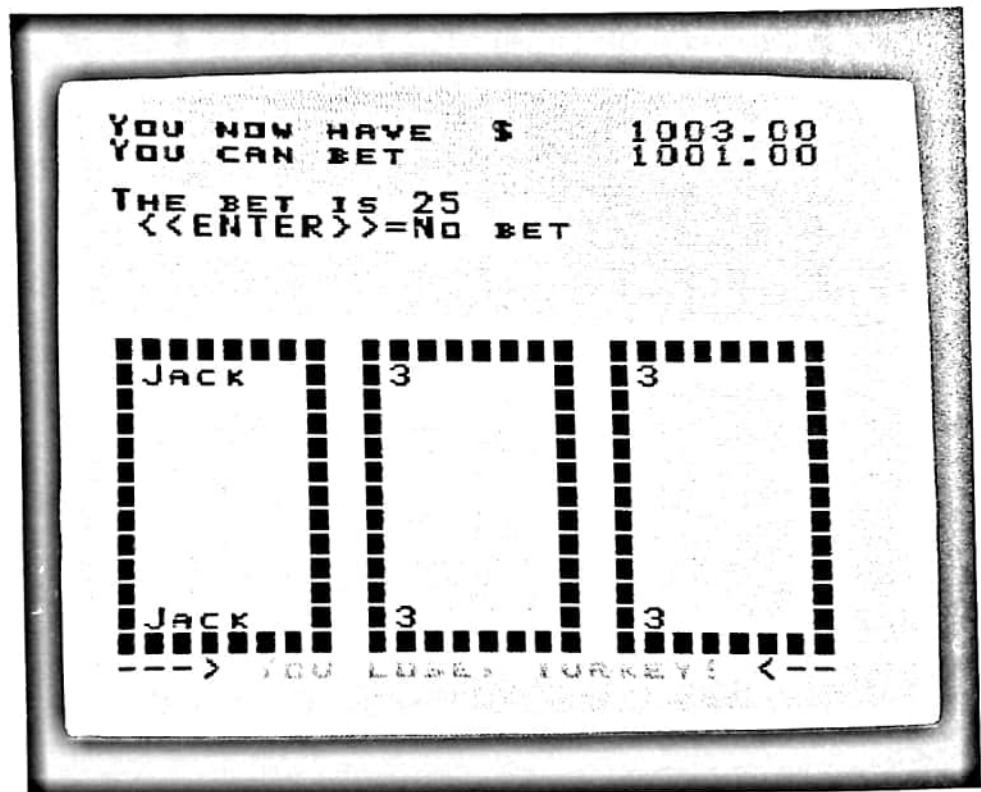
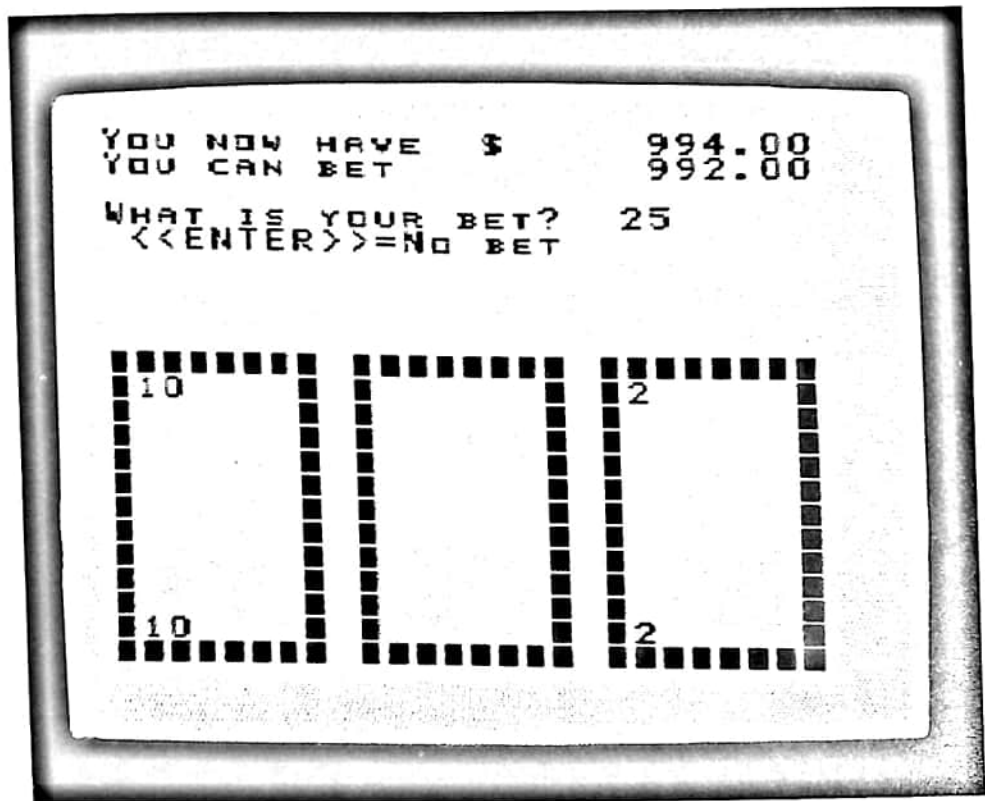
100 X$=" "
110 INPUT "How much do you have to lose":Q
120 PRINT "ante is $2 for all plays."
130 PRINT "All plays in dollars (no loose change, please)"
140 PRINT "(Patience while the cards are being shuffled."
150 GOSUB 5000
160 PRINT "The cards have been thoroughly shuffled."
170 INPUT "<<ENTER>>":A$ :: CALL CLEAR :: GOSUB 3000
180 DISPLAY AT(1,1):"You now have ";
190 DISPLAY AT(1,15):USING "$#####.##":Q
210 DISPLAY AT(4,1):RPT$(" ",28)
220 DISPLAY AT(3,6)BEEP:"Play? 1='yes' 2='no' (Ante is $2)";
230 CALL KEY(0,A,S):: IF S=0 THEN 230
240 IF CHR$(A)<>"2" THEN 280
250 DISPLAY AT(4,1):"You walk away with";Q;"dollars. "
260 FOR I=1 TO 500 :: NEXT I :: GOTO 9999
280 DISPLAY AT(23,1):RPT$(" ",28);
290 DISPLAY AT(2,1):"You can bet";
300 DISPLAY AT(2,15):USING "#####.##":Q-2
310 ! Clear cards
320 GOSUB 4000
330 Q=Q-2 :: IF Q<0 THEN 5000
340 GOSUB 1000 :: A=X :: DISPLAY AT(11,2):C$:: DISPLAY AT(21,2):C$;
350 GOSUB 1000 :: B=X :: DISPLAY AT(11,20):C$:: DISPLAY AT(21,20):C$;
360 IF A>B THEN T=A :: A=B :: B=T
370 DISPLAY AT(4,1)BEEP:"What is your bet?" :: DISPLAY AT(5,1):" <<ENTER>>=No be
t"
380 CALL KEY(0,K,S):: IF S=0 THEN 380
390 IF K=13 THEN DISPLAY AT(23,1):"Chicken!!";:: GOTO 180
400 M=VAL(CHR$(K)):: DISPLAY AT(4,19):M
405 CALL KEY(0,K,S):: IF S=0 THEN 405
406 IF K=13 THEN 408
407 M=VAL(CHR$(K))+10*M :: DISPLAY AT(4,19):M :: GOTO 405
408 DISPLAY AT(4,1):"The bet is";M
410 IF M<=Q THEN 440
420 DISPLAY AT(4,1):"You have only $";Q;"left to bet."
430 FOR I=1 TO 500 :: NEXT I :: GOTO 370
440 GOSUB 1000 :: DISPLAY AT(11,11):C$:: DISPLAY AT(21,11):C$;
450 IF X<=A OR X>=B THEN 470
460 Q=Q+M+2 :: Z=1 :: GOSUB 2000 :: GOTO 180
470 Z=2 :: GOSUB 2000
480 IF M<Q THEN Q=Q-M :: GOTO 180
490 DISPLAY AT(23,1):" "
500 DISPLAY AT(22,1):"Sorry but you blew your wad."
510 DISPLAY AT(23,1):"Try again? (yes=1 or no=0)"
520 CALL KEY(0,K,S):: IF S=0 THEN 520 ELSE IF K=49 THEN 50
530 DISPLAY AT(23,1):"O.K. Don't go away mad."
540 FOR I=1 TO 500 :: NEXT I :: CALL CLEAR :: STOP
1000 !***** Subroutine to display messages.*****
1010 N2=N2+1 :: IF N2<=52 THEN 1040
1020 DISPLAY AT(23,1):"(Shuffling)" :: GOSUB 5000
1030 GOTO 1010
1040 X=C2(N2)
1050 IF X<11 THEN C$=SEG$(STR$(X)&" ",1,5):: RETURN
1060 ON X-10 GOTO 1070,1080,1090,1100
1070 C$="Jack " :: RETURN
1080 C$="Queen" :: RETURN
1090 C$="King " :: RETURN
1100 C$="Ace " :: RETURN
2000 !***** Subroutine to display messages.*****
2010 R=INT(RND*10+1):: IF Z=2 THEN R=R+10
2020 FOR I=1 TO R :: READ A$ :: NEXT I
2030 DISPLAY AT(23,1):" "
2040 L=14-LEN(A$)/2
2050 FOR E=1 TO L-2
2060 DISPLAY AT(23,E):"-";
2070 DISPLAY AT(23,27-E):"-";
2080 NEXT E

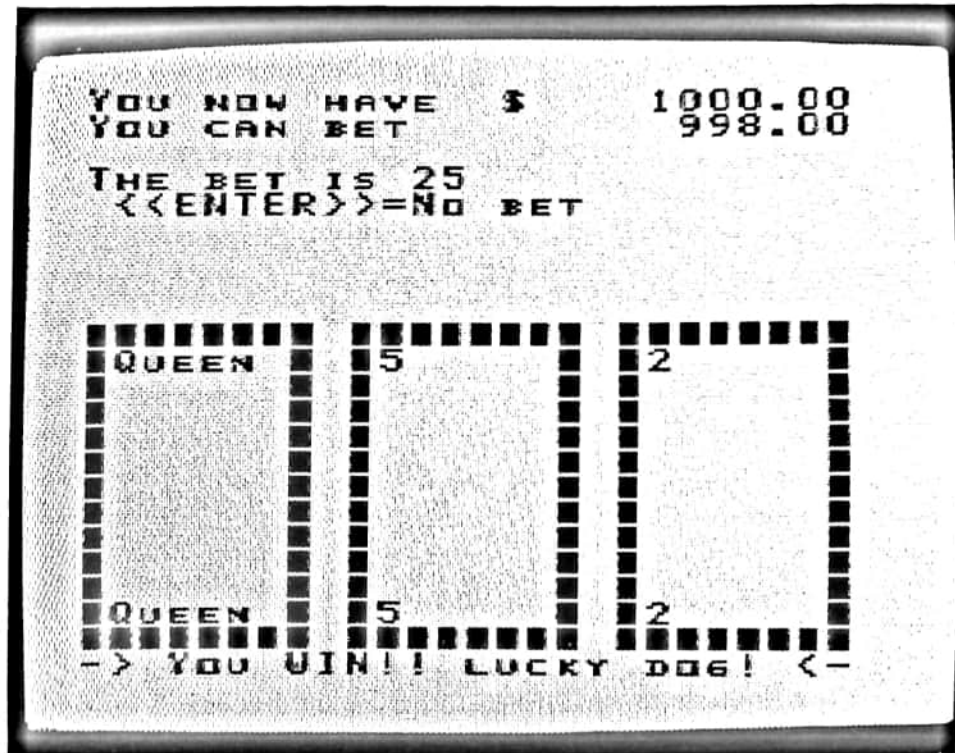
```

```

2090 DISPLAY AT(23,L-2):">";
2100 DISPLAY AT(23,27-L+2):"<";
2110 E1=INT(RND*5+1)
2120 FOR F=1 TO E1
2130 DISPLAY AT(23,L):A$;
2140 FOR E=1 TO 20 :: NEXT E
2150 DISPLAY AT(23,L):RPT$(" ",LEN(A$));
2160 FOR E=1 TO 10 :: NEXT E
2170 NEXT F
2180 DISPLAY AT(23,L):A$;
2190 RESTORE
2200 DISPLAY AT(4,22):"    ";
2210 RETURN
3000 !*****Subroutine to draw card outlines.*****
3010 CALL CLEAR
3020 FOR R=10 TO 22 STEP 12
3030 FOR C=3 TO 21 STEP 9
3040 CALL HCHAR(R,C,30,B)
3050 NEXT C :: C=3
3060 NEXT R
3070 R=10
3080 FOR J=0 TO 7 STEP 7
3090 FOR C=3 TO 24 STEP 9
3100 CALL VCHAR(R,C+J,30,12)
3110 NEXT C
3120 NEXT J
3130 RETURN
4000 !***** Subroutine to clear cards.*****
4010 FOR R=11 TO 21 STEP 10
4020 FOR J=0 TO 2 STEP 2
4030 FOR C=2 TO 20 STEP 9
4040 DISPLAY AT(R,C+J):X$;
4050 NEXT C
4060 NEXT J
4070 NEXT R
4080 RETURN
5000 !***** Card shuffling routine.*****
5010 FOR I=1 TO 100 :: DISPLAY AT(23,20):100-I;
5020 I1=INT(RND*52+1):: I2=INT(RND*52+1)
5030 T=C2(I1):: C2(I1)=C2(I2):: C2(I2)=T
5040 NEXT I
5050 N2=0
5060 RETURN
7000 !***** m e s s a g e s *****
7010 DATA "You WIN!! lucky dog!"
7020 DATA "This can't last forever!"
7030 DATA "You by lunch, lucky!"
7040 DATA "Say! Let's do it again!"
7050 DATA "You won, you sly dog!"
7060 DATA "You'll get yours--later!"
7070 DATA "It's yours, yours!"
7080 DATA "You won again!?!?!?"
7090 DATA "Remember, it's taxable!"
7100 DATA "##%&*(##$#@ - You won!"
7110 DATA "Gotcha!!","Gotcha again!!"
7120 DATA "You thought you were smart!"
7130 DATA "YOU LOSE -- big spender."
7140 DATA "You lose, turkey!"
7150 DATA "(Chuckle) House wins again."
7160 DATA "How astonishing!! You lose!"
7170 DATA "You lose again!"
7180 DATA "Stick to dominos!!"
7190 DATA "A dead dodo could do better."
9999 END

```





Normal Distribution of Values

We end this discussion on the selection of pointers with a program that simulates the production of a normal distribution. Consider the R positions of the array X to be the range of values that a normally distributed variate can assume. For example, if R is 100, you could consider the possible test scores from 1 to 100, or from 901 to 1000. Next, consider the standard deviation S of the variates within that range, and M the arithmetic mean of the scores. Last of all, the number of scores, for example the number of students who took the test, is N.

Program BELLCURV declares the array X to be dimensioned R, then fills the array with values that simulate the frequency of N observations. The illustration below shows visually what the program does when the user has indicated a range R of 10, starting at 41, with a standard deviation of 2, a sample size N of 100, and a mean of 45.

Observations	3	7	15	19	20	19	10	5	2	0
Score value	41	42	43	44	45	46	47	48	49	50
X subscript	1	2	3	4	5	6	7	8	9	10

The X array gets its frequency counts F by calculating the values that should be there on the basis of what the user has supplied as R, M, S, and N, and the following algorithm:

1. K is the sum of 12 random values between 0 and 1.

$$K = \text{RND} + \text{RND} + \dots + \text{RND}$$
2. P is defined using K

$$P = M + S * (K - 6)$$

The returned value P is a normally distributed random variate with a mean of M and a standard deviation of S.

Suppose you want to simulate a normal distribution of 300 scores between 0 and 100, with a mean of 70 and a standard deviation of 10. You can set up a loop that produces 300 values of P using R = 101 (0 to 100 inclusive), S = 10, and M = 70. Each P that is produced can act as a subscript (finally, back to the subject of pointers) to the X array, and tally the frequency of observations with the statement

$$X(P) = X(P) + 1$$

The variate P must be checked to make certain that it is within the DIMensioned range between the low L and the high H with the statement:

IF P < L OR P > H THEN (get new P)

Once the X array has been filled, it is easy to produce a graph of the bell-shaped normal curve that these simulated frequency distributions would generate. We refer you to our Graphics book in this series for the various graphing techniques you could use to display the frequencies in a more suitable fashion than does the program BELLCURV.

```
10 REM filename: "bellcurv"
20 REM purposr: simulation of normal distribution
30 REM author: jpg & jdr 9/82
40 REM -----
50 RANDOMIZE :: CALL CLEAR
60 DIM X(100)
65 OPEN #1:"RS232"
70 PRINT "Normal Distribution of Random Variates"
80 INPUT "What range of scores (low, high)":L,H
90 R=H-L !range R is high H minus low L
100 FOR I=0 TO 100 :: X(I)=0 :: NEXT I
110 INPUT "What is the mean of the observations":M
120 IF M<L OR M>H THEN PRINT "out of range" :: GOTO 110
130 INPUT "What is the std. dev. of observations":S
140 IF S>R THEN PRINT "too large" :: GOTO 130
150 INPUT "How many observations are to be generated":N
160 IF N>1000 THEN PRINT "too many" :: GOTO 150
170 FOR I=1 TO N !Generate normally distributed variate
180 K=0
190 FOR J=1 TO 12 :: K=K+RND :: NEXT J
200 P=M+S*(K-6)
210 IF P<L OR P>H THEN 180
220 X(P)=X(P)+1 :: PRINT I;!let the user know what's happening
230 NEXT I :: PRINT
240 GOSUB 1000 !Print vertical histogram
250 PRINT #1 :: PRINT #1
260 GOSUB 2000 !Print horizontal histogram
270 STOP
1000 REM ***** Vertical histogram *****
1010 B=-1 ! INITIALIZE MODE TO NEGATIVE
1020 F=0 :: S2=0 ! FLAG AND SUM
1030 PRINT #1:"Mean of";N;"observations=";M;" , std. dev.=";S
```

```

1040 FOR I=L TO H
1050 IF X(I)=0 AND F=0 THEN 1120 ! DON'T GRAPH
1060 PRINT #1:I;X(I);TAB(12);
1070 IF X(I)>B THEN B=X(I)! get mode B
1080 FOR R=1 TO X(I):: PRINT #1:"*";:: NEXT R :: PRINT #1:
1090 F=1 ! Set flag for first nonzero X
1100 S2=S2+X(I)
1110 IF S2=N THEN 1130 ! IF ALL PRINTED, SKIP
1120 NEXT I
1130 RETURN
2000 REM ** Subroutine to print histogram horizontally**
2010 P=0 :: Q=0 ! Locate first and last nonzero X
2020 FOR I=0 TO 100
2030 IF X(I)<>0 AND P=0 THEN P=I
2040 IF X(100-I)<>0 AND Q=0 THEN Q=100-I
2050 NEXT I
2070 PRINT #1:"Mean of";N;"observations=";M;"", std. dev=";S
2080 FOR I=B TO 1 STEP -1
2085 PRINT #1:I;TAB(12);
2090 FOR J=P TO Q
2100 IF X(J)>=I THEN PRINT #1:"*";ELSE PRINT #1:" ";
2110 NEXT J
2120 PRINT #1
2130 NEXT I
2140 RETURN
9999 END

```

Mean of 250 observations= 75 , std. dev.= 5

```

63 1      *
64 1      *
65 3      ***
66 5      *****
67 2      **
68 10     *****
69 8      *****
70 16     *****
71 19     *****
72 13     *****
73 24     *****
74 18     *****
75 24     *****
76 23     *****
77 14     *****
78 22     *****
79 10     *****
80 10     *****
81 6      *****
82 7      *****
83 5      *****
84 2      **
85 3      ***
86 1      *
87 1      *
88 0
89 1      *
90 0
91 0
92 1      *

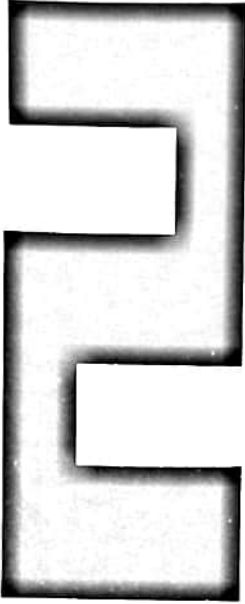
```

```

Mean of 250 observations= 75 , std. dev= 5
24      * *
23      * **
22      * ** *
21      * ** *
20      * ** *
19      * * ** *
18      * **** *
17      * **** *
16      ** **** *
15      ** **** *
14      ** **** *
13      ** **** *
12      ** **** *
11      ** **** *
10      * **** *
9       * **** *
8       * **** *
7       * **** *
6       * **** *
5       * **** *
4       * **** *
3       * **** *
2       * **** *
1       * **** *

```

This first chapter has reviewed some commonly used methods of subscript management. These techniques deserve special attention because of their power in simulation programs in which events occur according to the rules of chance. In the next chapter we will discuss how pointers can be used to specify single elements of arrays or records in various sort programs.



Sorting

For some unknown reason, the topic of sorts seems to hold a particular fascination for most programmers. Perhaps it is because sorts produce order out of chaos, or perhaps programmers like sorts because of their inherent comparability, that is, the ease with which their *effectiveness* (do they work?) and their *efficiency* (how fast do they work?) can be observed and measured. Whatever the reason for a programmer's interest in the topic, sorts also have a very important place in data management. As a future chapter will indicate, one cannot access a file or array using a binary search unless that data structure is in sorted order.

There exist dozens of sorting algorithms; some are bad and some are good. Their efficiency can be measured with two quantities: (1) The speed of the sort — that is, how many seconds does it take to sort a given number of values? (2) The size of the sort — that is, how many bytes does the BASIC code take up in memory? We have had opportunity to test a wide variety of sorts, and we have formulated our own simple scheme for categorizing them.

Sorting Categories

A sorting algorithm belongs to one of these four categories:

- *Brute force sorts* copy one unsorted list into a second sorted list. Program BRFRSORT is an example of such a sort.
- *Exchange sorts* rearrange the elements of a list in place so that no wasted memory space is used to hold that second array. Programs INRSORT (insertion), BBSORT (bubble),

EXCHSORT (simple exchange), and DELXSORT (delayed exchange) are all examples of exchange sorts.

- *Binary sorts* are significantly faster than either the brute force or exchange sorts. These algorithms rely on a logical restructuring of the data into smaller groups of elements before resorting to in-place exchanges. Binary sorts can be subdivided into two categories: (a) Binary sorts that use no additional memory space for stacks; programs SHELSORT (Shell), SMETSORT (Shell-Metzner), and HEAPSORT (Heap) are all in this category; (b) Binary sorts that use stacks. The best example of this type of sort is the Quicksort (one word), which we include in this chapter as program QUIKSORT.
- *Tree sorts* are the fastest of all, but they suffer a major disadvantage of requiring two separate arrays for links as well as the array of values to be sorted. Program TREESORT shows such a tree sort based on the AVL tree structure, also known as the B-tree or binary sequence search tree (BSST) structure. Chapter 7 of this book will discuss this data structure in more detail.

Some sorting techniques such as the radix sort are not included here. We will discuss others, such as the detached key sort and the sort-merge, in this and subsequent chapters. The radix sort, though once popular, is more suitable for discussion in conjunction with punch-card-oriented systems, of which the TI-99/4A is the antithesis. For those of you who wish to pursue the topic of sorts, we encourage you to beg, borrow, steal, or purchase a copy of Knuth's book referenced at the end of this chapter.

Brute Force Sorts

Brute force sorting is exemplified here by the program BRFRSORT. The array D is loaded with ten random 2-digit integers between 1 and 50. The program displays the contents of both the source array D and the destination array D1 at each of the ten successive scans, or passes, through the array. After ten passes, the array D1 contains all of the original elements of D, but in sorted order.

```

10 REM filename: "brfrsort"
20 REM purpose: Brute force sort
30 REM author: jpg & jdr 9/82
40 REM -----
50 DIM D(100),D1(100)
60 RANDOMIZE :: CALL CLEAR
65 OPEN #1:"RS232"
70 REM Place random integers between 1 and 50 into D.
80 FOR I=1 TO 10 :: D(I)=INT(RND*50+1):: NEXT I
90 REM Print table heading.
100 PRINT #1:"Pass # Position ->";
110 FOR I=1 TO 10 :: PRINT #1:TAB(18+I*4);I;:: NEXT I :: PRINT #1
120 P=0 :: N=10 :: GOSUB 2000
130 GOSUB 1000 !Perform sort.
140 GOTO 9999
1000 REM***** Brute force sort. *****
1010 FOR P=1 TO N
1020 S=99 !Set S to smallest seen so far.
1030 REM Scan D, copy smallest into Pth. position of D1.
1040 FOR J=1 TO N
1050 IF D(J)<S THEN S=D(J):: X=J
1060 NEXT J :: D1(P)=D(X):: D(X)=99
1070 GOSUB 2000 !Print out this pass.
1080 NEXT P
1090 RETURN
2000 REM***** Print result of this pass *****
2010 PRINT #1:P;TAB(7);"D";
2020 FOR I=1 TO N :: PRINT #1:TAB(18+I*4);D(I);:: NEXT I
2030 PRINT #1 :: PRINT #1:TAB(7);"D1";
2040 FOR I=1 TO N :: PRINT #1:TAB(18+I*4);D1(I);:: NEXT I
2050 PRINT #1 :: RETURN
9999 END

```

Pass #	D	D1	1	2	3	4	5	6	7	8	9	10
0	D		23	2	6	25	42	22	20	28	15	47
	D1		0	0	0	0	0	0	0	0	0	0
1	D		23	99	6	25	42	22	20	28	15	47
	D1		2	0	0	0	0	0	0	0	0	0
2	D		23	99	99	25	42	22	20	28	15	47
	D1		2	6	0	0	0	0	0	0	0	0
3	D		23	99	99	25	42	22	20	28	99	47
	D1		2	6	15	0	0	0	0	0	0	0
4	D		23	99	99	25	42	22	99	28	99	47
	D1		2	6	15	20	0	0	0	0	0	0
5	D		23	99	99	25	42	99	99	28	99	47
	D1		2	6	15	20	22	0	0	0	0	0
6	D		99	99	99	25	42	99	99	28	99	47
	D1		2	6	15	20	22	23	0	0	0	0
7	D		99	99	99	99	42	99	99	28	99	47
	D1		2	6	15	20	22	23	25	0	0	0
8	D		99	99	99	99	42	99	99	99	99	47
	D1		2	6	15	20	22	23	25	28	0	0
9	D		99	99	99	99	99	99	99	99	99	47
	D1		2	6	15	20	22	23	25	28	42	0
10	D		99	99	99	99	99	99	99	99	99	99
	D1		2	6	15	20	22	23	25	28	42	47

Exchange Sorts

Program INRSORT shows the first type of exchange sort, the insertion sort. This time, the program does not need any additional memory space, as it does its sorting in place. The output of the program shows the way the sort progresses. Notice that each iteration of the outer loop (the one with the P index) finds the first element of the array that is not in sorted order. Then the inner loop (the one with the J index) proceeds to squeeze that element into its proper place. This is done by shifting all higher-valued elements down one position until the vacancy that is left is the proper place for the "out of sorts" element.

```

10 REM filename: "insrsort"
20 REM purpose: Insertion sort
30 REM author: jpg & jdr 9/82
40 REM -----
50 DIM D(100)
60 RANDOMIZE :: CALL CLEAR
65 OPEN #1:"RS232"
70 REM Place random integers between 1 and 50 into D.
80 FOR I=1 TO 10 :: D(I)=INT(RND*50+1):: NEXT I
90 REM Print table heading.
100 PRINT #1:"Pass # Position ->";
110 FOR I=1 TO 10 :: PRINT #1:TAB(18+I*4);I;:: NEXT I :: PRINT #1
120 P=0 :: N=10 :: GOSUB 2000
130 GOSUB 1000 !Perform sort.
140 GOTO 9999
1000 REM ***** Insertion Sort *****
1010 FOR P=1 TO N-1
1020 X=D(P+1)
1030 FOR J=P TO 1 STEP -1
1040 IF X>=D(J)THEN 1050
1045 D(J+1)=D(J):: NEXT J :: J=0
1050 D(J+1)=X :: GOSUB 2000
1060 NEXT P
1070 RETURN
2000 REM ***** Print the result of this pass ****
2010 PRINT #1:P;TAB(7);"D";
2020 FOR I=1 TO N :: PRINT #1:TAB(18+I*4);D(I);:: NEXT I
2030 PRINT #1 :: RETURN
9999 CLOSE #1 :: END

```

Pass #	D	Position ->	1	2	3	4	5	6	7	8	9	10
0	D		27	12	22	20	35	31	5	43	1	38
1	D		12	27	22	20	35	31	5	43	1	38
2	D		12	22	27	20	35	31	5	43	1	38
3	D		12	20	22	27	35	31	5	43	1	38
4	D		12	20	22	27	35	31	5	43	1	38
5	D		12	20	22	27	31	35	5	43	1	38
6	D		5	12	20	22	27	31	35	43	1	38
7	D		5	12	20	22	27	31	35	43	1	38
8	D		1	5	12	20	22	27	31	35	43	38
9	D		1	5	12	20	22	27	31	35	38	43

If you are interested in a more detailed discussion of this method of sorting, we refer you to the article by Albert Nijenhuis which is referenced at the end of this chapter.

The third program in this chapter, BBSORT, shows the *bubble sort*. It is indeed unfortunate that this sort is so popular, because it, like all exchange sorts, is dreadfully slow when compared to any one of the binary sorts. We include it here to familiarize you with it as a technique, not to recommend its use but rather to allow you to recognize it when you see it again.

The bubble sort has one overwhelming advantage: It is very fast when the elements you are sorting are already in almost sorted order. This is a feature of the insertion sort also. However, since this condition occurs rarely, you would still do better to choose a Shell or Quicksort.

```

10 REM filename: "bbsort"
20 REM purpose: Bubble sort
30 REM author: jpg & jdr 9/82
40 REM -----
50 DIM D(100)
60 RANDOMIZE :: CALL CLEAR
65 OPEN #1:"RS232"
70 REM Place random integers between 1 and 50 into D.
80 FOR I=1 TO 10 :: D(I)=INT(RND*50+1):: NEXT I
90 REM Print table heading.
100 PRINT #1:"Pass # Position ->";
110 FOR I=1 TO 10 :: PRINT #1:TAB(18+I*4);I;:: NEXT I :: PRINT #1
120 P=0 :: N=10 :: GOSUB 2000
130 GOSUB 1000 !Perform sort.
140 GOTO 9999
1000 REM ***** Bubble Sort *****
1010 FOR P=1 TO N-1
1020 F=0 !Set "swap" flag to 0 no swaps yet).
1030 FOR J=1 TO N-P
1040 IF D(J+1)<D(J) THEN T=D(J):: D(J)=D(J+1):: D(J+1)=T :: F=1 ! Flag a swap
1050 NEXT J :: GOSUB 2000
1060 IF F=0 THEN RETURN
1070 NEXT P
1080 RETURN
2000 REM ***** Print the result of this pass ****
2010 PRINT #1:P;TAB(7);"D";
2020 FOR I=1 TO N :: PRINT #1:TAB(18+I*4);D(I);:: NEXT I
2030 PRINT #1 :: RETURN
9999 CLOSE #1 :: END

```

Pass #	D	Position ->	1	2	3	4	5	6	7	8	9	10
0	D		39	42	43	10	20	16	49	32	7	10
1	D		39	42	10	20	16	43	32	7	10	49
2	D		39	10	20	16	42	32	7	10	43	49
3	D		10	20	16	39	32	7	10	42	43	49
4	D		10	16	20	32	7	10	39	42	43	49
5	D		10	16	20	7	10	32	39	42	43	49
6	D		10	16	7	10	20	32	39	42	43	49
7	D		10	7	10	16	20	32	39	42	43	49
8	D		7	10	10	16	20	32	39	42	43	49
9	D		7	10	10	16	20	32	39	42	43	49

Before we go to the next type of sort, we will refer you to yet another source, this one a book by Thomas Dwyer and Margot Critchfield (see the end-of-chapter reference list) for a detailed discussion of the bubble sort.

Program EXCHSORT shows a sort that is commonly known by two names: *Exchange sort* or *Selection sort*. Its popularity rests on the fact that of all sorts it is probably the easiest to code.

```

10 REM filename: "exchsort"
20 REM purpose: Exchange sort
30 REM author: jpg & jdr 9/82
40 REM -----
50 DIM D(100)
60 RANDOMIZE :: CALL CLEAR
65 OPEN #1:"RS232"
70 REM Place random integers between 1 and 50 into D.
80 FOR I=1 TO 10 :: D(I)=INT(RND*50+1):: NEXT I
90 REM Print table heading.
100 PRINT #1:"Pass # Position ->";
110 FOR I=1 TO 10 :: PRINT #1:TAB(18+I*4);I;:: NEXT I :: PRINT #1
120 P=0 :: N=10 :: GOSUB 2000
130 GOSUB 1000 !Perform sort.
140 GOTO 9999
1000 REM ***** Exchange (or selection sort) *****
1010 FOR P=1 TO N-1
1020 FOR J=P+1 TO N
1030 IF D(P)>D(J) THEN T=D(P):: D(P)=D(J):: D(J)=T
1040 NEXT J :: GOSUB 2000
1050 NEXT P
1060 RETURN
2000 REM ***** Print the result of this pass *****
2010 PRINT #1:P;TAB(7);"D";
2020 FOR I=1 TO N :: PRINT #1:TAB(18+I*4);D(I);:: NEXT I
2030 PRINT #1 :: RETURN
9999 CLOSE #1 :: END

```

Pass #	Position ->	1	2	3	4	5	6	7	8	9	10
0	D	47	28	31	31	34	24	43	38	5	10
1	D	5	47	31	31	34	28	43	38	24	10
2	D	5	10	47	31	34	31	43	38	28	24
3	D	5	10	24	47	34	31	43	38	31	28
4	D	5	10	24	28	47	34	43	38	31	31
5	D	5	10	24	28	31	47	43	38	34	31
6	D	5	10	24	28	31	31	47	43	38	34
7	D	5	10	24	28	31	31	34	47	43	38
8	D	5	10	24	28	31	31	34	38	47	43
9	D	5	10	24	28	31	31	34	38	43	47

A close cousin to the exchange sort shown above is a distinct improvement, the *Delayed exchange sort* or the *Delayed selection sort*. With a little more coding, it marks the smallest value seen, and withholds the exchange (thus the name) until all values are checked.

```

10 REM filename: "delxsort"
20 REM purpose: Delayed exchange sort
30 REM author: jpg & jdr 9/82
40 REM -----
50 DIM D(100)
60 RANDOMIZE :: CALL CLEAR
65 OPEN #1:"RS232"
70 REM Place random integers between 1 and 50 into D.
80 FOR I=1 TO 10 :: D(I)=INT(RND*50+1):: NEXT I
90 REM Print table heading.
100 PRINT #1:"Pass # Position ->";
110 FOR I=1 TO 10 :: PRINT #1:TAB(18+I*4);I;:: NEXT I :: PRINT #1
120 P=0 :: N=10 :: GOSUB 2000
130 GOSUB 1000 !Perform sort.
140 GOTO 9999
1000 REM ***** Delayed exchange sort *****
1010 FOR P=1 TO N-1
1020 X=P
1030 FOR J=P+1 TO N
1040 IF D(J)<D(X) THEN X=J !Mark this position.
1050 NEXT J
1060 IF P<>X THEN T=D(X):: D(X)=D(P):: D(P)=T
1070 GOSUB 2000
1080 NEXT P
1090 RETURN
2000 REM ***** Print the result of this pass ****
2010 PRINT #1:P;TAB(7);"D";
2020 FOR I=1 TO N :: PRINT #1:TAB(18+I*4);D(I);:: NEXT I
2030 PRINT #1 :: RETURN
9999 CLOSE #1 :: END

```

Pass #	D	Position ->	1	2	3	4	5	6	7	8	9	10
0	D		3	37	42	41	9	8	38	5	13	37
1	D		3	37	42	41	9	8	38	5	13	37
2	D		3	5	42	41	9	8	38	37	13	37
3	D		3	5	8	41	9	42	38	37	13	37
4	D		3	5	8	9	41	42	38	37	13	37
5	D		3	5	8	9	13	42	38	37	41	37
6	D		3	5	8	9	13	37	38	42	41	37
7	D		3	5	8	9	13	37	37	42	41	38
8	D		3	5	8	9	13	37	37	38	41	42
9	D		3	5	8	9	13	37	37	38	41	42

Binary Sorts

The third category of sorts is the class of binary sorts. These are so much faster than any of the preceding that their use is almost universal in commercial sorting packages. Some programmers we have met try to justify the use of any of the exchange sorts by saying that they are just as fast with a low number of items. Unfortunately, that number is so low, between 8 and 20 depending on the sorts being compared, that the use of any exchange sort is hardly ever warranted.

Binary sorts have one principle in common. All of them subdivide the list that is to be sorted into sublists of various sizes and types. Once these sublists have been established, they are sorted using a simple exchange of elements. The reason binary sorts are faster is due to the

fact that it takes less time to sort two lists of 50 items, then merge those two lists, than to sort one list of 100 items. Each one of the sorts we show in this section uses this principle. We are primarily interested in presenting the code for these sorts rather than going into detail to show you how each one works. We refer you to the list of references at the end of the chapter if you are particularly interested in some of these techniques.

The first binary sort, SHELSORT, is the *Shell sort* named after its inventor, Donald Shell. A variation of the Shell sort is shown in program SMETSORT. The name *Shell-Metzner sort* was coined by one of us (JPG) in a magazine article in 1976. It has since gained a wide following as a fine, fast sort that is easy to code and uses no extraneous memory stacks or arrays.

```

10 REM filename: "shelsort"
20 REM purpose: Shell sort
30 REM author: jpg & jdr 9/82
40 REM -----
50 DIM D(100)
60 RANDOMIZE :: CALL CLEAR
65 OPEN #1:"RS232"
70 REM Place random integers between 1 and 50 into D.
80 FOR I=1 TO 10 :: D(I)=INT(RND*50+1):: NEXT I
90 REM Print table heading.
100 PRINT #1:"Pass # Position ->";
110 FOR I=1 TO 10 :: PRINT #1:TAB(18+I*4);I;:: NEXT I :: PRINT #1
120 P=0 :: N=10 :: GOSUB 2000
130 GOSUB 1000 !Perform sort.
140 GOTO 9999
1000 REM ***** Shell sort *****
1010 P=N
1020 IF P<=1 THEN RETURN
1030 P=INT(P/2):: M=N-P
1040 F=0
1050 FOR J=1 TO M
1060 X=J+P
1070 IF D(J)>D(X) THEN T=D(J):: D(J)=D(X):: D(X)=T :: F=1
1080 NEXT J
1090 IF F>0 THEN 1040
1100 GOSUB 2000 :: GOTO 1020
2000 REM ***** Print the result of this pass ****
2010 PRINT #1:P;TAB(7);"D";
2020 FOR I=1 TO N :: PRINT #1:TAB(18+I*4);D(I);:: NEXT I
2030 PRINT #1 :: RETURN
9999 CLOSE #1 :: END

```

Pass #	Position ->	1	2	3	4	5	6	7	8	9	10
0	D	25	16	4	32	25	32	5	23	35	21
5	D	25	5	4	32	21	32	16	23	35	25
2	D	4	5	16	23	21	25	25	32	35	32
1	D	4	5	16	21	23	25	25	32	32	35

```

10 REM filename: "smetsort"
20 REM purpose: Shell-Metzner sort
30 REM author: jpg & jdr 9/82
40 REM -----
50 DIM D(100)
60 RANDOMIZE :: CALL CLEAR
65 OPEN #1:"RS232"
70 REM Place random integers between 1 and 50 into D.
80 FOR I=1 TO 10 :: D(I)=INT(RND*50+1):: NEXT I
90 REM Print table heading.
100 PRINT #1:"Pass # Position ->";
110 FOR I=1 TO 10 :: PRINT #1:TAB(18+I*4);I;:: NEXT I :: PRINT #1
120 P=0 :: N=10 :: GOSUB 2000
130 GOSUB 1000 'Perform sort.
140 GOTO 9999
1000 REM ***** Shell-Metzner sort *****
1010 P=N
1020 P=INT(P/2)
1030 IF P=0 THEN RETURN
1040 K=N-P :: J=1
1050 I=J
1060 L=I+P
1070 IF D(I)<D(L) THEN 1100
1080 T=D(I):: D(I)=D(L):: D(L)=T :: I=I-P
1090 IF I>=1 THEN 1060
1100 J=J+1
1110 IF J<=K THEN 1050
1120 GOSUB 2000 :: GOTO 1020
2000 REM ***** Print the result of this pass ****
2010 PRINT #1:P;TAB(7);"D";
2020 FOR I=1 TO N :: PRINT #1:TAB(18+I*4);D(I);:: NEXT I
2030 PRINT #1 :: RETURN
9999 CLOSE #1 :: END

```

Pass #	Position ->	1	2	3	4	5	6	7	8	9	10
0	D	25	46	20	39	40	5	38	29	50	22
5	D	5	38	20	39	22	25	46	29	50	40
2	D	5	25	20	29	22	38	46	39	50	40
1	D	5	20	22	25	29	38	39	40	46	50

The next sort that we include in this category of binary sorts is the *Heap sort*. The principle on which it is based is again the subdivision of the list, this time into sublists called heaps, hence the name.

```

10 REM filename: "heapsort"
20 REM purpose: Heap sort
30 REM author: jpg & jdr 9/82
40 REM -----
50 DIM D(100)
60 RANDOMIZE :: CALL CLEAR
65 OPEN #1:"RS232"
70 REM Place random integers between 1 and 50 into D.
80 FOR I=1 TO 10 :: D(I)=INT(RND*50+1):: NEXT I
90 REM Print table heading.
100 PRINT #1:"Pass # Position ->";
110 FOR I=1 TO 10 :: PRINT #1:TAB(18+I*4);I;:: NEXT I :: PRINT #1
120 P=0 :: N=10 :: GOSUB 2000
130 GOSUB 1000 !Perform sort.
140 GOTO 9999
1000 REM ***** Heap Sort *****
1010 X=INT(N/2)+1 :: P=N :: GOSUB 2000
1020 IF X<>1 THEN X=X-1 :: A=D(X):: J=X :: GOTO 1060
1030 A=D(P):: D(P)=D(1):: P=P-1
1040 IF P<>1 THEN J=X ELSE D(1)=A :: GOSUB 2000 :: RETURN
1050 GOSUB 2000
1060 I=J :: J=J+J
1070 IF J=P THEN 1100
1080 IF J>P THEN D(I)=A :: GOTO 1020
1090 IF D(J)<D(J+1) THEN J=J+1
1100 IF A<D(J) THEN D(I)=D(J):: GOTO 1060 ELSE D(I)=A :: GOTO 1020
1110 J=J+1
1120 IF J<=K THEN 1060
1130 GOSUB 2000 :: GOTO 1020
2000 REM ***** Print the result of this pass ****
2010 PRINT #1:P;TAB(7);"D";
2020 FOR I=1 TO N :: PRINT #1:TAB(18+I*4);D(I);:: NEXT I
2030 PRINT #1 :: RETURN
9999 CLOSE #1 :: END

```

Pass #	D	Position ->	1	2	3	4	5	6	7	8	9	10
0	D		26	21	50	14	50	35	3	31	44	7
10	D		26	21	50	14	50	35	3	31	44	7
9	D		50	44	50	31	21	35	3	26	14	50
8	D		50	44	35	31	21	7	3	26	50	50
7	D		44	31	35	26	21	7	3	44	50	50
6	D		35	31	14	26	21	7	35	44	50	50
5	D		31	26	14	3	21	31	35	44	50	50
4	D		26	21	14	3	26	31	35	44	50	50
3	D		21	7	14	21	26	31	35	44	50	50
2	D		14	7	14	21	26	31	35	44	50	50
1	D		3	7	14	21	26	31	35	44	50	50

The last program in this group of sorts is somewhat different from the rest. Whereas the various Shell sorts and the heap sort required no additional memory space aside from the array to be sorted and the instructions, the *Quicksort* (one word, always capitalized) needs a stack. This data structure is a last-in, first-out list in which are placed array pointers, and as a general procedure for managing data will be discussed in Chapter 4. The memory overhead which the stack requires depends on the size of the array to be sorted. Programmers of the Quicksort usually dimension the stack array to a size between 60% and 80% of the array to be sorted.

```

10 REM filename: "quicksort"
20 REM purpose: Quicksort
30 REM author: jpg & jdr 9/82
40 REM -----
50 DIM D(100),STK(75)
60 RANDOMIZE :: CALL CLEAR
65 OPEN #1:"RS232"
70 REM Place random integers between 1 and 50 into D.
80 FOR I=1 TO 10 :: D(I)=INT(RND*50+1):: NEXT I
90 REM Print table heading.
100 PRINT #1:"Pass # Position ->";
110 FOR I=1 TO 10 :: PRINT #1:TAB(18+I*4);I;:: NEXT I :: PRINT #1
120 P=0 :: N=10 :: GOSUB 2000
130 GOSUB 1000 !Perform sort.
140 GOTO 9999
1000 REM ***** Quicksort *****
1010 X=0 :: I=X+X :: STK(I+1)=1 :: STK(I+2)=N :: X=X+1
1020 IF X=0 THEN RETURN
1030 X=X-1 :: I=X+X :: A=STK(I+1):: B=STK(I+2)
1040 Z=D(A):: TP=A :: BT=B+1
1050 BT=BT-1
1060 IF BT=TP THEN 1110
1070 IF Z<=D(BT)THEN 1050 ELSE D(TP)=D(BT)
1080 TP=TP+1
1090 IF BT=TP THEN 1110
1100 IF Z>D(TP)THEN 1080 ELSE D(BT)=D(TP):: GOTO 1050
1110 D(TP)=Z
1120 IF B-TP>=2 THEN I=X+X :: STK(I+1)=TP+1 :: X=X+1 :: STK(I+2)=B
1130 IF BT-A>=2 THEN I=X+X :: STK(I+1)=A :: X=X+1 :: STK(I+2)=BT-1
1140 P=P+1 :: GOSUB 2000 :: GOTO 1020
2000 REM ***** Print the result of this pass ****
2010 PRINT #1:P;TAB(7);"D";
2020 FOR I=1 TO N :: PRINT #1:TAB(18+I*4);D(I);:: NEXT I
2030 PRINT #1 :: RETURN
9999 CLOSE #1 :: END

```

Pass #	D	Position ->	1	2	3	4	5	6	7	8	9	10
0	D		22	10	6	26	39	29	16	45	9	34
1	D		9	10	6	16	22	29	39	45	26	34
2	D		6	9	10	16	22	29	39	45	26	34
3	D		6	9	10	16	22	29	39	45	26	34
4	D		6	9	10	16	22	26	29	45	39	34
5	D		6	9	10	16	22	26	29	34	39	45
6	D		6	9	10	16	22	26	29	34	39	45

Tree Sorts

Tree sorts represent the fourth category of techniques for sorting. We will show you one such sort based on the B-tree, called the *BSST sort*. This method of rearranging a list is very different from all preceding methods because it produces a set of pointers to the original list. When these pointers are used to access that still unsorted list, the result is the list in sorted order. The pointers are called left and right links, and each data element in any list to be sorted requires both links. Also, the sort has to have a stack structure for the sorted display procedure. The sort works in two phases: First, the appropriate left and right links are generated in their arrays; second, these links are used to traverse the tree, that is to access each list element in order.

Program TREESORT's output is slightly different in that it shows first the original data with its generated links, then the list in sorted order.

```

10 REM filename: "treesort"
20 REM purpose: Tree sort (BSST)
30 REM author: jpg & jdr 9/82
40 REM -----
50 DIM D(100),STK(75)
60 DIM LL(100),RL(100)
62 RANDOMIZE :: CALL CLEAR
65 OPEN #1:"RS232"
70 REM Place random integers between 1 and 50 into D.
80 FOR I=1 TO 10 :: D(I)=INT(RND*50+1):: NEXT I
90 N=10 :: GOSUB 1000
100 GOTO 9999
1000 REM ***** Tree sort (BSST) *****
1010 REM First, build all links.
1020 P=1 :: J=0
1030 X=1 :: IF P>N THEN 1060
1040 IF D(P)>D(X) THEN IF RL(X)=0 THEN RL(X)=P :: GOTO 1050 ELSE X=RL(X):: GOTO 1
040 ELSE IF LL(X)=0 THEN LL(X)=P :: GOTO 1050 :: ELSE X=LL(X):: GOTO 1040
1050 LL(P)=0 :: RL(P)=0 :: P=P+1 :: GOTO 1030
1060 GOSUB 1140 !Print table of data, links.
1070 REM !Then traverse the tree using the links.
1080 P=1 :: T=0
1090 T=T+1 :: STK(T)=P
1100 IF P<>0 THEN P=LL(P):: GOTO 1090 ELSE T=T-1 :: P=STK(T)
1110 IF T=0 THEN PRINT #1 :: RETURN
1120 PRINT #1:D(P);: T=T-1 :: P=RL(P):: GOTO 1090
1130 P=P+1 :: GOSUB 1140 :: GOTO 1030
1140 REM ***** Print table of data links *****
2000 PRINT #1:"Position "; "Contents "; "Left link "; "Right link"
2010 FOR J=1 TO N
2020 PRINT #1:J;TAB(15);D(J);TAB(30);LL(J);TAB(45);RL(J)
2030 NEXT J :: PRINT #1
2040 RETURN
9999 CLOSE #1 :: END

```

Position	Contents	Left link	Right link
1	1	0	2
2	7	3	4
3	5	6	8
4	21	9	5
5	23	0	7
6	3	0	0
7	49	0	0
8	6	0	0
9	10	0	10
10	16	0	0

1 3 5 6 7 10 16 21 23 49

Multikey Sorts

The next program concludes this portion of the chapter, which has been devoted to listing one sorting algorithm after another. The following program is nothing really new; it uses the Shell-Metzner sort as its rearrangement algorithm, but it could have used any one of the techniques. What is new here is the flexibility of the sort.

Imagine a doubly subscripted array of data dimensioned D(10,5). The first subscript represents the item number, and the second subscript represents the sub-field within that item. For example, D(I,1) could be the ID number, D(I,2) wages, D(I,3) hours worked, D(I,4) deductions, and D(I,5) job type. The type of program which the next listing exemplifies is called a multikey sort, wherein the user has a choice of which sub-field or key to operate upon.

D could just as well be a string array, dimensioned say M\$(14), as in the next program. The problem is to have a program that can sort on any one of the substring fields in the string array. Program MUSHSORT demonstrates a multikey sort of a string array on data extracted from two books on mushrooms.

```

10 REM filename: "mushsort"
20 REM purpose: multikey sort of string data
30 REM author: jpg & jdr 9/82
40 REM -----
50 DIM S(70),M$(14)! S is starting column, M$ is mushroom.
60 FOR K=1 TO 7 :: READ S(K):: NEXT K
70 FOR I=1 TO 14 :: READ M$(I):: NEXT I
80 CALL CLEAR
90 PRINT "Select the field on which to sort."
100 PRINT "1 = page #, 'Les Champignons de France', V. 2"
110 PRINT "2 = page #, 'Non-Flowering Plants'"
120 PRINT "3 = common name"
130 PRINT "4 = Latin name"
140 PRINT "5 = edibility"
150 PRINT "6 = color of cap"
160 INPUT "What key":K
170 C=S(K)!C is column number of field.
180 L=S(K+1)-S(K)!L is length of field.
190 N=14 :: GOSUB 1000
200 CALL CLEAR :: FOR I=1 TO 14 :: PRINT M$(I):: NEXT I
210 INPUT "/EN/":A$
220 GOTO 80
500 DATA 1,3,5,22,43,50,56
510 DATA "154Caesar's Amanita Amanita Caesaria V.Good Orange"
520 DATA "352Death Cap Amanita Phallooides Deadly Whitsh"
530 DATA "452Destroying Angle Amanita Virosa Deadly White "
540 DATA "552Spring Amanita Amanita Verna Deadly White "
550 DATA "7--Lemon Amanita Amanita Citrina SuspectYellow"
560 DATA "853Fly Amanita Amanita Muscaria Poison Red "
570 DATA "953Panther Amanita Amanita Pantherina Poison Brown "
580 DATA "1154Blusher Amanita Rubescens Good Reddsh"
590 DATA "1158Parasol Lepiota Lepiota Procera Good Brown "
600 DATA "1759Smooth Lepiota Lepiota Naucina Good White "
610 DATA "2181Horse Mushroom Agaricus Arvensis V.Good White "
620 DATA "2481Field Mushroom Agaricus Campestris Good White "
630 DATA "2684Shaggy Mane Coprinus Comatus Good White "
640 DATA "8061Blewit Tricholoma PersonatumV.Good Grey "
1000 REM ***** Shell-Metzner Sort *****
1010 M=N
1020 M=INT(M/2)
1030 IF M=0 THEN RETURN
1040 K=N-M :: J=1
1050 I=J
1060 P=I+M
1070 IF SEG$(M$(I),C,L)<=SEG$(M$(P),C,L) THEN 1100
1080 T$=M$(I):: M$(I)=M$(P):: M$(P)=T$ :: I=I-M
1090 IF I>=1 THEN 1060
1100 J=J+1
1110 IF J<=K THEN 1050 ELSE 1020
9999 END

```

Select the field on which to sort.

1 = page #, 'Les Champignons de France', V. 2

2 = page #, 'Non-Flowering Plants'

3 = common name

4 = Latin name

5 = edibility

6 = color of cap

What key?

7--Lemon Amanita	Amanita Citrina	Suspect	Yellow
452Destroying Angle	Amanita Virosa	Deadly	White
352Death Cap	Amanita Phallooides	Deadly	Whitsh
552Spring Amanita	Amanita Verna	Deadly	White
853Fly Amanita	Amanita Muscaria	Poison	Red
953Panther Amanita	Amanita Pantherina	Poison	Brown
154Caesar's Amanita	Amanita Caesaria	V.Good	Orange
1154Blusher	Amanita Rubescens	Good	Reddsh
1158Parasol Lepiota	Lepiota Procera	Good	Brown
1759Smooth Lepiota	Lepiota Naucina	Good	White
8061Blewit	Tricholoma Personatum	V.Good	Grey
2481Field Mushroom	Agaricus Campestris	Good	White
2181Horse Mushroom	Agaricus Arvensis	V.Good	White
2684Shaggy Mane	Coprinus Comatus	Good	White

8061Blewit	Tricholoma Personatum	V.Good	Grey
1154Blusher	Amanita Rubescens	Good	Reddsh
2481Field Mushroom	Agaricus Campestris	Good	White
2181Horse Mushroom	Agaricus Arvensis	V.Good	White
1158Parasol Lepiota	Lepiota Procera	Good	Brown
2684Shaggy Mane	Coprinus Comatus	Good	White
1759Smooth Lepiota	Lepiota Naucina	Good	White
154Caesar's Amanita	Amanita Caesaria	V.Good	Orange
953Panther Amanita	Amanita Pantherina	Poison	Brown
352Death Cap	Amanita Phallooides	Deadly	Whitsh
7--Lemon Amanita	Amanita Citrina	Suspect	Yellow
452Destroying Angle	Amanita Virosa	Deadly	White
853Fly Amanita	Amanita Muscaria	Poison	Red
552Spring Amanita	Amanita Verna	Deadly	White

1759Smooth Lepiota	Lepiota Naucina	Good	White
1158Parasol Lepiota	Lepiota Procera	Good	Brown
2181Horse Mushroom	Agaricus Arvensis	V.Good	White
2481Field Mushroom	Agaricus Campestris	Good	White
154Caesar's Amanita	Amanita Caesaria	V.Good	Orange
7--Lemon Amanita	Amanita Citrina	Suspect	Yellow
853Fly Amanita	Amanita Muscaria	Poison	Red
953Panther Amanita	Amanita Pantherina	Poison	Brown
352Death Cap	Amanita Phallooides	Deadly	Whitsh
1154Blusher	Amanita Rubescens	Good	Reddsh
552Spring Amanita	Amanita Verna	Deadly	White
452Destroying Angle	Amanita Virosa	Deadly	White
2684Shaggy Mane	Coprinus Comatus	Good	White
8061Blewit	Tricholoma Personatum	V.Good	Grey

Summary

The final program, COMPSORT, is simply a main program that calls most of the sorting subroutines one at a time to sort an increasingly large subset of an array containing random numbers. The output is a chart that indicates the relative efficiencies of the sorts we have included in this chapter by counting the number of passes and exchanges.

```
10 REM filename: "compsort"
20 REM purpose: Comparison of sorts
30 REM author: jpg & jdr 9/82
40 REM -----
50 REM D is data array      D2 for Brute Force
60 REM LL, RL are left and right links for Tree
70 REM STK is stack for Quicksort and Tree
80 REM N$ is sort name      NM is size of sort
90 RANDOMIZE
100 DIM D(500),D1(500),LL(500),RL(500),STK(300),NM(10),N$(10)
110 CALL CLEAR
120 REM          Print table heading.
130 PRINT "Sort #->";
140 DATA "Brute Force","Insertion","Bubble","Exchange"
150 DATA "Del. Exch.,""Shell","Shell-Metzner","Heap"
160 DATA "Quicksort","Tree"
170 DATA 10,20,30,40,50
180 N2=5
190 FOR K=1 TO 10 :: READ N$(K):: NEXT K
200 FOR I=1 TO N2
210 READ NM(I):: PRINT TAB(8+7*I+4);NM(I);
220 NEXT I :: PRINT
230 REM          Outer loop: Select sort.
240 FOR S1=1 TO 10 !Change (i.e. 7 to 10) for shallow chart.
250 PRINT N$(S1);
260 FOR I1=1 TO N2 :: N=NM(I1)!Inner loop.
270 FOR P=1 TO N :: D(P)=INT(RND*1000+1):: NEXT P
280 PS=0 :: EX=0
290 IF S1>7 THEN ON S1-7 GOSUB 8000,9000,1000
300 IF S1<=7 THEN ON S1 GOSUB 1000,2000,3000,4000,5000,6000,7000
320 PRINT TAB(8+7*I1+4);PS;"/";EX;
330 NEXT I1 :: PRINT
340 NEXT S1
350 GOTO 20000
1000 REM***** Brute force sort. *****
1010 FOR P=1 TO N :: PS=PS+1
1020 S=9999
1030 FOR J=1 TO N
1040 IF D(J)<S THEN S=D(J):: X=J :: EX=EX+1
1050 NEXT J :: D1(P)=D(X):: D(X)=S
1060 NEXT P
1070 RETURN
2000 REM ***** Insertion Sort *****
2010 FOR P=1 TO N-1
2020 PS=PS+1
2030 X=D(P+1)
2040 FOR J=P TO 1 STEP -1
2050 IF X>=D(J) THEN 2070
2060 D(J+1)=D(J):: EX=EX+1 :: NEXT J :: J=0
2070 D(J+1)=X
2080 NEXT P
2090 RETURN
```

```

3000 REM ***** Bubble Sort *****
3010 FOR P=1 TO N-1
3020 F=0 :: PS=PS+1
3030 FOR J=1 TO N-P
3040 IF D(J+1)<D(J) THEN T=D(J):: D(J)=D(J+1):: D(J+1)=T :: F=1 :: EX=EX+1
3050 NEXT J
3060 IF F=0 THEN RETURN
3070 NEXT P
3080 RETURN
4000 REM ***** Exchange (or selection sort) *****
4010 FOR P=1 TO N-1 :: PS=PS+1
4020 FOR J=P+1 TO N
4030 IF D(P)>D(J) THEN T=D(P):: D(P)=D(J):: D(J)=T :: EX=EX+1
4040 NEXT J
4050 NEXT P
4060 RETURN
5000 REM ***** Delayed exchange sort *****
5010 FOR P=1 TO N-1
5020 X=P :: PS=PS+1
5030 FOR J=P+1 TO N
5040 IF D(J)<D(X) THEN X=J
5050 NEXT J
5060 IF P<>X THEN T=D(X):: D(X)=D(P):: D(P)=T :: EX=EX+1
5070 NEXT P
5080 RETURN
6000 REM ***** Shell sort *****
6010 P=N
6020 IF P<=1 THEN RETURN
6030 P=INT(P/2):: M=N-P
6040 F=0 :: PS=PS+1
6050 FOR J=1 TO M
6060 X=J+P
6070 IF D(J)>D(X) THEN T=D(J):: D(J)=D(X):: D(X)=T :: F=1 :: EX=EX+1
6080 NEXT J
6090 IF F>0 THEN 6040 ELSE 6020
7000 REM ***** Shell-Metzner sort *****
7010 P=N
7020 P=INT(P/2)
7030 IF P=0 THEN RETURN
7040 K=N-P :: J=1 :: PS=PS+1
7050 I=J
7060 L=I+P
7070 IF D(I)<D(L) THEN 7100
7080 T=D(I):: D(I)=D(L):: D(L)=T :: I=I-P :: EX=EX+1
7090 IF I>=1 THEN 7060
7100 J=J+1
7110 IF J<=K THEN 7050 ELSE 7020
8000 REM ***** Heap Sort *****
8010 X=INT(N/2)+1 :: P=N
8020 PS=PS+1 :: IF X<>1 THEN X=X-1 :: A=D(X):: J=X :: GOTO 8050
8030 A=D(P):: D(P)=D(1):: P=P-1
8040 IF P<>1 THEN J=X ELSE D(1)=A :: RETURN
8050 I=J :: J=J+J
8060 IF J=P THEN 8090
8070 IF J>P THEN D(I)=A :: GOTO 8020
8080 IF D(J)<D(J+1) THEN J=J+1
8090 IF A<D(J) THEN D(I)=D(J):: EX=EX+1 :: GOTO 8050 ELSE D(I)=A :: GOTO 8020
9000 REM ***** Quicksort *****
9010 X=0 :: I=X+X :: STK(I+1)=1 :: STK(I+2)=N :: X=X+1
9020 IF X=0 THEN RETURN
9030 X=X-1 :: I=X+X :: A=STK(I+1):: B=STK(I+2)
9040 Z=D(A):: TP=A :: BT=B+1
9050 BT=BT-1
9060 IF BT=TP THEN 9110

```

```

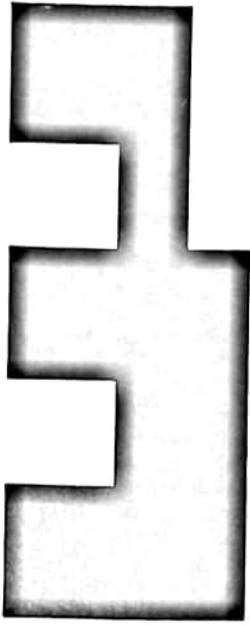
9070 IF Z<=D(BT)THEN 9050 ELSE D(TP)=D(BT):: EX=EX+1
9080 TP=TP+1
9090 IF BT=TP THEN 9110
9100 IF Z>D(TP)THEN 9080 ELSE D(BT)=D(TP):: EX=EX+1 :: GOTO 9050
9110 D(TP)=Z
9120 IF B-TP>=2 THEN I=X+X :: STK(I+1)=TP+1 :: X=X+1 :: STK(I+2)=B
9130 IF BT-A>=2 THEN I=X+X :: STK(I+1)=A :: X=X+1 :: STK(I+2)=BT-1
9140 P=P+1 :: PS=PS+1 :: GOTO 9020
10000 REM ***** Tree sort (BSST) *****
10010 P=1 :: J=0
10020 X=1 :: IF P>N THEN 10040
10030 IF D(P)>D(X)THEN IF RL(X)=0 THEN RL(X)=P :: GOTO 10040 ELSE X=RL(X):: GOTO
10030 ELSE IF LL(X)=0 THEN LL(X)=P :: GOTO 10040 :: ELSE X=LL(X):: GOTO 10030
10040 LL(P)=0 :: RL(P)=0 :: P=P+1 :: PS=PS+1 :: GOTO 10020
10050 P=1 :: T=0 !Traverse
10060 T=T+1 :: STK(T)=P
10070 IF P<>0 THEN P=LL(P):: GOTO 10060 ELSE T=T-1 :: P=STK(T)
10080 IF T=0 THEN RETURN
10090 T=T-1 :: P=RL(P):: GOTO 10060
20000 END

```

Sort #->	10	20	30	40	50
Brute Force	10/30	20/40	30/60	40/200	50/450
Insertion	9/15	19/89	29/205	39/374	49/582
Bubble	7/19	16/122	26/203	33/431	44/611
Exchange	9/20	19/106	29/214	39/384	49/595
Del. Exch.	9/5	19/18	29/26	39/37	49/45
Shell	9/11	10/27	13/64	18/98	19/118
Shell-Metzner	3/10	4/29	4/62	5/114	5/177
Heap	14/20	29/51	44/92	59/134	74/188
Quicksort	7/8	13/28	21/49	26/69	33/94
Tree	10/30	20/20	30/90	40/160	50/150

References

- Dwyer, Thomas A., and Margot Critchfield, *BASIC and the Personal Computer*, (Addison-Wesley, 1978), pp. 196-234.
- Grillo, John P., "A Comparison of Sorts", *Creative Computing*, Nov.-Dec., 1976), pp. 76-79.
- Grillo, John P., and J. D. Robertson, *Microcomputer Systems: An Applications Approach*, (Wm. C. Brown, 1979), pp. 192-212.
- Knuth, D. E., *The Art of Computer Programming, Volume 3: Searching and Sorting*, (Addison-Wesley).
- Nijenhuis, Albert, "How Not to Be Out of Sorts", *Creative Computing*, (Aug., Sept., Oct. 1980).



Strings

During the decade of the 70's, the use of computers changed dramatically to favor the individual user. This emphasis on distributed processing resulted in some applications which had not been considered appropriate for the larger, more scientific and business oriented computers. Two such applications are game playing and word processing.

Most computer games incorporate a wide variety of programming techniques, and in terms of the subject of this book, data management, it is fair to say that we could have included in each chapter several games that exemplify the techniques we discuss. We will include three games in this chapter because they are appropriate and suitable to the topic. We will use some games in other chapters because they also suit the techniques involved, such as G1A and Acey-Ducey in Chapter 1.

Word Processing

Word Processing has emerged as an important task for the computer to perform because it helps workers who deal with files and records of text information, such as mail-order houses, publishers, lawyers, and writers. These disciplines require the management of words as the data elements rather than values. The computer processes these data items by resorting to special instructions and routines that search, substitute, concatenate, extract, and arrange words and letters. It is not fruitful for the computer to perform arithmetic operations on words and letters.

BASIC is particularly appropriate as a language for word processing because of its large and flexible set of string manipulation functions. If the source program can be compiled with one of the presently available BASIC compilers, it becomes even more efficient and useful. Most commercial word processing programs, such as Radio Shack's SCRIPSIT or Easywriter II for the IBM PC, are not written in BASIC. They are written in the computer's assembly language for a variety of reasons. These commercial programs are intended for a diverse set of users, and can be considered in the class of utility programs. It is not our intent to show you how to write a word processing utility in BASIC, although you could if you so desired. Instead, we will describe some techniques that require the use of string management instructions.

Random Word Selection

The random selection of a word from a list is no different from choosing a value, except of course that the chosen word must be assigned a string variable name. Program JOTTO closely resembles the popular 5-letter word game by the same name. We include it here to demonstrate one way to store string data, as DATA statements, and to show you some elementary string searching techniques. The point of the game is to guess the word which the computer has selected at random from the list. After the player guesses, the computer first checks for duplicate letters, as none are allowed in this version of the game. Then it tallies all like letters in matching positions and builds its clue string P\$ to include all such matches. When the number of matches is 5, the game is over and the user is given the option to play again.

This game leaves much to be desired in embellishment, which we leave up to the user. Graphics would be a big help, as would some slight changes of rules, such as limiting the number of guesses or allowing duplicate letters in the five-letter words.

```

10 REM filename: "jotto"
20 REM purpose: Jotto, or 5-letter word game
30 REM author: jpg & jdr 10/82
40 REM -----
50 RANDOMIZE :: CALL CLEAR
60 OPEN #1:"RS232"
70 REM          179 WORDS IN THIS VERSION.
80 DIM W$(300)!Add 10 words at a time, max.==300.
90 REM          Restarting point.
100 FOR I=1 TO 300 :: READ W$(I)
110 IF W$(I)="end" THEN N=I-1 :: GOTO 130
120 NEXT I :: N=300
130 REM Number of words in then array is N
140 G=0 :: CALL CLEAR
150 S$=W$(INT(RND*N+1))!Get random guess.
160 PRINT "Guess a five-letter word."
170 INPUT L$
180 IF L$="list" THEN GOSUB 600 :: GOTO 160
190 IF L$="sort" THEN GOSUB 800 :: GOTO 160
200 IF L$="?" THEN PRINT "The secret word is ";S$ :: GOTO 420
210 IF LEN(L$)<>5 THEN PRINT " Five letters long!" :: G=G+1 :: GOTO 160
220 GOSUB 440 !Check for duplicate letters.
230 G=G+1 !Tally guesses.
240 IF L$=S$ THEN 410
250 M=0 :: Q=1 :: P$="_____" :: A$=P$
260 FOR I=1 TO 5 :: J=0

```

```

270 FOR J=1 TO 5
275 X#=SEG$(S$,I,1)
280 IF X#<>SEG$(L$,J,1) THEN 320
290 P#=SEG$(P$,1,J-1)&SEG$(L$,J,1)&SEG$(P$,J+1,5-J)
300 IF I=J THEN A#=SEG$(A$,1,J-1)&SEG$(L$,J,1)&SEG$(A$,J+1,5-J)
310 M=M+1
320 NEXT J
330 NEXT I
340 PRINT "There were";M;
350 PRINT "matches and the common letters were...",TAB(50);P$
360 PRINT "from the exact matches, you know ///";TAB(50);A$
370 IF A#=S$ THEN 410
380 IF M=1 THEN PRINT "If you give up type '?' for your next guess."
390 GOTO 160
410 PRINT "You have guessed the word in";G;"guesses!"
420 PRINT :: INPUT "Want to play again ":Q$
430 IF Q$="yes" THEN 140 ELSE STOP
440 REM ***** Check for duplicate letters *****
450 FOR I=1 TO 4
460 FOR J=I+1 TO 5
470 LET X#=SEG$(L$,I,1)
480 IF X#=SEG$(L$,J,1) THEN PRINT "Hint: All letters different." :: PRINT "Free g
uess. Try again." :: RETURN
490 NEXT J
500 NEXT I
510 RETURN
600 REM ***** Subroutine to list N words *****
610 INPUT "1=screen only 2=printer too":Z
620 FOR I=1 TO N STEP 10
630 FOR J=1 TO 10
640 PRINT W$(I+J-1);" ";
650 IF Z=2 THEN PRINT #1:W$(I+J-1);" ";
660 NEXT J :: PRINT :: IF Z=2 THEN PRINT #1
670 NEXT I
680 RETURN
800 REM ***** Shell-Metzner sort of data **
810 M=N :: PRINT N;" items to be sorted."
820 M=INT(M/2):: PRINT J:: IF M=0 THEN PRINT :: RETURN
830 K=N-M :: J=1
840 I=J
850 L=I+M :: IF W$(I)<=W$(L) THEN 870
860 LET T#=W$(I):: W$(I)=W$(L):: W$(L)=T$ :: I=I-M :: IF I>=1 THEN 850
870 J=J+1 :: IF J<=K THEN 840 ELSE 820
1000 REM***** W O R D S *****
1010 DATA phase,manic,study,spiel,stile,tough,shawl,topic
1020 DATA tonic,tunic,toxic,swarm,swine,storm,sworn,phlox
1030 DATA beaux,braid,sumac,thyme,sepal,petal,stoma,pedal
1040 DATA sciole,style,steal,taper,tapir,payer,scram,scrap
1050 DATA scrip,strip,joust,haunt,threw,throw,tober,mouse
1060 DATA sugar,adobe,steam,frond,batch,latch,loach,plume
1070 DATA peach,teach,hasty,haste,march,marsh,scary,wormy
1080 DATA remit,blond,quilt,brisk,peril,plumb,plump,poach
1090 DATA roach,leash,beach,quash,reach,earth,spawn,least
1100 DATA house,touch,mouth,imbue,anger,avoid,squib,slope
1110 DATA adobe,yeast,input,point,print,panic,spire,prank
1120 DATA rough,ghost,quest,pleat,smelt,cable,charm,dwelt
1130 DATA guest,harpy,judge,spare,spore,sport,spurt,spoil
1140 DATA spout,weild,apish,capon,coral,copra,cobra,flock
1150 DATA lunar,mauve,noise,plaid,quoit,divot,ducat,ducal
1160 DATA douse,rouse,louse,drape,drupe,cigar,patch,match
1170 DATA watch,awoke,atone,anode,staid,women,rival,spray
1180 DATA tonal,vouvh,pouch,toxin,wrist,wrest,choir,chair
1190 DATA stair,chaos,moxie,woman,xylem,alone,ruled,prose
1200 DATA stoic,azote,anole,arose,leach,rebus,plank,proud
1210 DATA realm,qualm,quark,sitar,weird,swale,shale,scale
1220 DATA scald,scalp,scarp,sharp,share,snare,squat,squid
1230 DATA spate,spite,gonad,end
9999 CLOSE #1 :: END

```



```

Guess a five-letter word.
about
There were 3 matches and the common letters were...
                                         __out
from the exact matches, you know ///
                                         -----
Guess a five-letter word.
joust
There were 3 matches and the common letters were...
                                         _ou_t
from the exact matches, you know ///
                                         _ou__
Guess a five-letter word.
tough
There were 4 matches and the common letters were...
                                         tou_h
from the exact matches, you know ///
                                         _ou_h
Guess a five-letter word.
mouth
You have guessed the word in 4 guesses!
Want to play again?

```

Pattern Matching

The next two games are probably more familiar to you than JOTTO, although perhaps not quite in their present form. PICOFORM is a variation of Pico-Fermi-Bagels, or Bull-Cow, or Max-Mix-Mux. The computer generates a three-digit number, all digits different. The user guesses a number and the computer compares that guess to its selected target. It reports every correct digit in the right position as a "Pico", every correct digit in the wrong position as a "Fermi", and every wrong digit as a "Bagel". The object is to have the computer respond "Pico Pico Pico" in as few guesses as possible.

MASTRMND is a variation of Mastermind, which is itself a variation of Pico-Fermi-Bagels. This version is the simplistic one that produces the color code which the player must guess. A much better version exists in *101 BASIC Games: Microcomputer Version* (Creative Computing Press, 1978). In that program the computer acts first as color code maker, then color code breaker. To have the computer do a good job of guessing the player's colors, and trapping the player's bad responses to its guesses, is definitely not a trivial task.

```

10 REM filename: "picoferm"
20 REM purpose: Game of Pico-Fermi-Bagels
30 REM author: jpg & jdr 10/82
40 REM -----
50 RANDOMIZE :: CALL CLEAR
60 GOSUB 1000
70 G=1 :: PRINT "P I C O F E R M I B A G L E S ";
80 PRINT "Guess #   Matches   Your guess."
90 INPUT N1$
100 GOSUB 2000
110 P=G+3
130 PRINT G;A$;N1$;
140 G=G+1
150 IF PC<3 THEN 90
160 PRINT "You guessed in";G-1;"times."
170 PRINT "Play again?" :: INPUT A$
180 IF A$="YES" OR A$="yes" THEN 50 ELSE STOP

```

```

1000 REM***** Generate different-digit random number****
1010 N=INT(RND*999+1)
1020 A=INT(N/100)
1030 B=N-100*A :: B=INT(B/10)
1040 C=N-100*A-10*B
1050 IF A=B OR A=C OR B=C THEN 1010
1060 IF N<100 THEN N$="0"&SEG$(STR$(N),1,2)ELSE N$=SEG$(STR$(N),1,3)
1070 RETURN
2000 REM***** Subroutine to get matches *****
2010 PC=0 :: FE=0 :: A$=""
2020 FOR I=1 TO 3
2030 X$=SEG$(N1$,I,1):: IF X$=SEG$(N$,I,1)THEN PC=PC+1 :: A*=A$&"Pico  " :: GOT
O 2080
2040 FOR J=1 TO 3
2050 IF I=J THEN 2070
2060 X$=SEG$(N1$,I,1):: IF X$=SEG$(N$,J,1)THEN FE=FE+1
2070 NEXT J
2080 NEXT I
2090 IF FE=0 THEN 2110
2100 FOR I=1 TO FE :: A*=A$&"Fermi  " :: NEXT I
2110 BA=3-PC-FE
2120 IF BA=0 THEN RETURN
2130 FOR I=1 TO BA :: A*=A$&"Bagels  " :: NEXT I
2140 RETURN
9999 END

```

```

P I C O F E R M I B A G L E S
Guess #   Matches      Your guess.
123
 1 Fermi  Bagels Bagels 123
132
 2 Pico   Bagels Bagels 132
231
 3 Pico   Bagels Bagels 231
435
 4 Pico   Pico   Bagels 435
534
 5 Pico   Fermi  Bagels 534
635
 6 Pico   Bagels Bagels 635
436
 7 Pico   Pico   Bagels 436
437
 8 Pico   Pico   Bagels 437
438
 9 Pico   Pico   Bagels 438
439
10 Pico   Pico   Bagels 439
430
11 Pico   Pico   Pico   430
You guessed in 11 times.
Play again?

```

```

10 REM filename: "mastrmnd"
20 REM purpose: To play the game of Mastermind
30 REM author: jpg & jdr 10/82
40 REM -----
50 REM Game goal is to guess color scheme selected by computer.
60 REM Four colors are used in any combination.
70 REM Use R for red, G for green, Y for yellow, B for blue.
80 REM Examples RBBB, RBGY, GBBR
90 REM Input a ? when a new scheme is desired.
100 REM Input a . when new scheme is desired.
110 REM Only 4 letter combinations.
120 REM B records number of colors guessed in right position.
130 REM W records number of colors guessed in wrong position.
140 DIM C$(4)
150 DATA R,G,B,Y
160 RANDOMIZE :: CALL CLEAR
170 REM This loop reads four colors into table c$
180 FOR I=1 TO 4 :: READ C$(I):: NEXT I
190 W$=""
200 REM This loop randomly selects 4 colors.
210 FOR I=1 TO 4
220 N=INT(RND*4+1)
230 W$=W$&C$(N)
240 NEXT I
250 DW$=W$
260 PRINT "Guess scheme. Type any combination of RGBY"
270 PRINT "Type ? to select another scheme or . to quit."
280 PRINT "If B=4, you guessed it. Select new scheme, or quit."
290 W$=DW$
300 INPUT G$
310 IF SEG$(G$,1,1)="?" THEN 190
320 IF SEG$(G$,1,1)="." THEN 999
330 DG$=G$
340 B=0
350 REM This loop records B, right color, right place.
360 REM These also replace matches with asterisk.
370 FOR I=1 TO 4
380 X$=SEG$(W$,I,1):: IF X$=SEG$(G$,I,1) THEN B=B+1 :: GOSUB 600
390 NEXT I
400 W=0
410 REM This loop compares the remaining combinations,
420 REM and counts 1 for 1 color matches.
430 FOR I=1 TO 4
440 IF "*" = SEG$(W$,I,1) THEN 480
450 FOR J=1 TO 4
460 X$=SEG$(W$,I,1):: IF X$=SEG$(G$,J,1) THEN W=W+1 :: GOSUB 600 :: GOTO 480
470 NEXT J
480 NEXT I
490 PRINT "B> ";B,"W> ";W
500 PRINT "B is right color in right location"
510 PRINT "W is number of right colors in wrong location."
520 GOTO 270
600 REM Subroutine to substitute asterisk. <<HMZ>>
610 DIM W2$(4),G2$(4)
620 FOR Z=1 TO 4
630 W2$(Z)=SEG$(W$,Z,1):: G2$(Z)=SEG$(G$,Z,1)
640 NEXT Z
650 W2$(I)="*" :: G2$(I)="*"
670 W$=W2$(1)&W2$(2)&W2$(3)&W2$(4)
680 G$=G2$(1)&G2$(2)&G2$(3)&G2$(4)
690 RETURN
999 END

```

```

Guess scheme. Type any combination of RGBY
Type ? to select another scheme or . to quit.
If B=4, you guessed it. Select new scheme, or quit.
RRRR
B> 0          W> 0
B is right color in right location
W is number of right colors in wrong location.
Type ? to select another scheme or . to quit.
If B=4, you guessed it. Select new scheme, or quit.
GGGG
B> 2          W> 0
B is right color in right location
W is number of right colors in wrong location.
Type ? to select another scheme or . to quit.
If B=4, you guessed it. Select new scheme, or quit.
GRRR
B> 1          W> 1
B is right color in right location
W is number of right colors in wrong location.
Type ? to select another scheme or . to quit.
If B=4, you guessed it. Select new scheme, or quit.
GRRR
B> 1          W> 0
B is right color in right location
W is number of right colors in wrong location.
Type ? to select another scheme or . to quit.
If B=4, you guessed it. Select new scheme, or quit.
RRRG
B> 0          W> 2
B is right color in right location
W is number of right colors in wrong location.
Type ? to select another scheme or . to quit.
If B=4, you guessed it. Select new scheme, or quit.
RYRY
B> 1          W> 0
B is right color in right location
W is number of right colors in wrong location.
Type ? to select another scheme or . to quit.
If B=4, you guessed it. Select new scheme, or quit.
RRRY
B> 0          W> 1
B is right color in right location
W is number of right colors in wrong location.
Type ? to select another scheme or . to quit.
If B=4, you guessed it. Select new scheme, or quit.
GYGB
B> 4          W> 0
B is right color in right location
W is number of right colors in wrong location.
Type ? to select another scheme or . to quit.
If B=4, you guessed it. Select new scheme, or quit.

```

Text Encoding

The second group of programs shows some of the more useful if perhaps less entertaining, techniques of string management programs. SUBCODE and GRAFCODE are text encoders. They accept strings of text from the user, then transform them into a weird looking set of encrypted information. Program SUBCODE uses a type of encryption called Playfair code. The user supplies a password, then the computer generates a 25-character long second alphabet based on this word. When the user submits a message as a string of text, the program transposes each letter of the message to its corresponding second alphabet representation. This is the encrypted message. The user then supplies the proper password and the computer decodes the encrypted message into plaintext.

This program serves no purpose except to demonstrate the technique of substitution code encryption. We urge you to devise a game based on this program. One player makes up a password and message, and the other tries to break the code and decypher the message. This is the principle of the cryptogram found in the crossword puzzles in the newspaper: simple substitution coding.

```
10 REM filename: "subcode"
20 REM purpose: Simple substitution code
30 REM author: jpg & jdr 10/82
40 REM -----
50 CALL CLEAR
60 PRINT "Type a one-line message with no punctuation."
70 INPUT X$
80 INPUT "Type a one-word password, the larger the better ";P$
90 GOSUB 1000
100 PRINT "Computer-encoded text:"
110 FOR I=1 TO LEN(C$)STEP 5
120 PRINT SEG$(C$,I,5);" ";
130 NEXT I
140 PRINT
150 GOSUB 2000
160 PRINT "Plaintext message as decoded by computer:"
170 PRINT D$
180 STOP
1000 REM*** Subroutine for Playfair code encryption ****
1010 K$="abcdefghijklmnopqrstuvwxy" :: A$=P$&K$
1020 M$=SEG$(P$,1,1):: P=2
1030 FOR I=2 TO LEN(A$)
1040 T$=SEG$(A$,P,1)
1050 FOR J=1 TO I-1
1060 IF SEG$(M$,J,1)=T$ THEN 1080
1070 NEXT J :: M$=M$&T$
1080 P=P+1
1090 NEXT I :: PRINT P$
1100 FOR I=1 TO LEN(X$)
1110 Y$=SEG$(X$,I,1)
1120 FOR J=1 TO 25
1130 IF Y$=SEG$(K$,J,1) THEN C$=C$&SEG$(M$,J,1):: GOTO 1150
1140 NEXT J
1150 NEXT I
1160 RETURN
```

```

2000 REM Subroutine to decrypt a Playfair-coded message.
2010 M2$="abcdefghijklmnopqrstuvwxy"
2020 A1$=P$&M2$
2030 M1$=SEG$(P$,1,1)
2040 P=2
2050 FOR I=2 TO LEN(A1$)
2060 T$=SEG$(A1$,P,1)
2070 FOR J=1 TO I-1
2080 IF T$=SEG$(M1$,J,1)THEN 2100
2090 NEXT J :: M1$=M1$&T$
2100 P=P+1
2110 NEXT I :: PRINT "Computer rebuilt substitution string:"
2120 PRINT M1$
2130 FOR I=1 TO LEN(C$)
2140 Y$=SEG$(C$,I,1)
2150 FOR J=1 TO 25
2160 IF Y$=SEG$(M1$,J,1)THEN D$=D$&SEG$(M2$,J,1):: GOTO 2180
2170 NEXT J
2180 NEXT I
2190 RETURN
9999 END

```

```

Type a one-line message with no punctuation.
Type a one-word password, the larger the better
Type a one-line message with no punctuation.
Type a one-word password, the larger the better
audubonsociety
Computer-encoded text:
auayy xchhi moipc olhnp vdhf fhgaf olida gcayn uoatm
Computer rebuilt substitution string:
audbonscietyfghjklmpqrvwxz
Plaintext message as decoded by computer:
aballyhooiseitheroftwocommonamericanhalfbeaks

```

Text Reordering

Program KWICINDEX produces a form of KWIC (Key Word In Context) index. It lists each journal title submitted to it in every variation, printing first an entry starting with its first nontrivial word, then the rest of the title. For example, the journal title "The Theory of Parsing and Compiling" should be listed in these forms, with each of the three keywords first.

Theory of Parsing and Compiling, The
 Parsing and Compiling, The Theory of
 Compiling, The Theory of Parsing and

The program has to have available a list of common words to eliminate as keywords. We supply them here as they appeared in Gregory Yob's article, "Tokens for Text" in the "Personal Electronic Transactions" section of the February 1980 issue of *Creative Computing* (page 160). The program KWICINDEX assesses the DATA statements that hold all of the common words and journal titles. If the program were in its final production form, its next task would be to sort all the generated KWIC index entries into a long list.

```

10 REM filename: "kwicindx"
20 REM purpose: KWIX index producer
30 REM author: car, jpg & jdr 10/82
40 !*****
50 !* Program produces a set of KWIC index entries, one*
60 !* for each non-trivial word in a title. Titles and*
70 !* nonsignificant words are loaded into an array. *
80 !* String functions, arrays, are used in problem *
90 !* solving. One array is used as a pointer. *
100 !*****
110 CALL CLEAR :: OPEN #1:"RS232"
120 !*****
130 !* Clear screen *
140 !* NS$ array is loaded with nonsignificant words. *
150 !* P(7) array will hold position of first word size*
160 !* for example, P(1)=1 where first 1 letter word is*
170 !*****
180 DIM NS$(99)
190 DIM P(7)
200 P(7)=100
210 K=0
220 !*****
230 !* Recorded in p(k)- the position of the first word*
240 !* of a different size is put in the P array. *
250 !*****
260 FOR I=1 TO 99
270 READ NS$(I)
280 IF LEN(NS$(I))>K THEN K=K+1 :: P(K)=I
290 NEXT I
300 !*****
310 !* Below are two arrays are created for titles. *
320 !* T$ holds titles as they are. *
330 !* D$ concatenates a space at either end. *
340 !* Space -used in the program, to check for a word.*
350 !* Titles are read in T$ and put in D$. *
360 !*****
370 DIM D$(6)
380 DIM T$(6)
390 FOR I=1 TO 6
400 READ T$(I)
410 D$(I)=" "&T$(I)&" "
420 NEXT I
430 W$=""
440 FOR I=1 TO 6
450 !*****
460 !* Below each character is examined in a title by J*
470 !* When a character is found, then a check is *
480 !* made of the previous one (J-1) and the following*
490 !* one (J+1). This is to determine if this is a *
500 !* word. In any case if the previous character was*
510 !* a space, the position of J is put into F to *
520 !* record the first character of the word. *
525 !* Each character is put in W$. *
530 !*****
540 FOR J=1 TO LEN(D$(I))
550 IF SEG$(D$(I),J,1)<>" " THEN 600
560 NEXT J
570 NEXT I
580 NEXT I
590 GOTO 870
600 IF SEG$(D$(I),J-1,1)=" " THEN F=J
610 W$=W$&SEG$(D$(I),J,1)
620 !*****
630 !* Below, the following character is checked. *
640 !* If a space, then a word has been found. *
650 !* If not, another character in D$ is examined. *
660 !*****

```

```

670 IF SEG$(D$(I),J+1,1)<>" " THEN 570
680 !*****
690 !* When a word is found in W$, first its length is *
700 !* checked. If length of W$ > 6, a word has been *
710 !* found. If not, the length of the word will *
720 !* determine where to go in array P, which will *
730 !* indicate between what locations in array NS$ to *
740 !* look for the NS word. *
750 !*****
760 IF LEN(W$)>6 THEN 840
770 FOR K=P(LEN(W$))TO P(LEN(W$)+1)-1
780 IF W$=NS$(K)THEN W$="" :: GOTO 570
790 NEXT K
800 !*****
810 !* If word is not found in NS$, it is significant. *
820 !* The title string is printed out using position P*
830 !*****
840 PRINT #1:SEG$(T$(I),F-1,LEN(T$(I))-F+2);"/";SEG$(T$(I),1,F-2)
850 W$=""
860 GOTO 570
870 PRINT #1:"No more titles in data. ";TAB(30);"Program end."
880 DATA a,i,of,to,in,is,it,as,he,be,by,on,or,at
890 DATA my,an,me,so,if,no,we,up,do,us,the,and
900 DATA for,was,his,not,but,you,are,her,had,all
910 DATA seh,has,his,one,who,may,its,out,our,man
920 DATA now,any,can,old,new,own,yet,that,with
930 DATA have,from,this,they,were,been,will,when
940 DATA what,your,more,them,some,than,upon,into
950 DATA like,such,only,then,made,well,must,said
960 DATA time,even,very,much,most,which,their
970 DATA there,would,these,shall,great,other,about
980 DATA those,could,might,first,after,should
990 DATA "the theory of parsing and compiling"
1000 DATA "so what if they do multiply"
1010 DATA "antipasto teleostasy"
1020 DATA "modern aramaic is greek to me"
1030 DATA "computer i/o or any port in a storm"
1040 DATA "the impact of palanquinism on trobrianders"
9999 CLOSE #1 :: END

```

```

theory of parsing and compiling//the
parsing and compiling//the theory of
compiling//the theory of parsing and
multiply//so what if they do
antipasto teleostasy//
teleostasy//antipasto
modern aramaic is greek to me//
aramaic is greek to me//modern
greek to me//modern aramaic is
computer i/o or any port in a storm//
i/o or any port in a storm//computer
port in a storm//computer i/o or any
storm//computer i/o or any port in a
impact of palanquinism on trobrianders//the
palanquinism on trobrianders//the impact of
trobrianders//the impact of palanquinism on
No more titles in data.      Program end.

```


Text Analysis

The last program in this chapter performs text analysis. The program's input is raw text, for example two or three paragraphs from *Moby Dick*. Its output consists of a histogram of word frequency, the mean, mode, median, standard deviation, and skewness of the word lengths. With such statistics, a linguistic analyst would have a start at determining whether a paragraph had been written by one author or another.

The output we have chosen to show you demonstrates analyses of good authorship from Edgar Lee Masters and John Irving. We have tested this program on John Milton's "Paradise Lost" and a play by Voltaire in French. It is noteworthy that the statistics on these last two writers is indistinguishable from those of Masters and Irving. We suggest that you try some Chaucer in its original old English, and perhaps some German, just to get a feel for the work this program does.

```
10 REM filename: "wordfreq"
20 REM purpose: Word frequency analysis
30 REM author: jpg & jdr 10/82
40 REM -----
50 DIM T$(100),F(25):: OPEN #1:"RS232"
60 CALL CLEAR
70 PRINT "WORD FREQUENCY GRAPHING"
80 PRINT
90 PRINT "1 Input text"
100 PRINT "2 Store text on disk"
110 PRINT "3 Load text from disk"
120 PRINT "4 Tally frequencies"
130 PRINT "5 Display text on screen or printer"
140 PRINT "6 Statistics with histogram"
150 INPUT "Which activity ":K
160 DN K GOSUB 1000,2000,3000,4000,5000,6000
170 GOTO 60
1000 !***** Text input routine -- load T$ array. *****
1010 CALL CLEAR
1020 PRINT "Type up to 100 lines of text, on line at a time."
1030 PRINT "Each line must be less than two screen lines long."
1040 PRINT "Type '$$end' to signal end of input."
1050 FOR I=1 TO 100 :: PRINT I;
1060 LINPUT T$(I)
1070 IF T$(I)="$end" THEN N=I :: RETURN
1080 NEXT I
1090 PRINT "Last count of 100 exceeded."
1100 PRINT "Last line will be '$end'."
1110 T$(I)="$end" :: N=100 :: RETURN
2000 !***** Text storage routine -- dump T$ to file.****
2010 CALL CLEAR
2020 INPUT "What is file name for text storage ":F$ :: F$="DSK1."&F$
2030 OPEN #2:F$,INTERNAL,OUTPUT,VARIABLE
2040 FOR I=1 TO N
2050 PRINT #2:T$(I)
2060 NEXT I
2070 CLOSE #2 :: RETURN
3000 !***** Load text from disk into t$ *****
3010 CALL CLEAR
3020 INPUT "What is file name ":F$ :: F$="DSK1."&F$
3030 OPEN #3:F$,SEQUENTIAL,INTERNAL,INPUT ,VARIABLE
3040 FOR I=1 TO 100
3050 INPUT #3:T$(I)
3060 IF T$(I)="$end" THEN N=I :: CLOSE #3 :: RETURN
3070 NEXT I
3080 CLOSE #3 :: RETURN
```

```

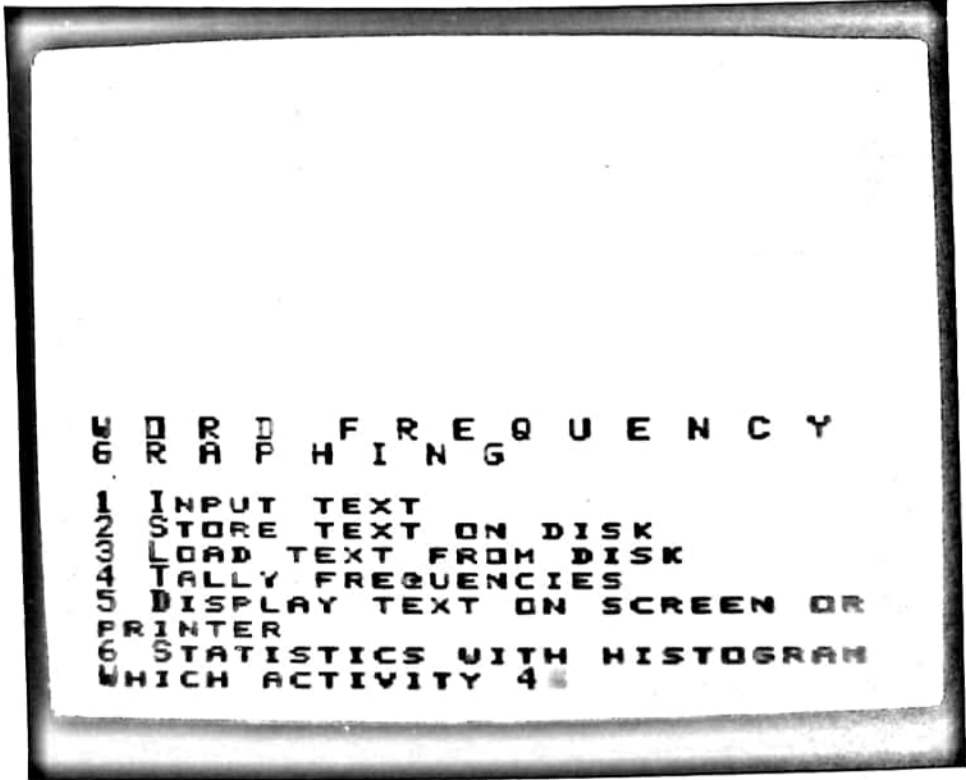
4000 !***** Tally Frequencies.*****
4010 CALL CLEAR
4020 SC=1
4030 GOSUB 7000
4040 FOR I=1 TO 25 :: F(I)=0 :: NEXT I
4050 FOR I=1 TO N-1 :: X#=T$(I)&" " :: L=LEN(X#):: Q=1
4060 P=POS(X#," ",Q)
4070 IF P=0 THEN 4120
4080 LW=P-Q :: F(LW)=F(LW)+1 :: J=F(LW)
4090 IF LW=0 THEN 4110 !found 2 blanks in a row.
4100 IF J<21 THEN DISPLAY AT(21-J,LW*2+3):CHR$(30);ELSE DISPLAY AT(21-20,LW*2+3)
:CHR$(30);
4110 Q=P+1 :: GOTO 4060
4120 NEXT I
4130 DISPLAY AT(23,19):"<<ENTER>>";: INPUT A# :: RETURN
5000 !***** Display text routine. *****
5010 INPUT "1=screen only 2=printer too":P
5020 CALL CLEAR
5030 FOR I=1 TO N
5040 PRINT I;TAB(5);T$(I)
5050 IF P=2 THEN PRINT #1:I;TAB(5);T$(I)
5060 NEXT I
5070 INPUT "<<ENTER>>":A# :: RETURN
6000 !***** Statistics wit histogram. *****
6010 S=0 :: S2=0 :: T=0 :: MD=1 :: T2=0
6020 FOR I=1 TO 25
6030 S=S+I*F(I)
6040 S2=S2+I*I*F(I)
6050 T=T+F(I)
6060 IF F(MD)<F(I) THEN MD=I
6070 NEXT I
6080 FOR I=1 TO 25
6090 T2=T2+F(I)
6100 IF T2>T/2 THEN 6120
6110 NEXT I
6120 MD=I
6130 CALL CLEAR
6140 SC=INT(F(MD)/10)+1 !Scaling factor based on mode.
6150 GOSUB 7000
6160 FOR I=1 TO 25
6170 IF F(I)=0 THEN 6200
6180 P=INT(F(I)/SC+.5)
6190 FOR J=1 TO P :: DISPLAY AT(21-J,I*2+3):CHR$(30);: NEXT J
6200 NEXT I
6210 M=S/T :: V=(S2-S*M)/(T-1)
6219 IMAGE ##.##
6220 DISPLAY AT(1,16):"Mode =";: DISPLAY AT(1,23):USING 6219:MD;
6230 DISPLAY AT(2,16):"Median=";: DISPLAY AT(2,23):USING 6219:MD;
6240 DISPLAY AT(3,16):"Mean =";: DISPLAY AT(3,23):USING 6219:M;
6250 DISPLAY AT(4,16):"St.Dv.=";: DISPLAY AT(4,23):USING 6219:SQR(V);
6260 DISPLAY AT(5,16):"Skew =";: DISPLAY AT(5,23):USING 6219:3*(M-MD)/SQR(V);
6270 DISPLAY AT(6,16):"Filem=";: DISPLAY AT(7,17):F#;
6280 INPUT "<<ENTER>>":A#
6290 RETURN
7000 !***** Subroutine to set up screen outline. ****
7010 CALL CLEAR
7020 DISPLAY AT(22,4):" 1 2 3 4 5 6 7 8 9 0 1 2"
7030 DISPLAY AT(23,10):"Word Size"
7040 FOR I=1 TO 9
7050 DISPLAY AT(I+8,1):SEG$("FREQUENCY",I,1)
7060 NEXT I
7070 FOR I=0 TO 13
7080 DISPLAY AT(I+7,2):(14-I)*SC;
7090 NEXT I
7100 RETURN
9999 END

```

WORD FREQUENCY GRAPHING

- 1 Input text
 - 2 Store text on disk
 - 3 Load text from disk
 - 4 Tally frequencies
 - 5 Display text on screen or printer
 - 6 Statistics with histogram
- Which activity

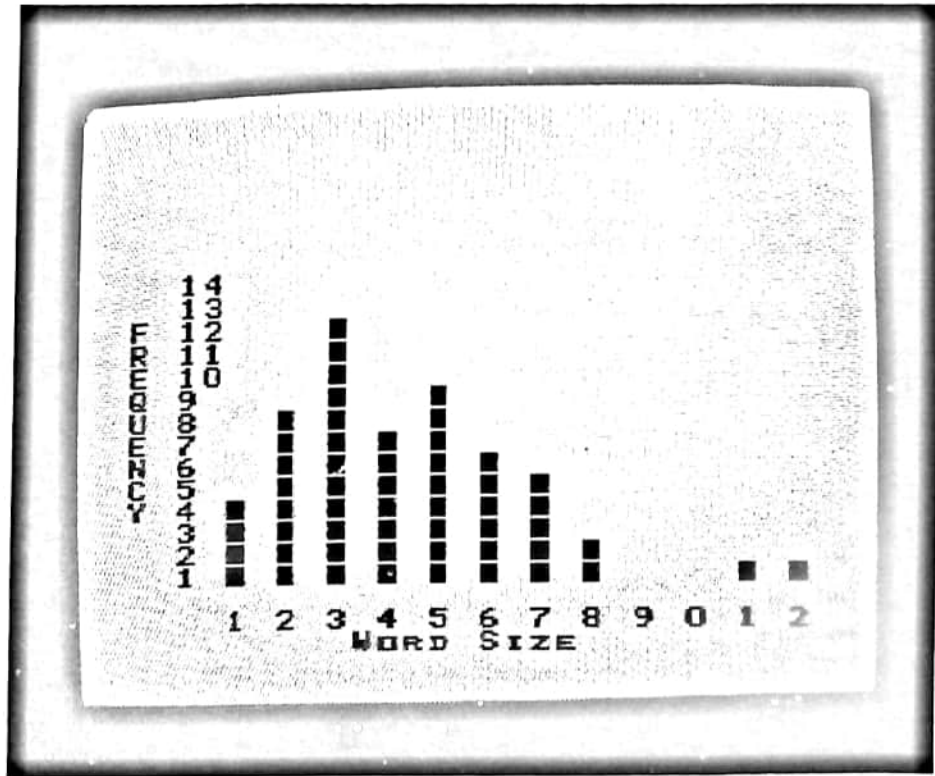
- 5
- 1 Out of a cell into this darkened space-
- 2 The end at twenty five!
- 3 My tongue could not speak what stirred within me,
- 4 And the village thought me a fool.
- 5 Yet at the start there was a clear vision,
- 6 A high and urgent ourpose inmy soul
- 7 Which drove me on trying to memorize
- 8 The encyclopedia Britannica!
- 9 \$\$end

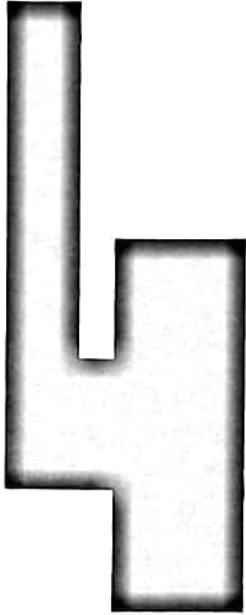


WORD FREQUENCY
GRAPHING

- 1 INPUT TEXT
- 2 STORE TEXT ON DISK
- 3 LOAD TEXT FROM DISK
- 4 TALLY FREQUENCIES
- 5 DISPLAY TEXT ON SCREEN OR
PRINTER
- 6 STATISTICS WITH HISTOGRAM

WHICH ACTIVITY 4





Linear and Linked Lists

The data management techniques that deal with arrays require the use of pointers in the form of subscripts to the arrays. The subscripts are freely variable in value, from 1 to N where N is the size of the array. In BASIC, the range of the subscript is even greater, because all arrays are defined automatically with a 0th element, whether it is used or not.

When a program uses a subscript, its value can skip anywhere within its range, forward or backward. However, there are data management techniques that use arrays or lists whose pointers are highly restricted in value. Such data structures are called *stacks*, *queues*, and *deques*.

Stacks

A stack is a list of data elements in which addition or deletion of an element occurs only at one end. The stack addition operation is called a *push* and its deletion operation is called a *pop*. The end of the stack at which a push or pop can occur is called its *top*, and its opposite end is called its *bottom*. The top of a stack is its most accessible end, while its bottom is its least accessible. The sketch Figure 4.1 indicates these essentials.

A stack has a single pointer, which represents the position of the top element relative to the bottom element. Thus a stack of four elements has a *stack pointer* with value 4. A push operation increases the value of the pointer to 5. A succession of pop operations can reduce the pointer to 0, but no further, else a stack underflow would occur. When the pointer is 0, the stack is empty.

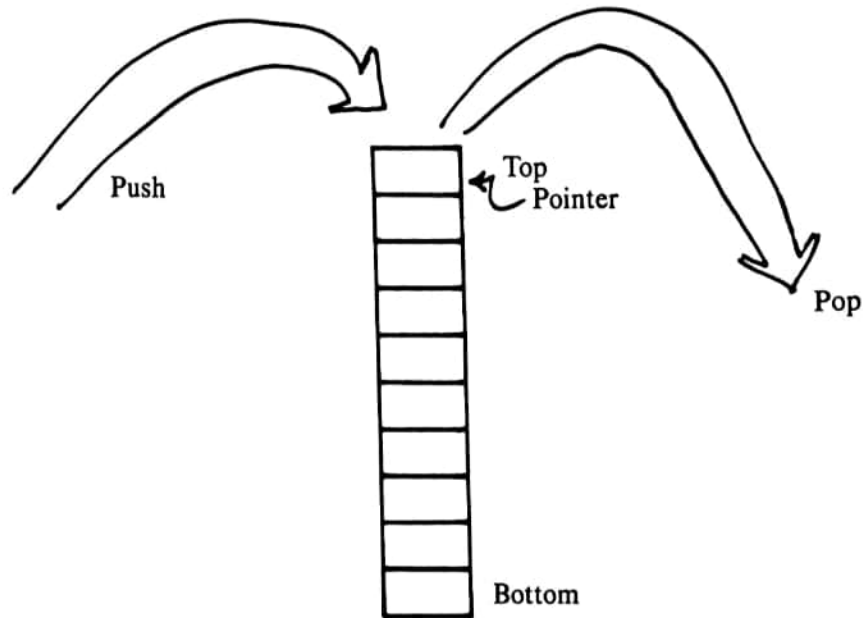


Figure 4.1 Stack

Suppose you want a stack called S. All that you need to do in BASIC to create that stack is to dimension a list. If you expect that the maximum size of the stack will never exceed 100, then you would say DIM S(100) to set up the stack. You must initialize its top pointer to 0 with an instruction such as P=0. Then the push and pop operations can be performed with the two algorithms that follow. They are written as subroutines, using P as the top pointer, S as the stack, and X and Y as values that are either pushed onto or popped from the stack.

Push Algorithm:

```

1000 '***** Subroutine to push element onto stack
1010 'Check for overflow OVER
1020 IF P>=N THEN PRINT "Underflow" :: RETURN
1030 P=P+1 'Increment top pointer
1040 S(P)=X 'Place X onto the stack
1050 RETURN

```

Pop Algorithm:

```

1000 '***** Subroutine to pop element Y from stack
1010 'Check for underflow
1020 IF P<=0 THEN PRINT "Underflow" :: RETURN
1030 Y=S(P)'Unstack, or pop, element into Y
1040 P=P-1 'Decrement pointer
1050 RETURN

```

Stacks are useful when you have to keep track of return addresses, or array elements that need to be processed later and in order. Two data management techniques that use stacks extensively have been mentioned already in this book. The Quicksort uses a stack to maintain

a list of pushed elements, and the binary tree structure exemplified by the tree sort in Chapter 2 uses a stack to keep track of its return addresses during sorted order traversal. You will see this stack mechanism again in subsequent chapters, particularly in Chapters 7 and 8, during the discussion of the Binary Sequence Search Tree, or BSST.

Queues

There is a list structure which permits the addition of elements at one end and their deletion at the other. This structure is called a queue. Whereas a stack is a *LIFO* (Last In, First Out) list, the queue is a *FIFO* (First In, First Out) list. These two ways to look at lists tend to determine more than anything else their application in computer programs.

Suppose you want to simulate a customer waiting line in a supermarket or at a drive-in window; you need some kind of data structure that lets the first person in line be the first one served, and conversely the last one in line should be the last one served. The queue structure has two pointers. The *front pointer* is the one at which deletions occur (the customer is served), and the *rear pointer* points to the queue element just in front of the place where additions occur (it's the end of the line).

Pictorially, you may think of a queue as a list of elements that looks like Figure 4.2.

Suppose you have created a queue of 100 possible positions with the statement `DIM Q(100)`. Initially, it has no elements in it, and the front (F) and rear (R) pointers are 0.

Add to Queue Algorithm:

```

1000 '***** Subroutine to add Y to queue
1010 'Check for overflow
1020 IF R:=100 THEN PRINT "Overflow" :: RETURN
1030 R=R+1 'Increment rear pointer
1040 Q(R)=Y 'Add element to queue
1050 'Adjust front pointer if necessary
1060 IF F=0 THEN F=1
1070 RETURN

```

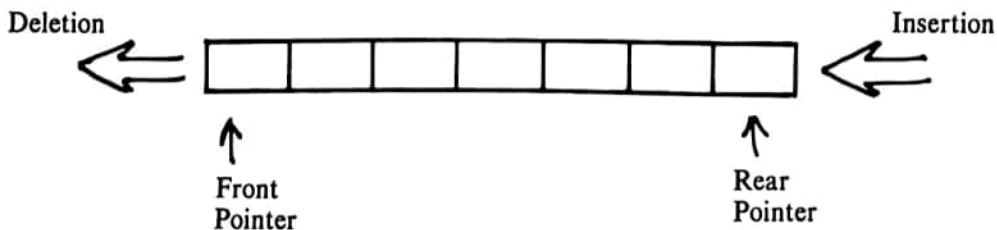


Figure 4.2 Queue

Delete from Queue Algorithm:

```
1000 '***** Subroutine to delete X from queue
1010 'Check for underflow
1020 IF F=0 THEN PRINT "Underflow" :: RETURN
1030 X=@(F)'Delete element
1040 IF F=R THEN F=0 :: RETURN 'Empty queue
1050 F=F+1 'Move front pointer to next element
1060 RETURN
```

The problem with this type of queue structure is that as you process the queue, the rear pointer keeps going higher and higher, regardless of how short your line is. One way to avoid this excessive use of memory is to make your queue circular, such that Q(1) follows Q(N). See Figure 4.3.

Add to Circular Queue Algorithm:

```
1000 '***** Subroutine to add Y to circular queue
1010 'Reset rear pointer if necessary
1020 IF R=N THEN R=1 ELSE R=R+1
1030 'Check for overflow
1040 IF F=R THEN PRINT "Overflow" :: RETURN
1050 @(R)=Y 'Insert element at rear
1060 IF F=0 THEN F=1 'Set front pointer if needed
1070 RETURN
```

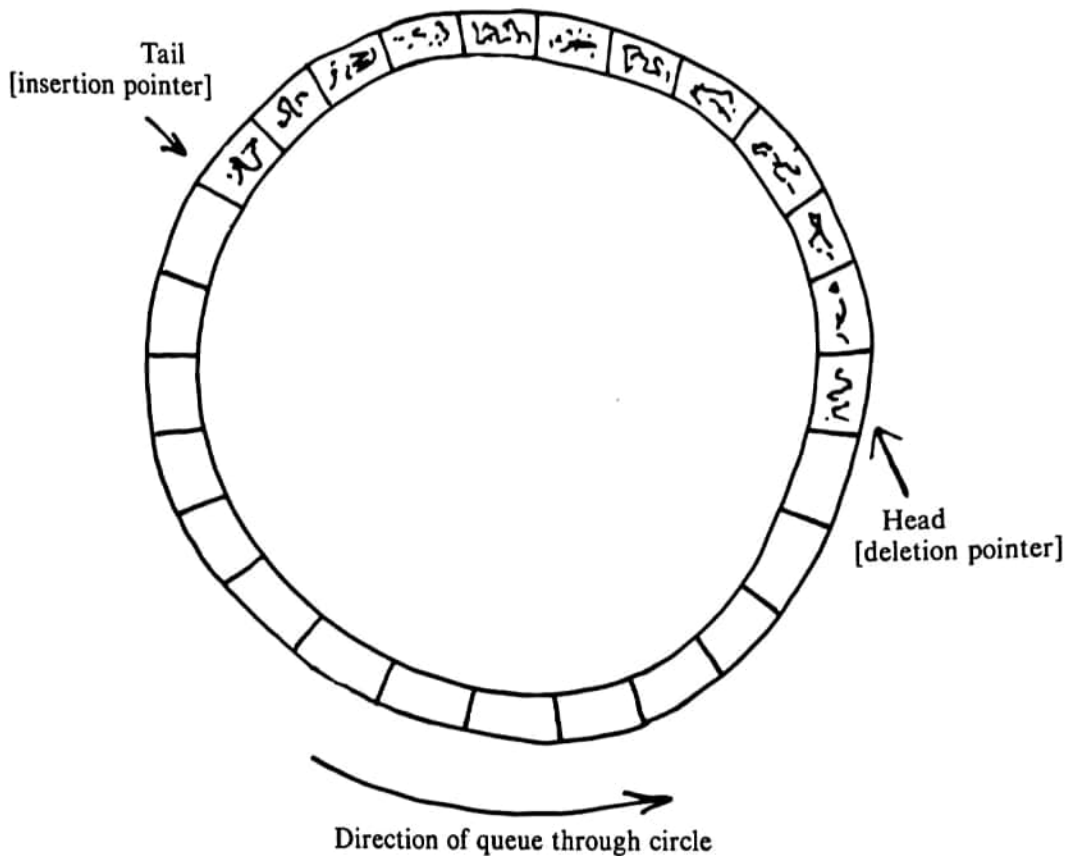


Figure 4.3 Circular Queue

Delete from Circular Queue Algorithm:

```
1000 !***** Subroutine to delete from circular queue
1010 !Check for underflow
1020 IF F=0 THEN PRINT "Underflow" :: RETURN
1030 Y=Q(F)!Remove front element
1040 IF F=R THEN F=0 :: RETURN !Empty queue
1050 !Increment front pointer or reset to 1
1060 IF F=N THEN F=1 ELSE F=F+1
1070 RETURN
```

Dequeues

Probably the most flexible type of linear list is the *double-ended queue*, often called deque. This type of structure is illustrated in Figure 4.4.

In this type of linear list, you arbitrarily call one end the front and the other end the rear, since insertions and deletions can occur at either end. This type of structure, like the circular queue, is generally less wasteful of space than the queue because it is not dimensioned much beyond its maximum size.

There are four required algorithms to manage a deque: two are used for insertion, and the other two are used for deletion.

Insert at Front of Deque Algorithm:

```
1000 !***** Subroutine to insert Y at front of deque
1010 !First, set pointer to where insertion will occur
1020 IF F=1 THEN T=N ELSE T=T-1
1030 !Then check for underflow
1040 IF T=R THEN PRINT "Underflow" :: RETURN
1050 X(F)=Y !insert element at front
1060 F=T !Reset front pointer
1070 RETURN
```

Insert at Rear of Deque Algorithm:

```
1000 !***** Subroutine to insert Y at rear of deque
1010 !Set pointer first
1020 IF R=N THEN T=1 ELSE T=R+1
1030 !Then check for underflow
1040 IF T=F THEN PRINT "Underflow" :: RETURN
1050 R=T !Reset rear pointer
1060 X(R)=Y !Insert element at rear
1070 RETURN
```

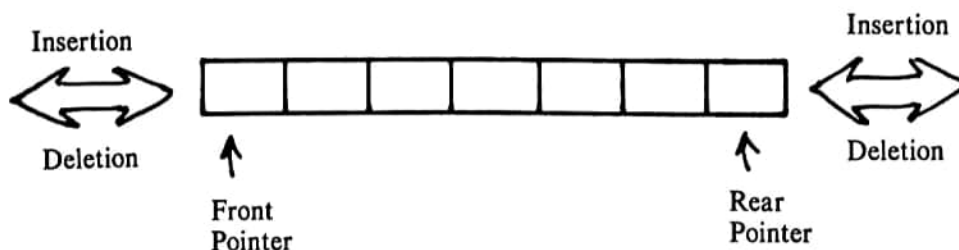


Figure 4.4 Double-ended Queue or Deque

Delete at Front of Deque Algorithm:

```
1000 !***** Subroutine to delete Y from deque front
1010 !Check for underflow first
1020 IF R=F THEN PRINT "Underflow" :: RETURN
1030 !Then reset front pointer
1040 IF F=N THEN F=1 ELSE F=F+1
1050 Y=X(F) !Last, remove front element
1060 RETURN
```

Delete at Rear of Deque Algorithm:

```
1000 !***** Subroutine to delete Y from deque rear
1010 !First, check for underflow
1020 IF R=F THEN PRINT "Underflow" :: RETURN
1030 Y=X(R) !Then remove from rear
1040 !Last, reset rear pointer
1050 IF R=1 THEN R=N ELSE R=R-1
1060 RETURN
```

The deque, the circular queue, and the stack generally enjoy use as minor pieces in much larger applications. Certain simulation programs in computer science, for example those that imitate the lineup of customers in timesharing operating systems, rely heavily on these algorithms. Stacks are used a lot in the compiler algorithms that produce machine language versions of algebraic expressions. They are also used in subroutine return address management and in recursive programming applications.

Linked Lists

Stacks, queues, and deques suffer as a result of one common failing: Only one element (top or head) or two elements (head and tail) can be accessed. Consider how difficult it would be to retrieve stacked data in sorted order if it is not stored in sorted order on the stack. This characteristic of all linear lists, which is inherent in their definition, limits their flexibility. Linked lists overcome the problems of linearity or consecutiveness of linear lists by providing extra pointers or links to the various elements in the linked list.

Singly Linked Lists

The simplest form of linked list structure is the one that has only one link per element. Suppose, for example, you wish to be able to access a list of data elements, say alphabetic letters, in sorted order, but without rearranging the data. What you want is a chain of data, with each record linked to the one higher in alphabetic order. You need a starting point, which is called the head pointer, or the head link. In simple English steps, here is the procedure using the set of data "B", "L", "E", "A", and "M".

1. Declare array space for both the data and its links. Call the data array *D* and the links *L*. Declare the head pointer *H* as a separate area, make it null (0). Declare the array pointer *E* (End-of-file, next available space for new data) and set the value of *E* to 1.

Array Position	Data Element	Link	
1	----	----	Head Pointer — H = 0 Array Pointer — E = 0
2	----	----	
3	----	----	
4	----	----	
5	----	----	
6	----	----	

2. Get the next data element. Add 1 to the array pointer E.
3. Is the head pointer H null?
 - a. Yes, set H = 1, insert element X into D(H), go to 2.
 - b. No, declare pointer P = H, go to 4.
4. Is X less than D(H), the head data element?
 - a. Yes, set H = E, perform Add-end routine, go to 2.
 - b. No, go to 5.
5. Is X less than D(P), the data element P points to?
 - a. Yes, set L(P2) = E, perform Add-end routine, go to 2.
 - b. No, go to 6.
6. Is L(P) = 0?
 - a. Yes, set L(P) = E, place X into D(E), go to 2.
 - b. No, set P2 = P, P = L(P), go to 5.

Add-end routine: Set L(P) = E, D(E) = X, return

The effect of carrying through this algorithm on B, L, E, A, and M is this final configuration.

Array Position	Data Element	Link	
1	B	3	Head Pointer — H = 4 Array Pointer — E = 6
2	L	5	
3	E	2	
4	A	1	
5	M	0	
6	----	----	

Notice that the letters B, L, E, A, and M are not in alphabetic order. But if you start at element 4, the head pointer element, and fetch the array elements in the order specified by their links, the order is

A-(go to 1)-B-(go to 3)-E-(go to 2)-L-(go to 5)-M-(end).

The *physical* order of the elements, by array position, is B-L-E-A-M, but their *logical* order by links is A-B-E-L-M, and they can be displayed in this sorted order.

Doubly Linked Lists

An even more flexible data structure, the doubly linked list, allows access to a list in forward or reverse order. This requires a second link for each element and a tail pointer. Here is the previous example in

table form, only this time it is accessible both in ascending and descending alphabetic order.

Array Position	Data Element	Forward Link	Backward Link	
1	B	3	4	Head Pointer — H = 4
2	L	5	3	Tail Pointer — T = 5
3	E	2	1	Array Pointer — E = 6
4	A	1	0	
5	M	0	2	

Aside from its added flexibility for displaying the data, a doubly linked list allows its links to be rebuilt in case one of the two chains is broken.

Circularly Linked Lists

A minor alteration to the doubly linked list makes the entire structure a circular list in both directions. Consider the example above: If the forward link in array position 5 is changed from null (0) to 4, the last letter of the alphabet now points to the first, and the linking is circular. You don't really need a head pointer anymore, because you can access any element in the list, or any subset of the list in sorted order. If you change the backward link in array position 4 from null to 5, the first letter of the alphabet points to the last, again closing the circle, this time for backward access through the list.

Circular Doubly Linked List Application

The only program we have decided to include in this chapter is a fairly comprehensive example of a circularly arranged double ended linked list written by Celia Robertson. The program accepts names (S\$) from the user and searches the list in one direction. As soon as the point in the alphabetically arranged linked list is reached where the name should appear, the program inserts it there, altering links accordingly. This structure allows alphabetically ordered display of the list's contents as well as addition and deletion of elements.

```

10 REM filename: "linklist"
20 REM purpose: Circularly doubly linked list
30 REM author: jpg & jdr 10/82
40 REM -----
50 ! This program creates and maintains a circular
60 ! doubly linked list of names.
70 ! Accessing, deleting and inserting are all possible.
80 ! Entries are linked in alphabetical order.
90 CALL CLEAR
100 DIM C$(20),L(20),R(20)
110 I=0
120 H=I
130 !*****
140 !* C$ will hold names of linked list. *
150 !* L is the left links. *
160 !* R is the right links. *
170 !* I is subscript. *
180 !* H will designate the head. *
190 !*****
200 PRINT "Space has been created for list and links"
210 PRINT "To insert, type +, to delete, type -"
220 PRINT "To access, type ?, to display, type ."

```

```

230 PRINT "To end, type # "
240 INPUT Q$
250 IF Q$="+" THEN 400
260 IF Q$="-" THEN 930
270 IF Q$="?" THEN 1260
280 IF Q$="." THEN 1500
285 IF Q$="#" THEN 1560
290 PRINT "Invalid character."
300 GOTO 210
310 !*****
320 !****      Insertion      ****
330 !*****
340 !* This module creates the first entry and      *
350 !*   subsequent insertions. Type a name beginning *
360 !*   with an alphabetic character.             *
370 !*****
380 !*****
390 !*****
400 PRINT "Now you are ready to insert"
410 PRINT "Type a name starting with an alphabetic character."
420 INPUT S$
430 IF H=0 THEN I=I+1 :: C$(I)=S$ :: L(I)=I :: R(I)=I :: H=I :: GOTO 210
440 !*****
450 !* Above tests for empty list.                  *
460 !* If empty, name inserted and head (H) = 1.   *
470 !*****
480 IF R(H)=H THEN IF S%=C$(H) THEN PRINT "No duplicates" :: GOTO 210 ELSE I=I+1
:: C$(I)=S$ :: L(I)=H :: R(I)=H :: L(H)=I :: R(H)=I :: GOTO 210
490 !*****
500 !* Above checks for only one entry.            *
510 !* If so, insertion complete. If not, duplicate. *
520 !*****
530 K=H
540 IF S%<C$(K) THEN 600
550 IF S%>C$(K) THEN 690
560 !*****
570 !* K is set to H, then head.                    *
580 !* Name is compared to head, and then to others. *
590 !*****
600 IF S%<C$(K) THEN J=L(K) :: IF C$(J)>C$(K) THEN 740 ELSE K=L(K) :: GOTO 600
610 IF S%=C$(K) THEN PRINT "ERROR! No duplicates." :: PRINT "S%>";S%; " C%>";C$(K)
):: GOTO 210 ELSE 800
620 !*****
630 !* Above compares names with those less.       *
640 !* List exhausted to left when C% of right link > *
650 !*   current insertion made.                  *
660 !* Otherwise, when > than found, insertion done. *
670 !* Equals indicates duplicates, - not allowed *
680 !*****
690 IF S%>C$(K) THEN J=R(K) :: IF C$(J)<C$(K) THEN 800 ELSE K=R(K) :: GOTO 690
700 IF S%=C$(K) THEN PRINT "ERROR! No duplicates." :: PRINT "S%>";S%; " C%>";C$(K)
):: GOTO 210 ELSE 740
710 !*****
720 !* Same logic as above, only less than.        *
730 !*****
740 M=L(K) :: I=I+1 :: C$(I)=S$ :: L(I)=M :: R(I)=K
750 L(K)=I :: R(M)=I
760 GOTO 210
770 !*****
780 !* Above is routine when a greater one is found. *
790 !*****
800 M=R(K) :: I=I+1 :: C$(I)=S$ :: L(I)=K :: R(I)=M
810 R(K)=I :: L(M)=I
820 GOTO 210

```

```

830 !*****
840 !* Above is insertion routine when less one found. *
850 !*****
860 !*** Deletion ***
870 !*****
880 !* This module deletes from the circular list. *
890 !* Type name beginning with alphabetic character. *
900 !*****
910 !*****
920 !*****
930 PRINT "Now you are in the deletion routine."
940 PRINT "Type a name starting with an alphabetic character."
950 INPUT S$
960 IF H=0 THEN PRINT "List is empty, must add first." :: GOTO 210
970 IF R(H)=H AND S#=C$(H) THEN PRINT "Deletion made on only entry in list" :: H=
0 :: GOTO 210
980 !*****
990 !* Check is made for an empty or 1-element list. *
1000 !*****
1010 K=H
1020 IF S#<C$(K) THEN 1040
1030 IF S#>C$(K) THEN 1060
1040 IF S#<C$(K) THEN J=L(K):: IF C$(J)>=C$(K) THEN 1150 ELSE K=L(K):: GOTO 1040
1050 IF S#=C$(K) THEN 1080 ELSE PRINT "Links followed, ";S#;" can't be found." ::
GOTO 1500
1060 IF S#>C$(K) THEN J=R(K):: IF C$(J)<=C$(K) THEN 1150 ELSE K=R(K):: GOTO 1060
1070 IF S#=C$(K) THEN 1080 ELSE PRINT "Links followed, ";S#;" can't be found." ::
GOTO 1500
1080 N=L(K):: M=R(K):: L(M)=L(K):: R(N)=R(K)
1090 IF K=H THEN H=M
1100 GOTO 210
1110 !*****
1120 !* Above is delete routine, similar to insertion. *
1130 !*****
1140 !*****
1150 PRINT "List has been searched. ";S#;" cannot be found."
1160 GOTO 1500
1170 !*****
1180 !*** Accessing ***
1190 !*****
1200 !* This module accesses a name in the list. *
1210 !* The head determines the start of the search. *
1220 !* This is similar to insertion, also. *
1230 !*****
1240 !*****
1250 !*****
1260 PRINT "Accessing routine is available."
1270 PRINT "Enter a name starting with an alphabetic character."
1280 INPUT S$
1290 IF H=0 THEN PRINT "List empty. Accessing not possible." :: PRINT "You must
insert names first." :: GOTO 210
1300 K=H
1310 IF S#=C$(K) THEN PRINT S#;" has been found. It is the head." :: GOTO 1400
1320 IF S#<C$(K) THEN 1340
1330 IF S#>C$(K) THEN 1360
1340 IF S#<C$(K) THEN J=L(K):: IF C$(J)>=C$(K) THEN 1380 ELSE K=L(K):: GOTO 1340
1350 IF S#=C$(K) THEN 1400
1360 IF S#>C$(K) THEN J=R(K):: IF C$(J)<=C$(K) THEN 1380 ELSE K=R(K):: GOTO 1360
1370 IF S#=C$(K) THEN 1400
1380 PRINT "List searched. ";S#;" cannot be found."
1390 GOTO 210

```

```

1400 PRINT "S$>";S$;" C$>";C$(K)
1410 PRINT "Left link of C$ is";L(K)
1420 PRINT "Right link of C$ is";R(K)
1430 PRINT "If you wish to see the name of the logical left,"
1440 PRINT "type L. Type R to see the name to the right."
1450 PRINT "Or type # to return to the main menue."
1460 INPUT L$
1470 IF L$="L" THEN P=L(K):: PRINT "To the left lies ";C$(P):: GOTO 1430
1480 IF L$="R" THEN P=R(K):: PRINT "To the right lies ";C$(P):: GOTO 1430
1490 IF L$="#" THEN GOTO 210 ELSE PRINT "Illegal character." :: GOTO 1430
1500 PRINT "List shown below."
1510 K=H
1520 PRINT "C$>";K;C$(K)
1530 K=R(K)
1540 IF K=H THEN 210
1550 GOTO 1520
1560 END

```

```

Space has been created for list and links
To insert, type +, to delete, type -
To access, type ?, to display, type .
To end, type #
+
Now you are ready to insert
Type a name starting with an alphabetic character.
persimmon
To insert, type +, to delete, type -
To access, type ?, to display, type .
To end, type #
+
Now you are ready to insert
Type a name starting with an alphabetic character.
carrot
To insert, type +, to delete, type -
To access, type ?, to display, type .
To end, type #
+
Now you are ready to insert
Type a name starting with an alphabetic character.
prune
To insert, type +, to delete, type -
To access, type ?, to display, type .
To end, type #
+
Now you are ready to insert
Type a name starting with an alphabetic character.
celery
To insert, type +, to delete, type -
To access, type ?, to display, type .
To end, type #
+
Now you are ready to insert
Type a name starting with an alphabetic character.
cucumber
To insert, type +, to delete, type -
To access, type ?, to display, type .
To end, type #
+
Now you are ready to insert
Type a name starting with an alphabetic character.
pomegranate
To insert, type +, to delete, type -
To access, type ?, to display, type .
To end, type #

```

```

?
Accessing routine is available.
Enter a name starting with an alphabetic character.
Space has been created for list and links
To insert, type +, to delete, type -
To access, type ?, to display, type .
To end, type #
+
Now you are ready to insert
Type a name starting with an alphabetic character.
persiimmon
To insert, type +, to delete, type -
To access, type ?, to display, type .
To end, type #
+
Now you are ready to insert
Type a name starting with an alphabetic character.
carrot
To insert, type +, to delete, type -
To access, type ?, to display, type .
To end, type #
+
Now you are ready to insert
Type a name starting with an alphabetic character.
prune
To insert, type +, to delete, type -
To access, type ?, to display, type .
To end, type #
+
Now you are ready to insert
Type a name starting with an alphabetic character.
cucumber
To insert, type +, to delete, type -
To access, type ?, to display, type .
To end, type #
+
Now you are ready to insert
Type a name starting with an alphabetic character.
celery
To insert, type +, to delete, type -
To access, type ?, to display, type .
To end, type #
+
Now you are ready to insert
Type a name starting with an alphabetic character.
pomegranate
To insert, type +, to delete, type -
To access, type ?, to display, type .
To end, type #

?
Accessing routine is available.
Enter a name starting with an alphabetic character.
celery
S$>celery C$>celery
Left link of C$ is 2
Right link of C$ is 4
If you wish to see the name of the logical left,
type L. Type R to see the name to the right.
Or type # to return to the main menu.
L
To the left lies carrot
If you wish to see the name of the logical right,
type R. Type L to see the name to the left.
Or type # to return to the main menu.
R
To the right lies cucumber

```


If you wish to see the name of the logical left,
type L. Type R to see the name to the right.
Or type * to return to the main menu.

*
To insert, type +, to delete, type -
To access, type ?, to display, type .
To end, type #

List shown below.

C#> 1 persiimmon

C#> 6 pomegranate

C#> 3 prune

C#> 2 carrot

C#> 5 celery

C#> 4 cucumber

To insert, type +, to delete, type -

To access, type ?, to display, type .

To end, type #

#

S

Sequential Access Files

A program's utility is often measured by its ability to manage a large volume of information. Without the use of external files on tape or disk, a program can store data in only two structures: dimensioned arrays and DATA statements. Arrays suffer the major limitation of being temporary in nature, so that when you turn off the computer, whatever was stored in the array is irretrievably lost. DATA statements on the other hand are a permanent part of the program, but they are cumbersome to type in and cannot be altered during program execution. The solution to these problems is to store the data separate from but accessible to the program. Two different types of files exist to perform this task, *sequential access files* and *direct access files*.

Sequential access files may exist on either tape or disk. The programs that we include in this chapter will use disk files, although they would run with tape files just as well. Sequential files are characterized by the serial nature of their stored records. If a file is composed of seven records, a program must access the first six records in sequence before it can deal with the seventh. Sequential files are also distinguished by the fact that they can be OPENed in either INPUT mode or OUTPUT mode, but not both. This means that if you wish to alter a record, you must OPEN two files, the first one in INPUT mode to access the records, and the second one in OUTPUT mode to rebuild the file in revised form.

This chapter will explore the potential and the limitations of sequential access files as a permanent storage medium. We will discuss the access, modification, and sorting of this type of file.

Sequential Search Techniques

The procedure that a programmer chooses for searching a sequential file for a specific record depends on whether the file is ordered or not. In the case where the file is unordered, the procedure is simply to step through the file one record at a time, checking each record to see if it is the one sought. The search terminates upon reaching either one of two conditions: (1) The record is found, or (2) the entire file has been searched and the record is not on the file. This form of search, the *unordered list directed scan*, is shown in the algorithm below, written as a subroutine.

```
1000 '***** Subroutine for unordered list seq. scan
1010 !X is key sought on file
1020 !A is value retrieved from file
1030 !I is sequential record number
1040 I=1 !Set record pointer to 1
1050 INPUT #1:A 'Read this record
1060 IF X=A THEN PRINT X;" found on record #";I :: RETURN
1070 I=I+1 !Increment pointer
1080 IF I<=N THEN 1050 'N is the number of records
1090 PRINT "Search unsuccessful" :: RETURN
```

In the case where the file is ordered, the procedure is slightly more complex, but it yields the dividend that if the key being sought is not on the file, the entire file need not be searched. This is because the search terminates as soon as the key being sought is found to be less than the key of the record being checked on the file. Thus the search terminates upon reaching one of two conditions: (1) The record is found, or (2) the key, X, of the record being sought is less than the key, A, of the record being checked on the file. This form of search, the *ordered list directed scan*, is shown below.

```
1000 '***** Subroutine for ordered list seq. scan
1010 !X is key sought on file
1020 !A is value retrieved from file
1030 !I is sequential record number
1040 !A(1)<=a(2)<=a(3)...<=a(n)
1050 I=1 !Set record pointer to 1
1060 INPUT #1:A 'Read this record
1070 IF X<A THEN PRINT "Search unsuccessful" :: RETURN
1080 IF X=A THEN PRINT X;" found on record #";I :: RETURN
1090 I=I+1 !Increment pointer
1100 IF I<=N THEN 1060 ELSE PRINT "Search unsuccessful" :: RETURN
```

Sequential File
Access

The only way that a program can access a specific record on a file is to search the file one record at a time, beginning at the first record. Program SEQWORDS shows a simple application of this search technique. The program deals with a sequential file called WORDS, which contains the 100 most commonly found words in written English. Each record of the file contains two data items, the word itself and its frequency of occurrence in 242,432 words of English text taken from various authors and publishers. The list is the same one that was referenced in Chapter 3.

The program allows the user to perform any one of six activities: 1—build the file; 2—edit a file entry; 3—search the file for a specific word; 4—search the file for the frequency selected; 5—display statistics for all frequencies higher than the one sought; 6—display the file. This program could be used as a general utility for such word-processing applications as the KWIC index generator in Chapter 3. Notice that the program uses both kinds of sequential search techniques, since the file's records are not arranged in any order if the words themselves are being sought, but they are arranged in order of frequency of occurrence if that is the value being sought. See the file's structure below.

Sequential file WORDS.DAT

Record	Field 1 (W\$)	Field 2 (F)
1	the	15568
2	of	9767
3	and	7638
4	to	5739
5	a	5074
6	in	4312
.	.	.
.	.	.
.	.	.
100	two	244

```

10 REM filename: "seqwords"
20 REM purpose: small words manager
30 REM author: jpg & jdr 8/82
40 REM -----
60 CALL CLEAR
70 PRINT "Small Words Manager" :: PRINT
80 PRINT "1 Build file" :: PRINT "2 Edit file entry"
90 PRINT "3 Search file for specific word"
100 PRINT "4 Search file for specific frequencies"
110 PRINT "5 Display statistics for all higher frequencies"
120 PRINT "6 Display file"
130 PRINT "7 Stop"
140 INPUT "What activity":K
150 ON K GOSUB 1000,2000,3000,4000,5000,6000,9999
160 GOTO 60
1000 !***** Build file *****
1010 OPEN #1:"DSK1.WORDS",SEQUENTIAL,INTERNAL,OUTPUT
1020 FOR I=1 TO 100
1030 PRINT I;"Word, then frequency"
1040 INPUT W$,F
1050 PRINT #1:W$,F
1060 NEXT I
1070 CLOSE #1 :: RETURN
2000 !***** Edit file entry *****
2010 DIM WD$(100),FR(100)'Use arrays to hold word data
2020 OPEN #1:"DSK1.WORDS",SEQUENTIAL,INTERNAL,INPUT
2030 ' Fill the array with words
2040 I=1
2050 INPUT #1:WD$(I),FR(I)
2060 PRINT I;WD$(I);TAB(8);FR(I)
2070 I=I+1 :: IF EOF(1)<>1 THEN 2050
2080 INPUT "What record number (0=return)":I
2090 IF I=0 THEN GOSUB 2500 :: RETURN
2100 PRINT WD$(I),FR(I);"are at position#";I
2110 PRINT "New word, frequency";
2120 INPUT WD$(I),FR(I)
2130 PRINT WD$(I);FR(I);"now at position#";I
2140 GOTO 2080
2500 !***** Closing routine *****
2510 CLOSE #1 :: OPEN #1:"DSK1.WORDS",SEQUENTIAL,INTERNAL,OUTPUT
2520 I=1
2530 IF WD$(I)="" THEN 2550
2540 PRINT #1:WD$(I),FR(I):: I=I+1 :: GOTO 2530
2550 CLOSE #1 :: RETURN
3000 !***** Search file for specific word *****
3010 ' Directed scan, unordered list
3020 OPEN #1:"DSK1.WORDS",SEQUENTIAL,INTERNAL,INPUT
3030 I=1 :: INPUT "What word is sought('end'=return)":X$
3040 IF X$="end" THEN CLOSE #1 :: RETURN
3050 IF EOF(1) THEN 3100
3060 INPUT #1:W$,F
3070 IF X$=W$ THEN PRINT "Found at rec.#";I :: CLOSE #1 :: GOTO 3020
3080 I=I+1
3090 GOTO 3050
3100 PRINT "No such word on file." :: CLOSE #1 :: GOTO 3020
4000 !***** Search file for specific frequency ***
4010 ! Directed scan, ordered list
4020 S=0 :: OPEN #1:"DSK1.WORDS",SEQUENTIAL,INTERNAL,INPUT
4030 I=1 :: INPUT "What word freq. sought (0=return)":X
4040 IF X=0 THEN CLOSE #1 :: RETURN
4050 INPUT #1:W$,F
4060 IF X>F THEN PRINT "No such freq." :: CLOSE #1 :: GOTO 4020
4070 IF X=F THEN PRINT X;"found on rec.#";I :: CLOSE #1 :: GOTO 4020
4080 I=I+1
4090 IF EOF(1)<>1 THEN GOTO 4050 ELSE PRINT "No such freq." :: CLOSE #1 :: GOTO
4020

```

```

5000 !***** Higher frequency statistics *****
5010 OPEN #1:"DSK1.WORDS",SEQUENTIAL,INTERNAL,INPUT
5020 INPUT "Statistics on all freq. higher than (0=return)":X
5030 IF X=0 THEN CLOSE #1 :: RETURN
5040 S=0 :: S2=0
5050 FOR I=1 TO 100
5060 INPUT #1:W$,F
5070 IF F<X THEN 5100
5080 S=S+F :: S2=S2+F*F
5090 NEXT I
5100 N=I-1
5110 PRINT "Sum=";S :: PRINT "Sum of squares=";S2
5120 PRINT "Mean=";S/N
5130 PRINT "Std.Dev.=";SQR((S2-S*S/N)/(N-1))
5140 CLOSE #1 :: GOTO 5010
6000 !***** Display all records *****
6010 OPEN #1:"DSK1.WORDS",SEQUENTIAL,INTERNAL,INPUT
6020 FOR I=1 TO 100
6030 INPUT #1:W$,F
6040 PRINT W$;TAB(8);F
6050 IF EOF(1)THEN 6070
6060 NEXT I
6070 CLOSE #1
6080 FOR I=1 TO 2000 :: NEXT I
6090 RETURN
9999 END

```

Small Words Manager

- 1 Build file
- 2 Edit file entry
- 3 Search file for specific word
- 4 Search file for specific frequencies
- 5 Display statistics for all higher frequencies
- 6 Display file
- 7 Stop

What activity

6	
the	15568
of	9767
and	7638
to	5739
a	5074
in	4312
that	3017
is	2509
i	2292
it	2255
for	1869
as	1853
with	1849
was	1761
his	1732
he	1727
be	1535
not	1496
by	1392
but	1379
have	1344
you	1336
which	1291
are	1222
on	1155

Small Words Manager

```
1 Build file
2 Edit file entry
3 Search file for specific word
4 Search file for specific frequencies
5 Display statistics for all higher frequencies
6 Display file
7 Stop
What activity
3
What word is sought('end'=return)
for
Found at rec.# 11
What word is sought('end'=return)
on
Found at rec.# 25
What word is sought('end'=return)
purple
No such word on file.
What word is sought('end'=return)
was
Found at rec.# 14
What word is sought('end'=return)
end
```

Small Words Manager

```
1 Build file
2 Edit file entry
3 Search file for specific word
4 Search file for specific frequencies
5 Display statistics for all higher frequencies
6 Display file
7 Stop
What activity
4
What word freq. sought (0=return)
1849
1849 found on rec.# 13
What word freq. sought (0=return)
1291
1291 found on rec.# 23
What word freq. sought (0=return)
1942
No such freq.
What word freq. sought (0=return)
```

Group Totals

The second application shows how sequential file operations could be used in, say, a manufacturing environment. The data file consists of an unknown number of records, each representing a specific operation that was performed during the manufacture of various finished goods. The record format is as follows:

Field	Type	Description
Name	string	Worker's name
Department	numeric	Department, 2 digits
Job	string	Job name, one word
Hourly	numeric	Wage, no decimal point
Hours	numeric	Hours, with decimal point

The file is organized so that workers in the same department are grouped together. The following series of records is typical:

Name	Dep't.	Operation	Wage	Hours
Billitatings D	12	flushing	725	3.0
Billitatings D	12	flanging	615	7.5
Giant JG	12	fluming	685	4.5
Giant JG	12	flashing	785	4.0
Pressed D	27	fluting	580	2.0
Willikers G	27	flanking	600	3.0
Willikers G	27	floating	585	2.5
Bolism M	48	flensing	555	3.5

The problem is to process the file as follows: Read all the records and accumulate the total cost, by department. Produce a report grouped by department that lists all workers and the amount charged to that department. For example:

```
      Department 12
Billitatings D      $.....
Giant JG            $.....
-----
      Total          $.....
```

```
      Department 27
Pressed D           $.....
Presstrain X        $.....
Willikers G         $.....
-----
      Total          $.....
```



```

10 REM filename: "gptot"
20 REM purpose: Group totals from sequential file
30 REM author: jpg & jdr 10/82
40 REM -----
45 OPEN #9:"RS232"
50 CALL CLEAR
60 PRINT "Monthly Jobs File Processing"
70 PRINT "1 Build file"
80 PRINT "2 Print file"
90 PRINT "3 Process file"
100 PRINT "4 Build daily transaction file"
110 PRINT "5 Merge daily file with master file"
120 PRINT "6 Stop"
130 INPUT "What activity":K
140 ON K GOSUB 1000,2000,3000,4000,5000,9999
150 GOTO 50
1000 !***** Build file *****
1010 OPEN #1:"DSK1.MJOBS",SEQUENTIAL,INTERNAL,OUTPUT
1020 INPUT "Department (0=same, 99=return)":D1
1030 IF D1=99 THEN CLOSE #1 :: RETURN
1040 IF D1<>0 THEN D=D1
1050 INPUT "Name ('s'=same)":N1$
1060 IF N1$<>"s" THEN N$=N1$
1070 INPUT "Operation":J$
1080 INPUT "Hourly wage":W
1090 INPUT "Hours worked":H
1100 PRINT #1:N$,D,J$,W,H
1110 GOTO 1020
2000 !***** Print file *****
2010 INPUT "File name":F$ :: F$="DSK1."&F$
2020 OPEN #1:F$,SEQUENTIAL,INTERNAL,INPUT
2030 PRINT #9:"File name";F$
2040 PRINT #9:"Name";TAB(20);"Dept";TAB(28);"Job";
2050 PRINT #9:TAB(35);"Hourly";TAB(48);"Hours" :: PRINT #9
2060 IF EOF(1)THEN CLOSE #1 :: RETURN
2070 INPUT #1:N$,D,J$,W,H
2080 PRINT #9:TAB(2);N$;TAB(20);D;TAB(25);J$;TAB(35);
2090 PRINT #9,USING "$#.##":W/100;
2091 PRINT #9:TAB(50);
2100 PRINT #9,USING "#.#":H
2110 GOTO 2060
3000 !***** Process file *****
3010 INPUT "File name":F$ :: F$="DSK1."&F$
3020 OPEN #1:F$,SEQUENTIAL,INTERNAL,INPUT
3030 PRINT #9:TAB(10);"Monthly Report" :: PRINT #9 :: PRINT #9
3040 PRINT #9:"Name";TAB(30);"Hours";
3050 PRINT #9:TAB(40);"Wages";TAB(50);"Dept"
3060 PRINT #9
3070 N$="" :: D=0
3080 ! Check for end of file.
3090 IF EOF(1)THEN GOSUB 3600 :: GOSUB 3500 :: RETURN
3100 INPUT #1:N1$,D1,J1$,W1,H1
3110 IF N1$<>N$ THEN GOSUB 3600
3120 N$=N1$
3130 ! Check for change in department.
3140 IF D1<>D THEN GOSUB 3700
3150 D=D1
3160 SH=SH+H1
3170 W3=W1/100*H1
3180 SW=SW+W3
3190 H2=H2+H1 :: W2=W2+W3 :: HD=HD+H1 :: WD=WD+W3
3200 GOTO 3090
3500 !***** Close up, print bottom line.

```

```

3510 PRINT #9:"=====
3520 A$="Grand Total" :: H9=H2 :: W9=W2 :: D9=99
3530 GOSUB 3800
3540 CLOSE #1 :: RETURN
3600 !***** Print line with worker.
3610 !      Check for first worker.
3620 IF N$="" THEN RETURN
3630 A$=N$ :: H9=SH :: W9=SW :: D9=D
3640 GOSUB 3800
3650 SH=0 :: SW=0
3660 RETURN
3700 !***** Print department subtotal.
3710 !      Check for first department.
3720 IF D=0 THEN RETURN
3730 PRINT #9:"-----"
3740 A$="      Subtotal" :: H9=HD :: W9=WD :: D9=D
3750 GOSUB 3800
3760 HD=0 :: WD=0 :: PRINT #9
3770 RETURN
3800 !***** Print detai line
3810 PRINT #9:A$;TAB(30);
3820 PRINT #9,USING "###.#":H9;
3821 PRINT #9:TAB(40);
3830 PRINT #9,USING "$###.#":W9;
3831 PRINT #9:TAB(50);
3840 PRINT #9:D9
3850 RETURN
4000 !***** Build daily transaction file *****
4010 PRINT "Name this transaction file:"
4020 PRINT "Suggest MONJOBS or TUEJOBS"
4030 INPUT "Filename":F$ :: F$="DSK1."&F$
4040 OPEN #1:F$,SEQUENTIAL,INTERNAL,OUTPUT
4050 GOSUB 1020 ! Use same routine as before.
4060 RETURN
5000 !*** Subroutine GPTOTSUB Merge sequential files **
5010 INPUT "What is old master file name":F1$ :: F1$="DSK1."&F1$
5020 INPUT "What is daily transactionfile name":F2$ :: F2$="DSK1."&F2$
5030 INPUT "What is new master file name":F3$ :: F3$="DSK1."&F3$
5040 E1=0 :: E2=0 ! End-of-file flag -- 0=not empty
5050 OPEN #1:F1$,SEQUENTIAL,INTERNAL,INPUT
5060 OPEN #2:F2$,SEQUENTIAL,INTERNAL,INPUT
5070 OPEN #3:F3$,SEQUENTIAL,INTERNAL,OUTPUT
5080 INPUT #1:N1$,D1,J1$,W1,H1
5090 INPUT #2:N2$,D2,J2$,W2,H2
5100 IF D1<=D2 AND E1=0 THEN GOSUB 5500 :: GOTO 5100
5110 IF D2<D1 AND E2=0 THEN GOSUB 5600:GOTO 5110
5120 IF E1=0 THEN 5100
5130 IF E2=0 THEN GOSUB 5600 :: GOTO 5130
5140 CLOSE #1 :: CLOSE #2 :: CLOSE #3 :: RETURN
5500 !*** Transfer old master record
5510 PRINT #3:N1$,D1,J1$,,W1,H1
5520 IF N1$=N2$ THEN GOSUB 5600
5530 IF EOF(1)THEN E1=1 :: D1=99 :: RETURN
5540 INPUT #1:N1$,D1,J1$,W1,H1
5550 RETURN
5600 !*** Transfer transaction record
5610 PRINT #3:N2$,D2,J2$,W2,H2
5620 IF EOF(2)THEN E2=1 :: D2=99 :: RETURN
5630 INPUT #2:N2$,D2,J2$,W2,H2
5640 RETURN
9999 CLOSE #9 :: END

```

Monthly Report

Name	Hours	Wages	Dept
Billitating D	10.5	\$ 67.88	12
Giant JG	8.5	\$ 70.33	12

Subtotal	19.0	\$ 138.20	12
Pressed D	2.0	\$ 11.60	27
Willikers G	5.5	\$ 32.63	27

Subtotal	7.5	\$ 44.23	27
Bolism M	3.5	\$ 19.43	48
=====			
Grand Total	30.0	\$ 201.85	99

Sequential File Merging

A common processing technique used with sequential files is *merging*. This process implies the bringing together of two files into one, with the records of the second file meshing with those of the first file according to a particular scheme. Usually, one small sorted file, for example a daily transactions file, is merged with a much larger sorted file, e.g. a year-to-date transaction master file, to produce an up-to-date version of the larger file.

We will show this technique here using the same file as was used in the previous example. The problem could be stated this way: Suppose the file MJOBS represents this month's jobs performed by the workers in the plant. At the end of each day, the various jobs which each employee has performed are placed in sorted order on a file, say, MONJOBS, representing Monday's schedule. The two files are arranged in sorted order by department number. The merging subroutine in GPTOT adds the file MONJOBS to the file MJOBS to produce a new file MJOBS2.

Daily jobs file MONJOBS

Pressed D	27	flinging	610	2.0
Ementary L	41	frisking	700	3.0

New monthly jobs master file MJOBS2.DAT

Billitating D	12	flushing	725	3.0
Billitating D	12	flanging	615	7.5
Giant JG	12	fluming	685	4.5
Giant JG	12	flashing	785	4.0
Pressed D	27	fluting	580	2.0
Pressed D	27	flinging	610	2.0
Willikers G	27	flanking	600	3.0
Willikers G	27	floating	585	2.5
Ementary L	41	frisking	700	3.0
Bolism M	48	flensing	555	3.5

The program GPTOT contains the subroutine which has been isolated and identified as GPTOTSUB below, and it performs the merge operation described above. Notice that it must anticipate a variety of possible arrangements in the file DJOBS, such as departments whose numbers are higher, lower, or between those in the MJOBS file.

```

5000 !*** Subroutine GPTOTSUB Merge sequential files **
5010 INPUT "What is old master file name":F1$ :: F1$="DSK1."&F1$
5020 INPUT "What is daily transactionfile name":F2$ :: F2$="DSK1."&F2$
5030 INPUT "What is new master file name":F3$ :: F3$="DSK1."&F3$
5040 E1=0 :: E2=0 ' End-of-file flag -- 0=not empty
5050 OPEN #1:F1$,SEQUENTIAL,INTERNAL,INPUT
5060 OPEN #2:F2$,SEQUENTIAL,INTERNAL,INPUT
5070 OPEN #3:F3$,SEQUENTIAL,INTERNAL,OUTPUT
5080 INPUT #1:N1$,D1,J1$,W1,H1
5090 INPUT #2:N2$,D2,J2$,W2,H2
5100 IF D1<=D2 AND E1=0 THEN GOSUB 5500 :: GOTO 5100
5110 IF D2<D1 AND E2=0 THEN GOSUB 5600:GOTO 5110
5120 IF E1=0 THEN 5100
5130 IF E2=0 THEN GOSUB 5600 :: GOTO 5130
5140 CLOSE #1 :: CLOSE #2 :: CLOSE #3 :: RETURN
5500 !*** Transfer old master record
5510 PRINT #3:N1$,D1,J1$,W1,H1
5520 IF N1$=N2$ THEN GOSUB 5600
5530 IF EOF(1) THEN E1=1 :: D1=99 :: RETURN
5540 INPUT #1:N1$,D1,J1$,W1,H1
5550 RETURN
5600 !*** Transfer transaction record
5610 PRINT #3:N2$,D2,J2$,W2,H2
5620 IF EOF(2) THEN E2=1 :: D2=99 :: RETURN
5630 INPUT #2:N2$,D2,J2$,W2,H2
5640 RETURN

```

The merging process on two files can be generalized to allow any number of small sorted files to produce one large sorted file. Program SORTMERG uses this technique. It accesses three unsorted files, FLORA1, FLORA2, and FLORA3, and sorts each in turn. Then it performs a merge on all three to produce a fourth file called NEW. It so happens that this program was written in such a way that the size of the files being merged must be the same, but such is unfortunately not the case very often in real life. We leave this embellishment to the program's merging routine to the stout-hearted reader.

```

10 REM filename: "sortmerg"
20 REM purpose: Sort-merg of sequential files
30 REM author: jpg & jdr 10/82
40 REM -----
50 RANDOMIZE :: CALL CLEAR :: DIM N$(50),K2(3):: OPEN #9:"RS232"
60 INPUT "How many files ":NF
70 INPUT "How many entries will be in each file ":SZ
80 INPUT "File group name ":F$
90 PRINT "1=Shuffle 2=Create 3=Edit"
100 INPUT "4=Sort 5=Merg 6=Stop":Y
110 ON Y GOSUB 1000,2000,3000,4000,5000,9999 :: GOTO 90
120 !*****
130 !** If the number of files to be used is more **
140 !** than three, be sure to issue the commands: **
150 !** CALL FILES(number) **
160 !** NEW **
170 !** before loading your program.(hmc) **
180 !*****

```

```

1000 !***** Shuffle the entries in all of the files. **
1010 FOR W=1 TO NF :: FL$="DSK1."&F$&STR$(W)
1020 K=1 :: GOSUB 7000
1030 FOR J=1 TO SZ :: A=INT(RND*SZ)+1 :: B=INT(RND*SZ)+1
1040 T$=N$(A):: N$(A)=N$(B):: N$(B)=T$ :: NEXT J
1050 PRINT #9:"File ";FL$;" in original order."
1060 GOSUB 8000 :: N=SZ :: GOSUB 9000 :: NEXT W :: RETURN
2000 !***** Create the files *****
2010 FOR W=1 TO NF :: FL$="DSK1."&F$&STR$(W):: PRINT FL$
2020 FOR J=1 TO SZ :: PRINT "File #";W;"", entry #";J
2030 INPUT N$(J):: NEXT J :: PRINT
2040 K=1 :: GOSUB 8000 ! Write names out on file.
2050 NEXT W :: RETURN
3000 !***** Edit entries within a file *****
3010 FOR W=1 TO NF :: FL$="DSK1."&F$&STR$(W)
3020 PRINT "Input file is ";FL$ :: K=1 :: GOSUB 7000 ! Read.
3030 FOR J=1 TO SZ
3040 PRINT N$(J);TAB(30);": ";
3050 INPUT NE$ :: IF LEN(NE$)<>0 THEN N$(J)=NE$
3060 NEXT J
3070 GOSUB 8000 !Write out edited file.
3080 NEXT W :: RETURN
4000 !***** Single file sort. *****
4010 CALL CLEAR :: FOR W=1 TO NF :: FL$="DSK1."&F$&STR$(W)
4020 K=1 :: GOSUB 7000 ! Read the file.
4030 M=SZ
4040 M=INT(M/2):: PRINT J:: IF M=0 THEN PRINT :: GOTO 4110
4050 K=SZ-M :: J=1
4060 I=J
4070 L=I+M :: IF N$(I)<=N$(L) THEN 4100
4080 T$=N$(I):: N$(I)=N$(L):: N$(L)=T$ :: I=I-M
4090 IF I>=1 THEN 4070
4100 J=J+1 :: IF J<=K THEN 4060 ELSE 4040
4110 PRINT #9:"File ";FL$;" in sorted order."
4120 N=SZ :: GOSUB 9000 !Print the sorted file.
4130 K=1 :: GOSUB 8000 !Write out sorted file to disk.
4140 PRINT :: PRINT :: PRINT :: PRINT :: NEXT W :: RETURN
5000 !***** Merge the files. *****
5010 PRINT "Requires ";NF+1;" files." :: INPUT A$
5020 PRINT #9:"merged file 'NEW'." :: OPEN #NF+1:"DSK1.NEW",SEQUENTIAL,INTERNAL,
OUTPUT
5030 FOR W=1 TO NF :: OPEN #W:"DSK1."&F$&STR$(W),SEQUENTIAL,INTERNAL,INPUT
5040 INPUT #W:N$(W):: K2(W)=1 :: NEXT W
5050 CN=0 :: S=0 :: PLUG$="zzzzzz" :: GOSUB 6000
5060 PRINT #NF+1:SMALL$ :: CN=CN+1 :: PRINT CN:: K2(M)=K2(M)+1
5070 IF K2(M)<=SZ THEN INPUT #M:N$(M):: GOSUB 6000 :: GOTO 5060
5080 N$(M)=PLUG$ :: S=S+1 :: IF S<NF THEN GOSUB 6000 :: GOTO 5060
5090 PRINT :: FOR I=1 TO NF+1 :: CLOSE #I :: NEXT I :: SZ=SZ*NF :: K=NF+1 :: FL$
="DSK1.NEW" :: GOSUB 7000
5100 N=SZ :: GOSUB 9000 :: SZ=SZ/NF :: RETURN
6000 !***** Find the smallest of NF strings. *****
6010 SMALL$=PLUG$
6020 FOR W=1 TO NF :: IF N$(W)<SMALL$ THEN SMALL$=N$(W):: M=W
6030 NEXT W :: RETURN
7000 !***** Load a data file from diskette. *****
7010 OPEN #K:FL$,SEQUENTIAL,INTERNAL,INPUT
7020 FOR J=1 TO SZ :: INPUT #K:N$(J):: NEXT J
7030 CLOSE #K :: RETURN
8000 !*****Save a data file on diskette. *****
8010 OPEN #K:FL$,SEQUENTIAL,INTERNAL,OUTPUT
8020 FOR J=1 TO SZ
8030 PRINT #K:N$(J)
8040 NEXT J :: PRINT :: CLOSE #K :: RETURN
9000 !***** Print a file from memory *****
9010 N1=INT(N/3):: FOR J=1 TO N1
9020 PRINT #9:J;N$(J);TAB(20);J+N1;N$(J+N1);TAB(40);J+2*N1;N$(J+2*N1)
9030 NEXT J :: PRINT #9 :: RETURN
9999 CLOSE #9 :: END

```

```

File DSK1.FLORA1 in original order.
 1 pilewort          5 peony             9 pyracantha
 2 peat moss         6 poison ivy       10 polygala
 3 portulaca         7 parsnip          11 pinkroot
 4 petunia           8 plantain         12 phlox

File DSK1.FLORA2 in original order.
 1 pansy             5 palm             9 plumbago
 2 polygonum         6 parsley          10 poinsettia
 3 primrose          7 pimperl         11 pennyroyal
 4 pipsissewa       8 pieplant        12 pachysandra

File DSK1.FLORA3 in original order.
 1 pak choi          5 privet           9 peperomia
 2 peyote            6 pokeweed         10 pigweed
 3 pothos            7 poppy            11 peppermint
 4 phacelia          8 palmetto         12 purslane

File DSK1.FLORA1 in sorted order.
 1 parsnip           5 phlox            9 poison ivy
 2 peat moss         6 pilewort         10 polygala
 3 peony             7 pinkroot         11 portulaca
 4 petunia           8 plantain         12 pyracantha

File DSK1.FLORA2 in sorted order.
 1 pachysandra       5 pennyroyal       9 plumbago
 2 palm              6 pieplant         10 poinsettia
 3 pansy             7 pimperl         11 polygonum
 4 parsley           8 pipsissewa      12 primrose

File DSK1.FLORA3 in sorted order.
 1 pak choi          5 peyote           9 poppy
 2 palmetto          6 phacelia         10 pothos
 3 peperomia         7 pigweed          11 privet
 4 peppermint        8 pokeweed         12 purslane

merged file 'NEW'.
 1 pachysandra       13 petunia         25 poinsettia
 2 pak choi          14 peyote          26 poison ivy
 3 palm              15 phacelia        27 pokeweed
 4 palmetto          16 phlox           28 polygala
 5 pansy             17 pieplant        29 polygonum
 6 parsley           18 pigweed         30 poppy
 7 parsnip           19 pilewort        31 portulaca
 8 peat moss         20 pimperl         32 pothos
 9 pennyroyal        21 pinkroot        33 primrose
10 peony             22 pipsissewa     34 privet
11 peperomia         23 plantain        35 purslane
12 peppermint        24 plumbago        36 pyracantha

```

Index Production

The last two programs in this chapter are true production programs. We use them to produce our indices in the books we wrote. Program INDEXBLD accepts strings of text as input. Each string represents an index topic and its page number separated by the @ symbol. These are typical input entries:

```

STR$@24
Real Variables@44
Six-bit graphic code@78
VAL@24

```

The user types any number of these entries in one sitting, ending with "end". These entries can be stored on disk in the form of a sequential file. Later, the user may type additional entries into this file until all index entries from a book, up to 750 of them, are entered. When all entries are filed in this fashion, the user can edit individual entries, add or delete entries, or sort the file. This is an in-memory sort, and the program for this reason needs to run on a 48K system. This is one of the few programs in the entire series of books which needs such a large system.

This program uses sequential files merely as resting places for the up to 750 entries. All processing is performed in memory, usually on string arrays.

We have elected to show you the listing of both the file builder and the index printer programs, followed by some of the output that is produced at different stages of the program's execution.

We thank Steve Grillo for programming this application.

```

10 REM filename: "idxbld"
20 REM purpose: Build a book's index
30 REM author: spg, jpg & jdr 10/82
40 REM -----
50 DIM X$(750)!Allow up to 750 separate index entries.
60 ! Print menu of activities.
70 CALL CLEAR
80 PRINT " A C T I V I T I E S:"
90 PRINT "1-Read a data file from      diskette to memory."
100 PRINT "2-Place the memory-based     file onto diskette."
110 PRINT "3-Type in new index entries."
120 PRINT "4-Alter any index entries."
130 PRINT "5-Sort the index in            alphabetical order."
140 PRINT "6-Display or print the index     as it is in memory."
150 PRINT "7-Reformat the entries (for      appearance)."
160 PRINT "8-Type the final edited        version of the index (It must be fi
rst sorted on disk)."
180 PRINT "9-Stop the execution of the  program."
190 PRINT
240 PRINT "Type the number of the      preferred activity. ";
250 INPUT A :: CALL CLEAR
260 IF A<1 OR A>9 THEN 70
270 ON A GOSUB 1000,2000,3000,4000,5000,6000,7000,8000,9999
280 PRINT :: PRINT :: GOSUB 8040 :: GOTO 70
1000 !***** Read old file into X$ array. *****
1010 PRINT "What file should be read " :: INPUT F$
1020 OPEN #2:"DSK1."&F$,SEQUENTIAL,INTERNAL,INPUT :: INPUT #2:F1$
1030 PRINT "New file name: ";F$
1040 N=0 :: PRINT :: PRINT
1050 IF EOF(2) THEN CLOSE #2 :: RETURN ELSE N=N+1
1060 INPUT #2:X$(N):: PRINT N;X$(N):: GOTO 1050
2000 !***** Write a new file. *****
2010 INPUT "What is the name of the file you want opened":F$
2020 INPUT "Are you sure you want the old file killed":A$
2030 IF A$<>"YES" AND A$<>"yes" THEN RETURN
2040 OPEN #1:"DSK1."&F$,SEQUENTIAL,INTERNAL,OUTPUT
2050 ! Write to the file
2060 PRINT #1:DATE$;CHR$(13);F$
2070 PRINT " Please be patient during data transfer."
2080 PRINT " (Data moving from memory to diskette.)"
2090 PRINT
2100 FOR I=1 TO N
2110 PRINT #1:X$(I):: PRINT I;X$(I)
2120 NEXT I :: CLOSE #1 :: RETURN

```

```

3000 !***** Add new entries. *****
3010 PRINT "Enter entries as follows:"
3020 PRINT ", "Keyword@pg,pg,etc..."
3030 PRINT :: PRINT
3040 PRINT "Type 'end' to finish entries."
3050 N=N+1
3060 PRINT USING " No. ###:":N;:: LINPUT X$(N)
3070 IF X$(N)="end" THEN X$(N)="" :: N=N-1 :: RETURN
3080 D=POS(X$(N),"@",1)! Divider position.
3090 IF D=0 THEN PRINT ">>>Syntax error<<<" :: PRINT "re-enter with @" :: GOTO 3
060
3100 GOTO 3050
4000 !***** Edit an entry *****
4010 CALL CLEAR
4020 PRINT "1-Delete a specific entry."
4030 PRINT "2-Search the sorted file for duplicate keywords and"
4040 PRINT " alter them accordingly."
4050 PRINT "3-Re-enter a specific entry."
4060 PRINT "4-Alter the page numbers on an enter."
4070 PRINT "5-Switch two entries around."
4080 PRINT "6-Return to normal command mode."
4090 INPUT "Which activity":A$ :: IF A$="" THEN 4000
4100 CALL CLEAR :: A=VAL(A$)
4110 ON A GOTO 4130,4250,4390,4560,4450,4120
4120 RETURN
4130 ! Delete an entry
4140 INPUT "What is the entry number (0 to end deletions)":D
4150 IF D<>0 AND D<=N THEN X$(D)="" :: GOTO 4130
4160 ! Rearrange the text entries after deletion.
4170 C=0 :: PRINT "Please wait during rearrngment of index."
4180 FOR I=1 TO N :: IF X$(I)="" THEN 4200
4190 C=C+1 :: X$(C)=X$(I):: PRINT C;X$(C)
4200 NEXT I
4210 ! Blank all of the extra array positions after deletion.
4220 FOR I=C+1 TO N :: X$(I)="" :: NEXT I :: N=C
4230 GOSUB 8040 :: GOTO 4000
4240 ! Delete all duplicate entries and alter the page numbers.
4250 CALL CLEAR :: I=0
4260 DISPLAY AT(11,1):"Checking entry#";
4270 I=I+1 :: A=POS(X$(I),"@",1)
4280 ! If search of duplicates is done, rearrange the
4290 ! text with the same subroutine that the deletion
4300 ! routine uses.
4310 IF I>=N THEN 4170
4320 IF SEG$(X$(I),1,A)=SEG$(X$(I+1),1,A) THEN 4340
4330 DISPLAY AT(11,17):I;:: GOTO 4270
4340 PRINT :: PRINT I;" and";I+1;" are duplicates."
4350 PRINT X$(I):: PRINT X$(I+1)
4360 X$(I+1)=X$(I)&","&SEG$(X$(I+1),A+1,LEN(X$(I+1))-A):: PRINT :: PRINT X$(I+1)
4370 X$(I)="" :: GOSUB 8040 :: CALL CLEAR :: GOTO 4260
4380 ! Re-enter a complete entry
4390 INPUT "What is the entry number(0 to exit)":A
4400 IF A=0 THEN 4260
4410 PRINT " This is the old entry...";X$(A)
4420 LINPUT "Please type in new entry":X$(A)
4430 PRINT :: PRINT "#";A;" is now ";X$(A)
4440 GOSUB 8040 :: GOTO 4000
4450 ! Switch two entries around.
4460 INPUT "What are the two numbers ":A,B
4470 IF A<1 OR A>N OR B<1 OR B>N THEN RETURN
4480 PRINT "The old entries were:"
4490 PRINT A;X$(A):: PRINT B;X$(B)
4500 PRINT :: PRINT
4510 PRINT "The revised entries look like this:"
4520 T=X$(A):: X$(A)=X$(B):: X$(B)=T
4530 PRINT A;X$(A):: PRINT B;X$(B)
4540 GOSUB 8040 :: GOTO 4000
4550 ! Add pages to an entry.

```



```

4560 INPUT "What is the key number (0 to exit)":D
4570 IF D=0 OR D>N THEN 4000
4580 LINPUT "What are the page additions (pg,pg...)":C$
4590 X$(D)=X$(D)&"", "&C$
4600 PRINT "Entry#";D;" now looks like this:";X$(D)
4610 PRINT :: GOTO 4560
5000 !*** Befor sorting change all 1st. to capitals.****
5010 CALL CLEAR
5020 PRINT "Before sorting the items, "
5030 PRINT "all items will be capitalized."
5040 PRINT "Capitalizing #";
5050 FOR I=1 TO N :: A=ASC(X$(I)):: IF A<97 OR A>123 THEN 5070
5060 X$(I)=CHR$(A-32)&SEG$(X$(I),2,LEN(X$(I))-1)
5070 PRINT I;:: NEXT I
5080 !** Sort entries--Shell-Metzner
5090 CALL CLEAR
5100 M=N
5110 PRINT "Now sorting the items."
5120 PRINT
5130 PRINT " When the number above"
5140 PRINT " reaches zero, the sort will be complete."
5150 M=INT(M/2):: DISPLAY AT(4,10):" ";; DISPLAY AT(4,10):M;
5160 IF M=0 THEN RETURN
5170 K=N-M :: J=1
5180 I=J
5190 L=I+M
5200 P8=POS(X$(I),"@",1):: P9=POS(X$(L),"@",1)
5210 IF SEG$(X$(I),1,P8)<SEG$(X$(L),1,P9)THEN 5260
5220 IF SEG$(X$(I),1,P8)>SEG$(X$(L),1,P9)THEN 5240
5230 IF VAL(SEG$(X$(I),P8+1,1))<=VAL(SEG$(X$(L),P9+1,1))THEN 5260
5240 T$=X$(I):: X$(I)=X$(L):: X$(L)=T$ :: I=I-M
5250 IF I>=1 THEN 5190
5260 J=J+1
5270 IF J<=K THEN 5180 ELSE 5150
6000 !***** Display memory.*****
6010 INPUT "Output to the printer(1=yes)":P1 :: IF P1=1 THEN OPEN #9:"RS232"
6020 PRINT "Type the first and last key# that you want to see."
6030 INPUT " Type 0,0 for all entries.":A,B
6040 IF A<1 OR A>N OR A>B OR B<1 OR B>N THEN A=1 :: B=N
6050 CALL CLEAR
6060 FOR I=A TO B :: PRINT I;X$(I)
6070 IF P1=1 THEN PRINT #9:I;X$(I)
6080 IF INT(I/22)=I/22 THEN GOSUB 8040
6090 NEXT I :: IF P1=1 THEN CLOSE #9
6100 RETURN
6110 ! Format the entries by:
6120 ! deleting unwanted spaces and
6130 ! adding desired spaces after all commas.
7000 !***** Format entries *****
7010 FOR I=1 TO N :: I3=1
7020 IN=POS(X$(I)," ",1):: IF IN=0 THEN 7040
7030 X$(I)=SEG$(X$(I),1,IN-1)&SEG$(X$(I),IN+1,LEN(X$(I))-IN):: GOTO 7020
7040 IN=POS(X$(I),"",I3):: IF IN=0 THEN 7080
7050 IF SEG$(X$(I),IN+1,1)=" " THEN 7070
7060 X$(I)=SEG$(X$(I),1,IN)&" "&SEG$(X$(I),IN+1,LEN(X$(I))-IN+1)
7070 I3=IN+2 :: GOTO 7040
7080 PRINT I;X$(I)
7090 NEXT I
7100 RETURN
8000 !***** Print the index from a disk file.*****
8010 INPUT "Do you have a perfect file saved on a disk ":A$
8020 IF A$="YES" OR A$="yes" THEN RUN "DSK1.INDEXPRT"
8030 RETURN
8040 PRINT :: PRINT "<<ENTER>>";
8050 INPUT A$ :: CALL CLEAR :: RETURN
9999 END

```

Enter entries as follows:
Keyword@pg,pg,etc...

Type 'end' to finish entries.

No. 2:gymnosperm@20
No. 3:yew@20
No. 4:pacific yew@20
No. 5:western yew@20
No. 6:california torrya@20
No. 7:cycad@20
No. 8:gymnosperm@21
No. 9:angiosperm@
No. 10:angiosperm@21
No. 11:pacific yew

>>>Syntax error<<<

re-enter with @

No. 11:pacific yew@21
No. 12:zamia@21
No. 13:sago palm@21
No. 14:pine@22
No. 15:eastern white pine@22
No. 16:pinus strobus

>>>Syntax error<<<

re-enter with @

No. 16:pinus strobus@22

```
10 REM filename: "indexprt"
20 REM purpose: Prints a book's index
30 REM author: jpg & jdr 10/82
40 REM -----
50 CALL CLEAR :: DIM Y*(850)! Allow for 850 final index strings.
60 OPEN #1:"RS232" :: INPUT "What file name for the index":F$
70 OPEN #2:"DSK1."&F$,SEQUENTIAL,INTERNAL,INPUT
80 !
90 !
100 !
110 ! Retrieve all necessary information now.
120 INPUT "How many lines long should the columns be ":L
130 PRINT :: INPUT "How wide (maximum) should the columns be ":W
140 PRINT :: PRINT "How many blank lines should be left"
150 INPUT " in between different letters":S
160 PRINT :: PRINT "How many lines should be left"
170 INPUT " for the title on the first page of the index":T
180 PRINT :: INPUT "Should the index be printed (1=yes) ":P1
190 IF P1<>1 THEN 320
200 PRINT :: PRINT " Pages can be printed in two ways."
210 PRINT " 1-Type the index automatically or"
220 PRINT " 2-Type each page at your command."
230 PRINT :: INPUT "Which way do you prefer (1 or 2)":D1
240 IF D1<>1 THEN 320
250 PRINT :: PRINT
260 PRINT " If the page length isn't standard, type the"
270 PRINT "number of lines on each page. Otherwise,"
280 PRINT "just press <<ENTER>>."
290 PRINT :: INPUT "How many lines per page. ":LL
300 IF LL=0 THEN LL=66
310 IF L>LL THEN PRINT "More than column length of";L:GOTO 180
320 PRINT
330 PRINT "How many spaces should be left between the"
340 INPUT " index word(key) and the page numbers":SP
350 PRINT :: PRINT "How many spaces should be indented on the"
360 INPUT " second line of a long index entry":LS
370 !
380 !
390 !
```

```

400 ! Convert the unformatted X$ strings (on file)
410 ! into formatted Y$ stored in memory.
420 P$=" " :: P=T
430 P=P+1
440 ! Blank the lines at top of first page.
450 IF P>L AND P<T+L+1 THEN Y$(P)=" " :: GOTO 430
460 ! Read an entry from the data file.
470 IF EOF(2) THEN 800 ELSE INPUT #2:X$
480 IF SEG$(X$,1,1)=P$ OR S=0 THEN 580
490 IF P-INT(P/L)*L=1 THEN 580
500 FOR I=1 TO S+1
510 Y$(P)=" " :: PRINT P;Y$(P):: P=P+1
520 IF P=L+1 THEN P=T+L-1 :: GOTO 580
530 IF (P-1)/L=INT((P-1)/L) THEN 580
540 NEXT I
550 P=P-1
560 ! Keep track of the first letters with P$
570 P$=SEG$(X$,1,1)
580 ! Replace @ in the string with correct # of spaces.
590 IN=POS(X$,"@",1):: IF IN>0 THEN X$=SEG$(X$,1,IN-1)&RPT$(" ",SP)&SEG$(X$,IN+1
,LEN(X$)-IN)
600 ! Skip next section if entire entry short enough.
610 IF LEN(X$)<W THEN Y$(P)=X$ :: PRINT P;Y$(P):: GOTO 430
620 T$=X$ :: FLAG=1 ! Flag to make loop go to 1st space..
630 ! Shorten the long string here if no commas or spaces.
640 IF FLAG>1 THEN GOSUB 980
650 FOR K=W TO FLAG STEP -1
660 IF SEG$(T$,K,1)=" " THEN 710
670 NEXT K
680 ! Shorten the long string here if no commas or spaces.
690 C$=SEG$(T$,1,W)&"-" :: GOSUB 1080 :: P=P+1
700 T$=RPT$(" ",LS&SEG$(T$,W+1,LEN(T$)-W):: GOTO 740
710 ! Shorten the string here if there is a comma or a space.
720 C$=SEG$(T$,1,K-1):: GOSUB 1080 :: P=P+1
730 T$=RPT$(" ",LS)&SEG$(T$,K+1,LEN(T$)-K)
740 FLAG=LS+1
750 IF LEN(T$)>W THEN 640 ELSE GOSUB 980 :: C$=T$ :: GOSUB 1080 :: GOTO 430
760 !
770 !
780 !
790 ! Print the index
800 PC=0 :: I=1 :: L5=0 !Initialize the variables used here.
810 PC=PC+1 :: PRINT TAB(30);"Page#";PC
820 IF P1=1 THEN PRINT #1:TAB(60);"Page#";PC :: PRINT #1
830 FOR J=I TO I+L-1
840 PRINT Y$(J);TAB(W+4);Y$(J+L)
850 L5=L5+2
860 IF P1=1 THEN PRINT #1:Y$(J);TAB(W+4);Y$(J+L)
870 NEXT J
880 IF D1=1 AND P1=1 THEN 885 ELSE GOTO 910
885 FOR J=1 TO LL-L-2 :: PRINT #1 :: NEXT J :: GOTO 910
890 PRINT "Press /en/ when you are ready for another page";
900 INPUT A$ :: GOTO 900
910 PRINT :: PRINT :: PRINT
920 IF L5<P THEN I=I+L*2 :: GOTO 810
930 GOTO 1120
940 !
950 !
960 !
970 ! Subroutine to delete unwanted preceding spaces from T$
980 IF FLAG>LEN(T$)OR SEG$(T$,FLAG,1)<>" " THEN 1000
990 T$=SEG$(T$,1,FLAG-1)&SEG$(T$,FLAG+1,LEN(T$)-FLAG):: GOTO 980
1000 RETURN

```

```

1010 !
1020 !
1030 !
1040 !   The following subroutine makes sure that the
1050 ! first line at the top of the column dosen't begin
1060 ! with a page number left over from the previous
1070 ! column. No "windows".
1080 IF P=T+L+1 THEN T4=1 ELSE T4=(P+L-1)/L
1090 IF T4<>INT(T4)OR SEG*(C$,1,LS)<>RPT$(" ",LS)THEN Y$(P)=C$ :: GOTO 1110
1100 Y$(P)=Y$(P-1):: Y$(P-1)=" " :: P=P+1 :: Y$(P)=C$
1110 PRINT P;Y$(P):: RETURN
1120 CLOSE #1 :: CLOSE #2 :: END

```

What file name for the index TREE
How many lines long should the columns be 22
How wide (maximum) should the columns be 25
How many blank lines should be left
in between different letters 0
How many lines should be left
for the title on the first page of the index 3
Should the index be printed (1=yes) 1
Pages can be printed in two ways.
1-Type the index automatically or
2-Type each page at your command.
Which way do you prefer (1 or 2) 2
How many spaces should be left between the
index word(key) and the page numbers 2
How many spaces should be indented on the
second line of a long index entry 4

Page# 1

Angiosperm 21	Jeffery pine 26,27
Apache pine 32,33	Knobcone pine 30,31
Austrian pine 34,35	Limber pine 25,24
Bishop pine 24,25	Loblolly pine 32,33
Bristlecone pine 24 ,25	Lodgepole pine 26,27
	Longleaf pine 32,33
	Mexican pinyon 25
California torreyia	Monterey pine 34,35
20,21	Pacific yew 20,21
Chihuahua pine 34,35	Parry pinyon 25
Coulter pine 30,31	Pine 22
Cycad 20	Pinus albicaulis 24
Digger pine 32,33	Pinus aristata 24
Eastern white pine	Pinus attenuata 30
22,23	Pinus balfouriana 24
Foxtail pine 24,25	Pinus banksiana 26
Gymnosperm 20,21	Pinus clausa 28
Hard pine 23	Pinus contorta 26
Jack pine 26,27	Pinus coulteri 30

Pinus echinata 26
Pinus elliottii 32
Pinus engelmannii 32
Pinus flexilis 24
Pinus glabra 28
Pinus jeffreyi 30
Pinus lambertiana 22
Pinus leiophylla 34
Pinus monticola 22
Pinus muricata 28
Pinus nigra 34
Pinus palustris 32
Pinus ponderosa 30
Pinus pungens 28
Pinus resinosa 26
Pinus rigida 32
Pinus sabiniana 32
Pinus serotina 34
Pinus strobus 22
Pinus sylvestris 34
Pinus taeda 32
Pinus torreyana 34

Pinyon 25
Pitch pine 34,35
Pond pine 34,35
Ponderosa pine 30,31
Red pine 26,27
Sago palm 21
Sand pine 28,29
Scotch pine 26,27
Shortleaf pine 26,27
Singleleaf pinyon 25
Slash pine 28,29
Soft pine 22,23
Spruce pine 28,29
Sugar pine 22,23
Table-mountain pine
28,29
Torry pine 34,35
Virginia pine 35,35
Virginiana 28
Western white pine
22,23
Western yew 20

Page# 2

Whitebark pine 24,25
Yew 20
Zamia 21

Page# 3

This chapter has included programming examples that use disk sequential files. Of course, some of these examples could just as well have used tape sequential files with minor cosmetic changes in the code as long as only one tape file is opened at a time. This is one advantage of sequential file management techniques: they can be performed on both tape and disk files. Chapter 6 will deal with a file management technique that is restricted to the disk for storage. You will see that although you are limited to one storage medium, the disk, you will gain a great deal of flexibility in choice of processing techniques.



Direct Access Files

The advent of the personal computer in the late 1970's has prompted the designers and manufacturers of disk drives to place a wide variety of hardware to support direct access files on the market. When we purchased our first microcomputer early in 1978, we were placed on a six-month waiting list for a disk drive. We owned the drive and its operating system software for only two months before the vendor supplied us with an updated version of the Disk Operating System.

Since then, frequent upgrades in our disk system have been accompanied by a host of other products touted by other vendors. This results in a disk capable of holding enough information for many small businesses.

It's not increases in the speed of computation that serious users need and want. They would much rather have a reliable and supported, large-capacity, direct access file system. The direct access feature is of paramount importance. Most file-based application programs rely on the technique of directly accessing a selected record without having to search all previously written records, as was discussed in the previous chapter.

In this chapter we will discuss four major topics that deal with direct access files: (1) Searching the file for a specific record, (2) generating a record's address by hashing, (3) sorting the file in ascending or descending order, and (4) indexed sequential access method (ISAM) organization of data on disk.

File Searching

A direct access file is characterized by the way records are either read from or written onto the file. The program produces a *pointer* to the file which acts as a *record number*, or *address*, and then the command PRINT #n,REC x: or INPUT #n,REC x: is used to deal with that specific record, without ever having to deal with any of the other records on file. In this scheme, if the file contains seven records and the desired record is the seventh, an INPUT #n,REC x command can access that record directly, hence the name *direct access*.

Suppose a file exists on disk, and that file is unordered. That is, if you know that record 5 has the name MANDALAY, you can't assume that the record with the name FARQUHAR is before record 5 just because it is lower in alphabetical order. Therefore to search the file for a given access key, such as FARQUHAR, you must proceed exactly as you did in Chapter 5 with a sequential file.

But suppose a file on disk is sorted in ascending order. Then, if you know that record 5 has the key MANDALAY, the record with the key FARQUHAR must be stored before record 5, and that provides a tidy limit on your search. Suppose further that you access record 3 and find that it contains the key DANDIFY. Now you know for certain that if the key FARQUHAR is not on record 4, it is not on the file at all!

Binary Search

The *binary searching technique* relies on the fact that the direct access file is ordered. It may be ordered in either ascending or descending sequence, but the direction of ordering must be known. Suppose a file with nine records has keys that are numeric values, arranged as shown below:

Record	1	2	3	4	5	6	7	8	9
Key	24	30	31	44	49	55	68	73	82

What is the most efficient way to find a specific record on the file? One way to find a record would be to access each record in turn, from 1 to 9, until the one you want is found. This is the directed scan on an ordered file. But since you can fetch any record directly, it would be more advantageous to skip around a bit. Suppose you want to get the record that has the key value 55.

1. Get the middle record with key A(5).
2. A(5) is a 49, which is less than 55.
3. Subdivide the upper half of the file.
4. Get the middle record of this segment with key A(7).
5. A(7) is a 68, which is more than 55.
6. Subdivide the lower half of this segment.
7. Get the middle record of this segment with key A(6).
8. A(6) is a 55. Found!

This technique of successively splitting the file into segments whose bounds surround the key sought is called the *binary search*. The algorithm in its general form is shown below as a subroutine that searches the array A for the key X.

```

1000 '***** Binary Search subroutine
1010 'X= key sought
1020 'A = array being searched
1030 'N = number of records (entries) in A
1040 L=1 'Set lower limit to 1
1050 U=N+1 'Set upper limit to array size +1
1060 I=INT((L+U)/2) 'Set pointer half way between L & U
1070 IF A(I)=X THEN PRINT X;" found at pos. ";I :: RETURN
1080 IF A(I)<X THEN IF L=I THEN PRINT X;" not found" :: RETURN ELSE L=I :: GOTO
1060
1090 IF A(I)>X THEN IF U=I THEN PRINT X;" not found" :: RETURN ELSE U=I :: GOTO
1060

```

The file WORDS is the same file that was used in Chapter 5. Here we will use it to illustrate the binary search. The program SEARCH reads the entire file into the two memory arrays W\$ and F in the order that they appear on the sequential file, that is, in descending order of frequency of occurrence in English text. Then the program deals with the memory array as if it were a direct access file, using subscripts to access selected array entries rather than pointers to access selected file records. Notice the strong parallels between sequential files and DATA statements on one hand, and direct access files and arrays on the other. Here we simulate the activity that is possible on direct access files using memory arrays. We are dealing with a rather small set of data, which if distributed one word and one frequency per record over a direct access file would occupy $100 \times 255 = 25,500$ bytes, or about half of the available space on a disk. The only other alternative would be to *block* the records, say 10 words and frequencies per physical record. If we did this, it would completely mask the intent of the program, which is to demonstrate the technique of the binary search on an ordered list.

The program SEARCH performs a binary search on the array F at the request of the user. The request is worded as a specific frequency. Then the computer responds by displaying all words with a higher frequency, their count, and their cumulative percentage of use. The response also includes the number of accesses to the array that were necessary to find the desired frequency's position in the sorted list. Notice that any frequency in this list of 100 can be accessed in 8 tries or less every time. If the list contained 1000 entries, the required number of accesses would be 11 or less. You could find one in 1,000,000 with 21 accesses or less. The *maximum* number of accesses required to fetch a given record in a set of N records is given by the formula:

$$A = \log_2 N + 1 \quad (A = \log_2 N + 2 \text{ if item sought is not on file})$$


```

10 REM filename: "search"
20 REM purpose: In-memory binary and interpolation searches
30 REM author: jpg & jdr 10/82 (car)
40 REM -----
50 DIM W$(25),F(26)!Note the extra space at end
60 OPEN #1:"RS232" :: OPEN #2:"DSK1.WORDS",SEQUENTIAL,INTERNAL,INPUT
70 FOR I=25 TO 1 STEP -1
80 IF EOF(2)THEN 110
90 INPUT #2:W$(I),F(I)
100 NEXT I :: N=25
110 CLOSE #2 :: S$(1)="Binary" :: S$(2)="Interpolation"
120 INPUT "What type of search(1=Binary, 2=Interpolation,0=STOP) ":L
130 INPUT "What is minimum frequency ":X
140 IF L=0 THEN 9999 :: PRINT #1:S$(L);" search / minimum frequency";X
150 ON L GOSUB 1000,2000 'Find pos. just larger than X
160 S=0 :: PRINT #1:"Next higher freq. is ";F(P);"at";26-P
170 FOR I=25 TO P STEP -1
180 S=S+F(I):: PRINT #1:W$(I);" ";
190 IF INT(I/9)*9=I THEN PRINT #1
200 NEXT I
210 PC=S/2424.32 'Cumulative % of 242432 words in text analyzed
220 PRINT #1 :: PRINT #1:"Cumulative % of use is";PC
230 PRINT #1:"For these";25-P;"words";
240 PRINT #1:" it took";k;"accesses to find";F(P)
250 GOTO 120
260 CLOSE #1 :: CLOSE #2 :: END
1000 !***** Binary search of F array for X *****
1010 L=1 :: U=N+1 :: K=0
1020 I=INT((L+U)/2):: K=K+1 'K is count of accesses to array
1030 IF F(I)=X THEN P=I :: RETURN
1040 ' Return the next higher pointer if not exact
1050 IF F(I)<X THEN IF L=I THEN P=I+1 :: RETURN ELSE L=I :: GOTO 1020
1060 IF U=I THEN P=I+1 :: RETURN ELSE U=I :: GOTO 1020
2000 !**** Subroutine INSEARCH interpolation search ****
2010 F(0)=F(1)-1 :: F(N+1)=F(N)+1 :: K=0
2020 L=0 :: R=N+1 :: X1=F(L):: X2=F(R)
2030 IF X<=X1 OR X>=X2 THEN PRINT "out of range" :: RETURN
2040 IF R<L THEN S=R :: GOTO 2090
2050 T=L+INT((R-L)*(X2-X1)/(X2-X1)):: K=K+1
2060 H=F(T)
2070 IF H=X THEN S=T :: P=S :: RETURN
2080 IF H<X THEN L=T+1 :: X1=H :: GOTO 2040 ELSE R=T-1 :: X2=H :: GOTO 2040
2090 IF X=F(S)THEN P=S :: RETURN ELSE P=S+1 :: RETURN
9999 CLOSE #1 :: END

```

```

Binary search / minimum frequency 1500
Next higher freq. is 1535 at 17
the of and to a in that is
i it for as with was his he be

```

```

Cumulative % of use is 29.07908197
For these 16 words it took 6 accesses to find 1535

```

Interpolation Search

The program SEARCH also demonstrates another technique for searching an ordered file. This method, called the *interpolation search*, is not commonly used, perhaps because it is less obvious and more difficult to code. However, it is not any less efficient, and in some cases of ordering, it may result in fewer accesses to the file to either find an entry or to determine that it isn't there.

The algorithm below shows the interpolation search on an ordered list A as a subroutine.

```

1000 !***** Subroutine for interpolation search
1010 ! A & X are array being searched and key sought
1020 A(0)=A(1)-1 !Set lowest value
1030 A(N+1)=A(N)+1 !Set highest value
1040 L=0 :: R=N+1 !Set low and high pointers
1050 X1=A(L):: X2=A(R)!Fetch boundary values
1060 IF X<X1 THEN 1500 !Not found
1070 IF X>X2 THEN 1500 !Not found
1080 IF R>=L THEN 1100
1090 S=R
1100 IF X=A(S)THEN 1600 ELSE 1500
1110 T=INT((R-L)*(X-X1)/(X2-X1))!Interpolate position
1120 IF A(T)=X THEN S=T :: GOTO 1600 ELSE IF A(T)>X THEN R=T-1 :: X2=A(T):: GOTO
1080 ELSE L=T+1 :: X1=A(T):: GOTO 1080
1500 PRINT X;" not found" :: RETURN
1600 PRINT X;" at position ";S :: RETURN

```

The interpolation search technique is obviously more complex than the binary search, so the real test of its worth should rest in its performance. Program INSEARCH is the interpolation search subroutine which is present in SEARCH and we have elected to isolate it here as a separate routine. Notice that the output shows that the number of accesses to the array is not significantly different from the number of accesses which the binary search used to fetch a desired entry. We suggest that you try both techniques in programs that require the access to sorted files, and adopt the one which seems to work better for you.

```

2000 !**** Subroutine INSEARCH interpolation search ****
2010 F(0)=F(1)+1 :: F(N+1)=F(N)-1 :: K=0
2020 L=0 :: R=N+1 :: X1=F(L):: X2=F(R)
2030 IF X<=X1 OR X>=X2 THEN PRINT "out of range" :: RETURN
2040 IF R<L THEN S=R :: GOTO 2090
2050 T=L+INT((R-L)*(X2-X1)/(X2-X1)):: K=K+1
2060 H=F(T)
2070 IF H=X THEN S=T :: P=S :: RETURN
2080 IF H<X THEN L=T+1 :: X1=H :: GOTO 2040 ELSE R=T-1 :: X2=H :: GOTO 2040
2090 IF X=F(S)THEN P=S :: RETURN ELSE P=S+1 :: RETURN

```

```

Interpolation search / minimum frequency 1500
Next higher freq. is 1535 at 17
the of and to a in that is
i it for as with was his he be

```

```

Cumulative % of use is 29.08238186
For these 16 words it took 19 accesses to find 1535

```

Hash Address Processing

There are many occasions when sorting the direct access file is not practical, for example if the file is highly *volatile*, such that records are deleted or added often. In such instances, a record on the file can be accessed by using the access key itself to generate a record number (the record's *address* on the file) so that only one access to the file is necessary. This technique is called *hash addressing*.

Hash address processing to build a direct access file proceeds as follows:

1. Transform the key A(I) into a record address X.
2. Fetch the record X.

3. If the record is blank (if the Xth position of the file has never been used) then write the key A(I) and its associated elements there.
4. If the record is not blank, write it at the first available space in an overflow area.

The overflow area is necessary when hash address processing is used because there is no way to guarantee that every key A(I) can be transformed into a unique record number. When a key generates an already used record number, the effect is called a *collision*. The transformation of a key into an address depends on a *hashing function*, which may be effective (generate few collisions and many different record numbers) when used on one type of key, but not on another type of key. For example, hashing function F may yield 40 unique numbers between 1 and 100, with only five duplications (collisions), when used with alphabetic last names, but yields only 25 unique numbers and 20 duplications with street addresses.

We will show you five hashing functions, and demonstrate their effectiveness, using the program HASHING. The functions will generate addresses from a list of names provided in DATA statements, and the number of unique and duplicated record addresses will be displayed.

```

10 REM filename: "hashing"
20 REM purpose: To demonstrate hashing function effectiveness
30 REM author: jpg & jdr 10/82(car)
40 REM -----
50 N=40 :: DIM A$(100),R(100),A(100),H$(100),T$(5)
60 OPEN #1:"RS232"
70 FOR I=1 TO N :: READ A$(I):: S=0
80 FOR J=1 TO LEN(A$(I))!sum of ASCII values
90 S=S+ASC(SEG$(A$(I),J,1))
100 NEXT J :: A(I)=S
110 NEXT I
120 FOR I=1 TO 5 :: READ T$(I):: NEXT I
130 FOR I=1 TO N :: R(I)=0 :: H$(I)=" " :: NEXT I :: NU=0 :: U=0
140 PRINT "1 Division" :: PRINT "2 Midsquare" :: PRINT "3 Folding"
150 PRINT "4 Radix transformation" :: PRINT "5 First-last-length"
160 INPUT "What hashing technique ":K :: KK=K
170 ON K GOSUB 1000,2000,3000,4000,5000
180 PRINT #1 :: PRINT #1 :: PRINT #1:T$(KK);" hashing technique"
190 PRINT #1:" *Unique keys";U,"Nonunique keys=";NU
200 PRINT #1:" Number is generated record position"
210 N1=N/4
230 FOR I=1 TO N1
240 J#=SEG$(A$(I),1,9):: PRINT #1:H$(I);J#;
250 P=I+N1 :: J#=SEG$(A$(P),1,9):: PRINT #1:TAB(20);:: PRINT #1:H$(P);J#;
260 P=I+2*N1 :: J#=SEG$(A$(P),1,9):: PRINT #1:TAB(40);:: PRINT #1:H$(P);J#;
270 P=I+3*N1 :: J#=SEG$(A$(P),1,9):: PRINT #1:TAB(60);:: PRINT #1:H$(P);J#
280 NEXT I
290 GOTO 130
1000 !***** Division method of hashing *****
1010 ! H(X)=X MOD M + 1
1020 M=41 !Select M a prime close totable size
1030 FOR I=1 TO N
1040 K=A(I)-INT(A(I)/M)*M !Set remainder MOD M.
1050 GOSUB 6000
1060 NEXT I
1070 RETURN

```

```

2000 !***** Midsquare method *****
2010 FOR I=1 TO N
2020 P=A(I)*A(I)!square the key
2030 P#=STR$(P)
2040 Q#=SEG$(P#,1,2)&SEG$(P#,LEN(P#)-2,2)!use both end digits
2050 K=VAL(Q#)
2060 GOSUB 6000
2070 NEXT I
2080 RETURN
3000 !***** Folding method *****
3010 FOR I=1 TO N :: K=0
3020 P#=STR$(A(I))
3030 FOR J=1 TO LEN(P#)STEP 2
3040 K=K+VAL(SEG$(P#,J,2))!add all digits of key
3050 NEXT J
3060 GOSUB 6000
3070 NEXT I
3080 RETURN
4000 !***** Radix transformation method *****
4010 FOR I=1 TO N :: K=0
4020 P#=STR$(A(I)):: L=LEN(P#)
4030 FOR J=1 TO L !change to base 11
4040 K=K+VAL(SEG$(P#,J,1))*11^(L-J)
4050 NEXT J
4060 K=INT(K+.5)
4070 GOSUB 6000
4080 NEXT I
4090 RETURN
5000 !*** First-last-length method *****
5010 FOR I=1 TO N :: K=0 :: P#=STR$(A(I))
5020 Q#=SEG$(P#,1,2)&SEG$(P#,LEN(P#)-2,2)!set both ends
5030 K=VAL(Q#):: K=K*LEN(P#)!MULTIPLY BY LENGTH
5040 GOSUB 6000
5050 NEXT I
5060 RETURN
6000 !** Subroutine to fill R, H# arrays, tally U,NU **
6010 K=K-INT(K/N)*N+1 !K is hashed key
6020 H$(I)=SEG$(STR$(K)&" ",1,4)
6030 IF R(K)=0 THEN R(K)=INT(K):: H$(I)=" *&H$(I):: U=U+1 ELSE NU=NU+1 :: H$(I)
=" "&H$(I)
6040 RETURN
7000 DATA sithle,beller,anomaly,minderbinder,saliva,hegira
7010 DATA lavoris,pontiac,uvula,gnashgnat,fistula,parsley
7020 DATA grommet,parvenu,power,cuervo,alizuou,monaco
7030 DATA siddhartha,ghautama,asinine,arnold,lancaster,bart
7040 DATA vermiform,veron,rubella,nathaniel,felicity,anthony
7050 DATA andromeda,sibilant,morris,marvin,bandersnatch
7060 DATA cuclone,samantha,marginal,mergerm,annabelle
7070 DATA "Division","Midsquare","Folding"
7080 DATA "Radix transformation","First-last-length"
9999 CLOSE #1 :: END

```

Radix transformation hashing technique

*Unique keys 27 Nonunique keys= 13

Number is generated record position

*20 sithle	*34 fistula	*15 asinine	12 andromeda
*40 beller	2 parsley	11 arnold	28 sibilant
*26 anomaly	*37 grommet	*32 lancaster	*1 morris
*7 minderbin	*3 parvenu	32 bart	25 marvin
*11 saliva	28 power	*21 vermiform	1 bandersna
*33 hegira	33 cuervo	*25 veron	*17 cuclone
*2 lavoris	*12 alizuou	15 rubella	*18 samantha
*23 pontiac	7 monaco	*22 nathaniel	*16 marginal
*28 uvula	*29 siddharth	*31 felicity	*24 mergerm
*30 gnashgnat	*13 ghautama	3 anthony	3 annabelle

Sorting Large Files

Aside from their necessity in binary or interpolation searching, sorted data files are useful in a variety of applications. For this reason, we will demonstrate in the next section of this chapter several procedures for sorting direct access data files.

Disk Sort

The most direct and simple technique for sorting a data file is to access pairs of records and compare and exchange them exactly as if they were array elements in memory. Any sorting algorithm is applicable to this method, but a good binary sort such as the Shell-Metzner is of course more appropriate. Program DACCSORT is a Shell-Metzner sort of a direct access data file called MDATA.

```
10 REM filename: "daccsort"
20 REM purpose: Shell-Metzner sort of a Direct Access file
30 REM author: jpg & jdr 10/82 (car)
40 REM -----
50 RANDOMIZE :: CALL CLEAR
60 OPEN #1:"RS232" :: DIM W$(25),F(25)
70 ! Input sequential data
80 INPUT "How many direct access records to sort ":ND
90 OPEN #2:"DSK1.WORDS",SEQUENTIAL,INTERNAL,INPUT
100 FOR I=1 TO ND :: INPUT #2:W$(I),F(I):: NEXT I
110 CLOSE #2
120 ! Print original version
130 PRINT #1:"Original form of file."
140 GOSUB 1000
150 ! Scramble array and corresponding F locations
160 FOR I=1 TO 50
170 A=1+INT(ND*RND):: B=1+INT(ND*RND)
180 T=W$(A):: W$(A)=W$(B):: W$(B)=T$
190 T=F(A):: F(A)=F(B):: F(B)=T
200 NEXT I
210 ! Print scrambled version
220 PRINT #1 :: PRINT #1:"Scrambled file"
230 GOSUB 1000
240 ! Creat direct access file
250 OPEN #3:"DSK1.MDATA",RELATIVE,INTERNAL,OUTPUT
260 FOR I=1 TO ND
270 PRINT #3:W$(I),F(I)
280 NEXT I
290 CLOSE #3
300 ! Sort file with Shell-Metzner algorithm
310 OPEN #3:"DSK1.MDATA",RELATIVE,INTERNAL,UPDATE
320 N=ND :: M=N
330 M=INT(M/2):: PRINT M
340 IF M=0 THEN 470
350 K=N-M :: J=1
360 I=J
370 L=I+M
380 INPUT #3,REC I-1:W1$,F1
390 INPUT #3,REC L-1:W2$,F2
400 IF F1>F2 THEN 450
410 PRINT #3,REC I-1:W2$,F2
420 PRINT #3,REC L-1:W1$,F1
430 I=I-M
440 IF I>1 THEN 370
450 J=J+1
460 IF J<=K THEN 360 ELSE 330
470 CLOSE #3
```

```

480 ! Input sorted file
490 PRINT #1 :: PRINT #1:"Sorted file" :: PRINT #1
500 OPEN #3:"DSK1.MDATA",RELATIVE,INTERNAL,INPUT
510 FOR I=1 TO ND STEP 2
520 INPUT #3:W1#,F1 :: INPUT #3:W2#,F2
530 PRINT #1:W1#;F1,W2#;F2
540 NEXT I
550 CLOSE #3 :: GOTO 9999
1000 !***** Subroutine to print W# and F arrays.*****
1010 PRINT #1
1020 FOR I=1 TO ND STEP 2
1030 PRINT #1:W$(I);F(I),W$(I+1);F(I+1)
1040 NEXT I
1050 RETURN
9999 CLOSE #1 :: END

```

Original form of file.

```

the 15568      of 9767
and 7638      to 5739
a 5074        in 4312
that 3017     is 2509
i 2292        it 2255
for 1869      as 1853
with 1849     was 1761
his 1732      he 1727
be 1535       not 1496
by 1392       but 1379
have 1344     you 1336
which 1291    are 1222

```

Scrambled file

```

which 1291    be 1535
the 15568     for 1869
of 9767       you 1336
not 1496      have 1344
to 5739       are 1222
was 1761      is 2509
a 5074        i 2292
with 1849     but 1379
in 4312       his 1732
it 2255       and 7638
that 3017     as 1853
he 1727       by 1392

```

Sorted file

```

of 9767      the 15568
and 7638     to 5739
a 5074       in 4312
that 3017    is 2509
i 2292       it 2255
for 1869     as 1853
with 1849    was 1761
his 1732     he 1727
be 1535      not 1496
by 1392      but 1379
have 1344    you 1336
which 1291   are 1222

```

Detached Key Sort

The major difficulty with the above method of sorting is that of time. A sort that takes several minutes in memory can take several hours if performed on disk, as demonstrated in the above program. The reason, of course, is that every fetch of a disk record is roughly a hundred times slower than a fetch from memory. We can use this large difference to our advantage by extracting the keys from the file to be sorted, sorting them in memory, and keeping track of where they are on the original file. Then we rebuild the file in sorted order.

This procedure is called a *detached key sort* and we detail the procedure below while tracing an example. An unsorted file named A.DAT is to be sorted producing the file B.DAT.

Original file A.DAT on disk

Record #	Key	Other
1	J	XYZ
2	B	ABC
3	X	MNO
4	R	PQR
5	A	VWX
6	D	ZAB
7	L	LMN
8	Q	BCD
9	V	RST
10	F	GHI

1. Copy all keys from file A.DAT into memory array A\$. This is just the record's key, not all fields. If the key portion of each record is 20 bytes long, for example a customer name, and the record itself is 255 bytes long, the memory array is less than a tenth the size of the file. With numeric keys the advantage in space savings is even greater.
2. Generate the array R to contain the numbers 1 to N in sequence, where N is the number of records in the file A.DAT (also the number of detached keys in the array A\$). These values represent the record numbers that correspond to each of the keys in A\$.

A\$ (keys)	R (record #s)	Memory arrays before sort
J	1	
B	2	
X	3	
R	4	
A	5	
D	6	
L	7	
Q	8	
V	9	
F	10	

- Sort the array A\$ in memory, making sure that for every exchange of the two elements of A\$, say the P1 and P2 positions, there is also an exchange of the corresponding P1 and P2 positions of the array R.

A\$ (keys)	R (record #s)	Memory arrays after sort
A	5	
B	2	
D	6	
F	10	
J	1	
L	7	
Q	8	
R	4	
V	9	
X	3	

- Create a new file B.DAT, using the contents of the array R as pointers to the original file A.DAT. Thus if R(1) is 5, get the 5th record of A.DAT and copy it into the 1st position of B.DAT; if R(2) is 2, get the 2nd record of A.DAT and copy it into the 2nd position of B.DAT; if R(3) is 6, get the 6th record of A.DAT and copy it into the 3rd position of B.DAT; etc.

New file B.DAT on disk

Record #	Key	Other
1	A	VWX
2	B	ABC
3	D	ZAB
4	F	GHI
5	J	XYZ
6	L	LMN
7	Q	BCD
8	R	PQR
9	V	RST
10	X	MNO

Note that if the resulting sorted file B.DAT is renamed A.DAT (after deleting the original unsorted A.DAT) the overall effect is the same as if you had sorted A.DAT in place. This is a useful technique when you must keep the name of the file constant.

Finally, let us consider a way to sort extremely large files. The procedure combines two previously described techniques, the sort-merge and detached key sorts. In step-by-step fashion, the procedure is as follows:

1. Determine the amount of memory space available for an in-memory sort. Call that free memory space FM. In this example, let's have 50 bytes free. This is unrealistically small, but for our purposes it will serve nicely.
2. Determine the length L, in bytes, of the key field on the file to be sorted. For example, a name field could be as long as 30 bytes, but a numeric field comprising real numbers would be only 4 bytes, and an integer numeric field would be just 2 bytes long. In this example, assume L is 10.
3. Divide the free memory space FM by the key length L to get the number of keys NK that can be stored in memory. FM is 50 and L is 10, so NK is 5. Thus only 5 keys can be stored and sorted in memory at a time.
4. Segment the file A.DAT to be sorted into NS number of segments or blocks of NK records each. Most likely the last block will be shorter than NK records, but no matter. Suppose the file A.DAT has 17 names in it. They would be segmented into 3 blocks of 5 and one block of 2. In this example, let us use single letters of the alphabet to represent the 10-character-long keys.

File A.DAT					
	Key	Other		Key	Other
Block 1	J	...		B	...
	P	...		X	...
	G	...		K	...
	D	...		W	...
	R	...		H	...
Block 2	A	...		S	...
	L	...		Q	...
	V	...			
	N	...			
	F	...			
					Block 3
					Block 4

5. Perform a detached key sort on each one of the NS segments. Store the record pointers that represent the sorted segments into a file called RP.DAT. This file will be comprised of NS blocks of record numbers.

File A.DAT			RP.DAT	
Record #	Key	Other	Record #	Contents
1	J	...	1	4 (points to D)
2	P	...	2	3 (points to G)
3	G	...	3	1 (points to J)
4	D	...	4	2 (points to P)
5	R	...	5	5 (points to R)
6	A	...	6	6 (points to A)
7	L	...	7	10 (points to F)
8	V	...	8	7 (points to L)
9	N	...	9	9 (points to N)
10	F	...	10	8 (points to V)
11	B	...	11	11 (points to B)
12	X	...	12	15 (points to H)
13	K	...	13	13 (points to K)
14	W	...	14	14 (points to W)
15	H	...	15	12 (points to X)
16	S	...	16	17 (points to S)
17	Q	...	17	16 (points to Q)

6. Establish two arrays S1 and S2 to be used as stack pointers to each of the sorted segments in RP.DAT. Each array will be NS elements long. Place the first record number of each of the NS sorted segments into the NS elements of S1. Also place the key from A.DAT that the corresponding element of S1 points to in each element of S2.

Array Position	S1 Contents	S2 Contents
1	4 (points to D in A.DAT)	D
2	6 (points to A in A.DAT)	A
3	11 (points to B in A.DAT)	B
4	17 (points to Q in A.DAT)	Q

7. Scan all positions of S2 to find the smallest. In this example it is the A. Go to the corresponding position of S1 (the 2nd) and use its contents as a pointer to A.DAT. Transfer this record to the next available position of B.DAT.
8. Pop the stack. Get the next record, in RP.DAT in this (the second) block. Now S1(2) is 10. Get record 10 in A.DAT and place its key in S2(2). Go to step 7. S1, S2, and the file B.DAT will look like this as their contents are altered during this merging operation.

Pass	S1	S2	B.DAT
0	4.6.11.17	D.A.B.Q	empty
1	4.10.11.17	D.F.B.Q	A
2	4.10.15.17	D.F.H.Q	A.B
3	3.10.15.17	G.F.H.Q	A.B.D
4	3.7.15.17	G.L.H.Q	A.B.D.F
5	1.7.15.17	J.L.H.Q	A.B.D.F.G
6	1.7.13.17	J.L.K.Q	A.B.D.F.G.H
7	2.7.13.17	P.L.K.Q	A.B.D.F.G.H.J
8	2.7.14.17	P.L.W.Q	A.B.D.F.G.H.J.K
9	2.9.14.17	P.N.W.Q	A.B.D.F.G.H.J.K.L
10	2.8.14.17	P.V.W.Q	A.B.D.F.G.H.J.K.L.N
11	5.8.14.17	R.V.W.QN.P
12	5.8.14.16	R.V.W.SN.P.Q
13	999.8.14.16	ZZ.V.W.SN.P.Q.R
14	999.8.14.999	ZZ.V.W.ZZN.P.Q.R.S
15	999.999.14.999	ZZ.ZZ.W.ZZP.Q.R.S.V
16	999.999.12.999	ZZ.ZZ.X.ZZQ.R.S.V.W
17	999.999.999.999	ZZ.ZZ.ZZ.ZZR.S.V.W.X

Notice that as each block or segment of the file A.DAT is used up, the stack pointers S1 and S2 are plugged with signal values 999 and ZZ so that these segments aren't used beyond their limit.

This implementation of a file sort is sufficiently general that it can be applied to any direct access file with any possible key, whether it is numeric or string. Note that at the very beginning, when the program must determine the amount of free memory space, a certain degree of leeway must be allowed to permit the dimensioning of the stack pointers.

ISAM File Processing

There exists a popular form of direct access file processing that is more commonly known by another name: *Index Sequential Access Method*, or *ISAM*. Although this technique is on the surface only a variation of direct access processing, it has several features that tend to make it appropriate for large file management, particularly when the file is spread out over many disks.

Before we describe this technique in detail, remember that there are some applications for which sequential files are not only appropriate but desirable. For example, a sequential file is advantageous if its records are already in their order of access.

The real advantage of ISAM appears only on very large files, in which a particular sub-file is accessed first, then a specific area of that sub-file, and finally just one or a small sequence of records from that specific area. On a TI-99/4A, such a hierarchical system of access might exist if the system has multiple drives. Some vendors of compatible software may supply ISAM file capability, but we have no familiarity with any such implementation.

The single characteristic of ISAM that makes it popular is that it allows both sequential and direct access processing. You can start sequential processing at the beginning of any file or any other record in

the file. To perform a direct access, you can specify a key-field value and the ISAM system fetches the appropriate record. Once you have that record, you can start sequential file processing if you wish.

ISAM Storage Areas

Indexed sequential files are composed of three areas of storage: (1) The *prime area* contains the records that were written onto the file at the time of its creation. (2) The *overflow area* contains the added records that won't fit into the prime area. (3) The *index area* contains pointers to particular segments of the file.

We will discuss the structure of ISAM files for a TI-99/4A (or for any other disk-oriented microcomputer, for that matter) at one elementary level of complexity, in which the system uses a single disk drive with unblocked records.

DOS Physical Characteristics

The most straightforward way to build an ISAM file structure is to use a single disk and unblocked records. Consider the layout of the IBM PC minidisk. It is arranged in this descending order of physical size:

1 disk = 40 tracks (or 80 tracks on 320 KB systems)
1 track — eight sectors
1 sector = 512 bytes

DOS files can be thought of in logical rather than physical terms, though, and this is the way the operating system deals with them:

1 disk = 1 to m files
1 file = 1 to n clusters
1 cluster = 1 to 40 physically contiguous tracks
1 track = ten sectors
1 sector = 1 record

The most important part of the description above is the phrase "physically contiguous tracks". This means that a cluster of 16 tracks of a file is laid out in track-to-track physical order, say from tracks 10 to 25. The advantage of this kind of organization is that access time to successive physical records is minimized. But this advantage exists only if the user's requests are logically arranged in the same order as the file's physical layout. Such a condition exists when a sequential file is created on disk, because the DOS will write the successive sequential records on successive sectors and tracks. This condition speeds up sequential access considerably, because the disk drive's read-write head need not return to a "home" position and can easily move to the next track.

ISAM Structuring

Knowing all of these physical details is not only nice but necessary when you are implementing an ISAM-structured file. Let's use these facts now to build an idealized ISAM file.

Consider these important preconditions:

1. The entire file will be limited to a single segment of 16 tracks, starting at Track 10 and ending at Track 25.
2. Your initial data is made up of 100 records, each record consisting of name (key), address, city-state-zip, and phone

number. Although each record is FIELDed to use an entire 512 byte sector, it occupies only 120 bytes of the sector. We won't worry about this waste in this application, because we don't want to complicate things with blocking factors.

3. The prime area will be tracks 11 to 20, with each of the sectors storing a record.
4. Tracks 21 to 25 will be kept in reserve as the overflow area for future growth of the file.
5. Track 10 will hold the index to the file.

The following table shows what the file might look like before the overflow area is used. We show you only the first three letters of each key for clarity.

Sectors	1	2	3	4	5	6	7	8	9	10
Tracks 11	abi	abl	aca	ach	acl	acr	act	adm	ali	als
12	ana	ani	avo	bri
13	bro	dan
14	dar	eep
..
19	tho	thr	van
20	vel	ver	zam

Track 10 (the index track) contains the highest key value on each track.

Track 10 11-als 12-bri 13-dan 14-eep 15-gim 16-hol
17-kni 18-plu 19-van 20-zam

ISAM Access

To locate the track that contains the key "lim", for example, you need only find the first track that contains a record with a key greater than "lim". A sequential scan of Track 10 locates 18-plu, so Track 18 either has the key "lim" or that key is not on the file. Notice that using the track index as a shortcut to the file doesn't eliminate a sequential search. Rather, it reduces the amount of sequential searching in this case to only two tracks, the index (Track 10), and the record track (Track 18).

The process for building this structure is somewhat involved. By now you have discovered a universal property of programming: The convenience of a good data structure is bought at the price of programming complexity. Consider the steps involved in building the initial ISAM prime and index areas:

1. Sort the initial file of 100 records. This is necessary because the prime area must start in sorted order.
2. Fetch the first 10 records and copy them in sorted order onto Track 11. Copy the last key into Track 10 as the first index entry.

- Fetch the remaining records in groups of 10, and copy them onto successive Tracks 12 to 20. Copy the last key of each group in succession onto Track 10 as the rest of the index entries.

If you are lucky enough to have had your file of 100 records in sorted order on a direct access file, all you really have to do is to read them 10 at a time and grab the last one in each group of 10 as the highest key on the track.

Overflow Area

Once you have built the prime and index areas, you need to consider the *overflow area*, into which all additional records will be placed. You have reserved this area in the physical location of Tracks 21 through 25, so you should be able to allow a growth of 50 records for the file. After that, you're on your own.

The overflow area in essence is a linked list. If the key being sought is not in its proper track in the prime area, the index entry to that track points to a record in the overflow area. But remember that you must keep all of the records in sequential order on each track of the prime area. Otherwise you defeat the purpose of an ISAM search.

ISAM Insertion

To illustrate the complexity of insertion, consider the previously described prime area. Suppose you need to add the key "alm" to the file. The only way to preserve key sequence in Track 11 is to have "alm" take the place of "als" on that track. But then what happens to "als"? You surely wouldn't want to move all 90 records on Tracks 12-20 down one position just to fit "als" in its place. What you must do is place "als" into the overflow area, and indicate that fact on the track index. Therefore the index must contain, besides the 10 high keys on each track, some kind of link to the overflow area. Before the insertion of "alm", the index looks like the example below:

Index record (Track 10):

Entry	Contents Highest Key on Track	Contents Highest Key on Overflow	Overflow Link
1	als	als	null
2	bri	bri	null
3	dan	dan	null
.	.	.	.
.	.	.	.
9	van	van	null
10	zam	zam	null

After the insertion of the new record with the key of "alm", the index entry 1 changes to this:

1	alm	als	null
---	-----	-----	------

After yet another insertion, say of a record with key "abr", the index entry 1 becomes:

1	ali	als	alm
---	-----	-----	-----

What happened? Where are these records? If the highest on the track is 'ali', the track must contain

abi abl abr aca ach acl acr act adm ali

Where are "alm" and "als"? The hint is in the contents of the overflow link. If you access the record with the key "alm", you will reach that record in the overflow area. It will contain a link to the "als" record, which will indicate completion of the linked list with a null link, like this:

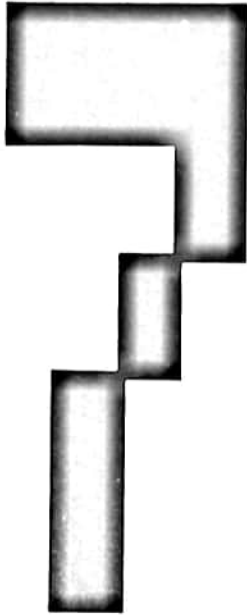
Overflow area:	
Key	Link
alm	als
als	null)

What a hassle to insert a record!

1. Find the right track using the index.
2. Shift all higher records down the track to insert the new one.
3. Transfer the overflow record to the overflow area.
4. Adjust all links accordingly.

In order to reduce the burden of insertion processing, many ISAM systems plan ahead by leaving some empty space on each track so that the overflow area is not impacted quite as quickly. Also, the periodic complete reformatting of an ISAM file reduces the overhead of insertion and also increases file access speed. The advantages of ISAM are apparent if your application requires the access or display of the records in key-sorted order, or if you need to access individual records in random order. Programming overhead is rather high, but the speed of response is a distinct improvement over sequential files for single-record access, and direct access files for sorted-order display.

In the next chapter we will introduce another method of file access and search which, like ISAM, takes advantage of record links and direct access files. These are the tree structures.



Trees

You will remember from Chapter 4 that when pointers are incorporated into the data, such as in linked lists, there is a certain amount of overhead in the form of space used for the links. The convenience of having a pointer to an associated record comes at the cost of space used to store that pointer. When doubly linked circular lists are used, the space used by the links is even greater, but still there are occasions when these links are so useful and necessary that their storage is a small price to pay for the convenience they lend to the data structure.

The association of records through links is often based on some form of binary logic. For example, a field in record B is larger, or smaller, than the corresponding field in record A; or perhaps the relationship is one of inclusiveness, such that record B represents a subset of record A, or vice versa. When a series of records can be arranged hierarchically in the form of a pyramid, so that each record has one or more records under it (unless it is on the bottom layer of the pyramid), the arrangement is called a *tree structure*.

Tree structures use links as integral parts of their data elements, just as linked lists do, except that the links point to subordinate records in the tree. In some cases tree structure links can point to records above them in the hierarchy, but we will not consider these in this chapter. Also, some tree structures, called *trinary trees* or *tries*, allow more than two links to point to subordinate records. We will not consider these either; rather we will limit our discussion of trees to the structures called *binary trees*.

The overhead cost of maintaining links in a binary tree is of course directly proportional to the number of links that the tree must use in order to provide the necessary branches. For example, simple binary trees can have two links for each of several fields in each record. We will endeavor to show you some generalized applications of trees in the form of easily modifiable programs.

The first program is an isomorph of two other programs: ANIMAL, found in the *Systems Applications* volume of this series, and GEOGRAPH, found in the series book, *Techniques of BASIC*. Both of those programs were based on the generalized tree structure shown in Figure 7.1.

The tree structure is based on a series of binary (YES or NO) relationships of the included characteristic. A YES answer to characteristic-1 produces guess-1. If that isn't the sought after element, characteristic-2 is displayed, and the program proceeds through the left subtree. A NO answer to characteristic-1 produces a display of characteristic-3 and subsequent branching through the right subtree.

Figure 7.2 shows what the structure might look like for the ANIMAL game.

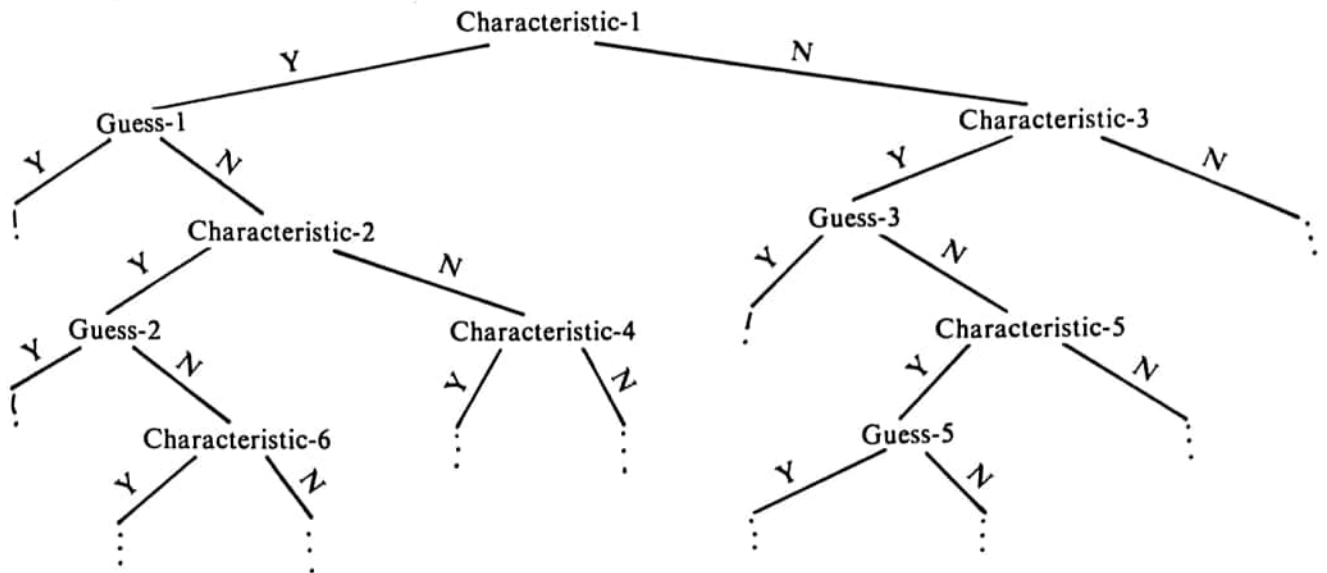


Figure 7.1 Generalized Tree Structure for Binary Relationships

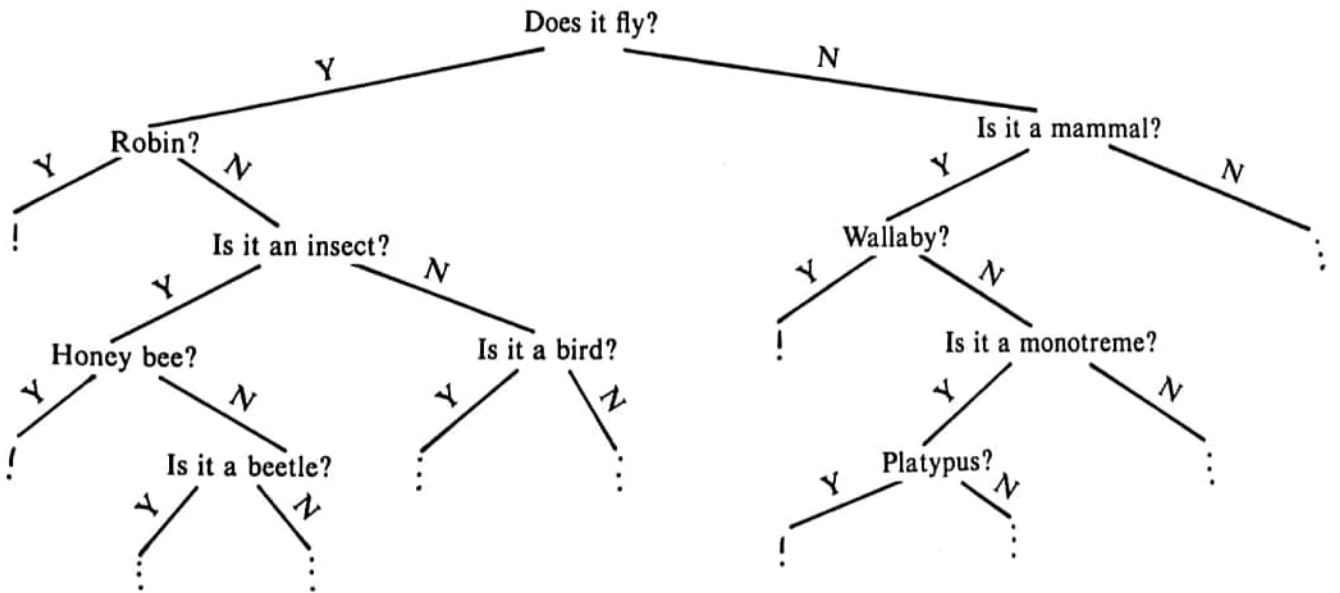


Figure 7.2 Tree Structure for ANIMAL Game

A possible dialogue generated by an interaction with the ANIMAL game could be:

```

Computer: Are you thinking of an animal?
User:     Yes
Computer: Does it fly?
User:     Yes
Computer: Is it a robin?
User:     No
Computer: Is it an insect?
User:     No
Computer: Is it a bird?
User:     Yes
      .       .
      .       .
      .       .
  
```

Notice that as the dialogue continues, the computer continually narrows down the area of the search, based on the user's YES or NO answers. If the answer is YES to a characteristic, the computer guesses the animal associated with that characteristic. If that isn't the animal the user has in mind, the computer traverses the YES sub-branch. If the answer to a displayed characteristic is NO, the computer traverses the NO sub-branch, without bothering to guess that characteristic's associated animal.

This form of a binary tree is easy to follow, relatively easy to program, and generalizable to a wide variety of applications. For instance, a GEOGRAPHY program could be used to entertain waiting customers in a travel agency office. The customer thinks of a geographical location and tries to stump the machine. Or what if a veterinarian had such a program in the waiting room? The client could test his or her knowledge of pets, perhaps serving as a mild distraction while the family beast is in for alterations.

This program's main attraction is its flexibility. With the change of a few lines, it looks like an entirely different program serving a completely different group of users. Consider a few of the other applications possible with this program: A bookstore could have an Authors or Titles program; an unemployment office or job placement office could modify the program to list occupations; a new or used car sales lot could record the makes of cars; or a doctor's waiting room could list diseases. Many other possibilities exist, all based on three general objectives:

1. Involve the user with the computer.
2. Teach the user about the field in question.
3. Increase and improve the data file with user responses.

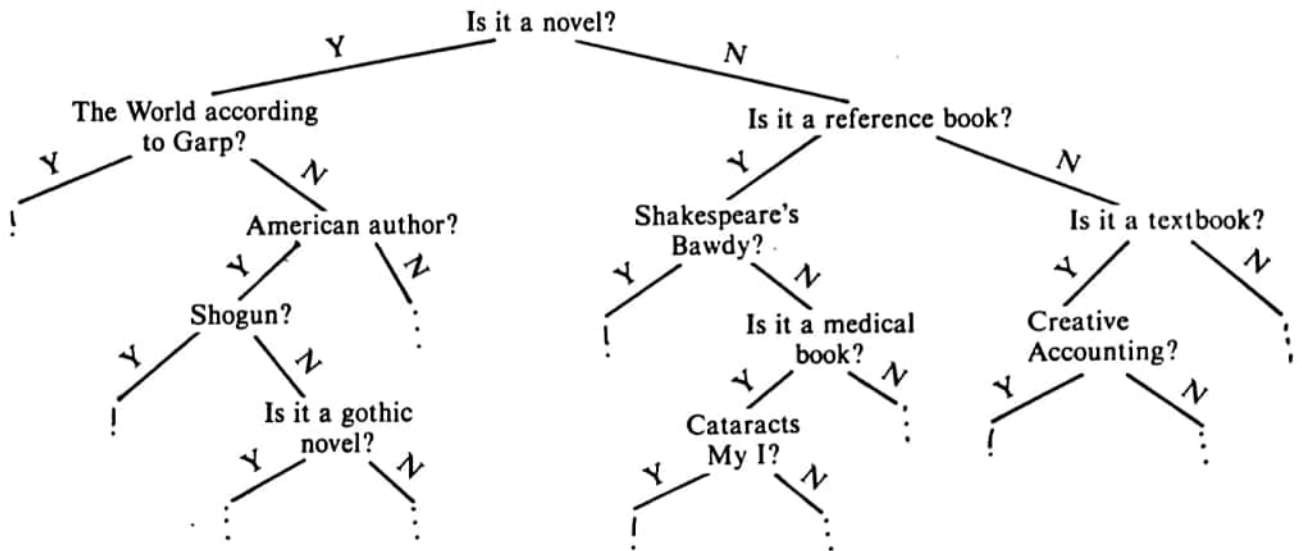


Figure 7.3 Tree Structure for an ANIMAL-type program based on books

The program VEGGIES that follows is yet another version of the ANIMAL game, but it is concerned with the vegetable kingdom. The following dialogue is a typical interaction with VEGGIES.

```

Computer: Are you thinking of a vegetable?
User:     Yes
Computer: Is it a tree?
User:     No
Computer: What vegetable were you thinking of?
User:     No
Computer: What vegetable were you thinking of?
User:     Ragweed
Computer: What characteristic distinguishes a ragweed from a
          Norway maple?
User:     Is it a weed?
Computer: Are you thinking of a vegetable?
User:     Yes
Computer: Is it a tree?
User:     No
Computer: Is it a weed?
User:     Yes
Computer: Is it a ragweed?
User:     No
Computer: What vegetable were you thinking of?
User:     Purslane
Computer: What characteristic distinguishes a puslane from a
          ragweed?
User:     Is it edible?
          .
          .
          .
    
```

The data file called VEGGIES.DAT in the program VEGGIES could look like the table below after several additions by the user.

Rec. #	L1	L2	A1\$	Q\$
1	4	2	Norway maple	Is it a tree
2	3	5	Ragweed	Is it a weed
3	6	999	Purslane	Is it edible
4	4	999	Scotch pine	Is it an evergreen
5	5	999	Pole bean	Is it a garden vegetable
6	6	999	Poke weed	Does it have berries

The variable names are those that are used in the program. When L1, the YES link, is the same as the record number, the tree has no further information on this characteristic beyond this record. If the L1 link is not its own record number, it means that there exists one or more further records under the YES branch. The L2, or NO link, also has two forms. If the NO link is a 999, there are no records under the NO link of this characteristic. If the NO link is not a 999 it points to the record which contains a continuation of the tree in the NO direction.

We include the listing and some typical dialogue to show you how this game progresses. Of course since most of the fun of this program is building the file, we leave that up to you. Remember, though, to select as general a characteristic as possible, and as specific a vegetable as possible.

```

100 REM filename: "veggies"
110 REM purpose: Quis game "VEGETABLES"
120 REM author: jpg & jdr 10/82 (car)
130 REM -----
140 !Q%=characteristic, A1%=vegetable title,A%=temporary string
150 !L1=leftlink, L2=right link, N=number of vegetables
160 DIM Q$(50),A1$(50),L1(50),L2(50)
170 CALL CLEAR
180 ! Set up first record if needed
190 INPUT "Does the data file exist ":A$
200 IF A$<>"YES" THEN Q$(1)="IS IT A TREE " :: L1(1)=1 :: L2(1)=999 :: N=1 :: A1
$(1)="NORWAY MAPLE" :: GOTO 290
210 ! Get all filed vegetables
220 OPEN #2:"DSK1.VEGES",SEQUENTIAL,INTERNAL,INPUT :: I=1
230 IF EOF(2)THEN 250 :: INPUT #2:Q$(I),L1(I),A1$(I),L2(I):: PRINT I;:: I=I+1
240 GOTO 230
250 N=I-1 :: CLOSE #2 :: INPUT "<<ENTER>>":A$
260 IF F>50 THEN PRINT "C A REFUL! MORE THAN 50!"
270 !
280 ! Next place
290 CALL CLEAR :: INPUT "ARE YOU THINKING OF A VEGETABLE ":A$
300 K=0
310 IF A$="LIST" THEN K=1
320 IF A$="YES" THEN K=2
330 IF A$="DEBUG" THEN K=3
340 IF A$="SAVE" THEN K=4
350 IF K=0 THEN 380
360 ON K GOSUB 1000,2000,5000,6000
370 IF K>0 AND K<5 THEN 290
380 INPUT "TYPE YES, LIST, SAVE, OR DEBUG THEN <<ENTER>>":A$ :: GOTO 300
1000 !***** LIST ROUTINE*****
1010 PRINT "THE VEGETABLES I KNOW ARE..."
1020 FOR I=1 TO N :: PRINT A1$(I),: NEXT I :: PRINT
1030 INPUT " <<ENTER>>":A$ :: RETURN
2000 !***** YES, the player is thinking of a vegetable.
2010 I=1
2020 PRINT Q$(I);: INPUT A$
2030 IF A$="YES" THEN 2110
2040 IF A$<>"NO" THEN 2020
2050 ! NO, not the listed characteristic so if right link
2060 ! is null, use it--otherwise get new veg's details
2070 IF L2(I)<>999 THEN I=L2(I):: GOTO 2020 ELSE GOSUB 3000
2080 ! Set up all links and get a new vegetable
2090 L2(I)=N+1 :: GOSUB 4000 :: RETURN
2100 ! Ask if this is the correct characteristic
2110 PRINT "IS IT ";A1$(I);: INPUT A$
2120 IF A$<>"YES" THEN 2160
2130 ! Yes, the computer guessed it (print the message)
2140 PRINT :: PRINT :: PRINT "...I THOUGHT SO."
2150 FOR I=1 TO 400 :: NEXT I :: RETURN
2160 IF A$<>"NO" THEN 2110
2170 ! Get the next vegetable if link isn't null
2180 IF I<>L1(I)THEN I=L1(I):: GOTO 2020
2190 ! Get the ne veg's details
2200 GOSUB 3000
2210 ! Set up all new links and get next vegetable
2220 L1(I)=N+1 :: GOSUB 4000 :: RETURN

```

```

3000 !***** DIALOG *****
3010 INPUT "WHAT WAS THE VEGETABLE YOU WERE THINKING OF ":A$
3020 PRINT "TYPE A CHARACTERISTIC THAT WOULD DISTINGUISH"
3030 PRINT A$;" FROM ";A1$(I):: INPUT Q1$ :: RETURN
4000 !*****New veteable links *****
4010 N=N+1 :: Q$(N)=Q1$ :: A1$(N)=A$ :: L1(N)=N :: L2(N)=999 :: RETURN
5000 !***** DEBUG routine--displays all links, data ***
5010 PRINT "I  Q$(I)  A1$(I)  L1(I)  L2(I)"
5020 FOR I=1 TO N :: PRINT I;Q$(I),A1$(I),L1(I);L2(I):: NEXT I
5030 INPUT "<<ENTER>>":A$ :: RETURN
6000 !***** SAVE routine *****
6010 PRINT "REC.#";:: OPEN #2:"DSK1.VEGES",SEQUENTIAL,INTERNAL,OUTPUT
6020 FOR I=1 TO N
6030 PRINT #2:Q$(I),L1(I),A1$(I),L2(I)
6040 PRINT I;:: NEXT I
6050 CLOSE #2
6060 INPUT "<<ENTER>>":A$ :: RETURN
9999 END

```

```

ARE YOU THINKING OF A VEGETABLE
YES
IS IT A TREE
YES
IS IT NORWAY MAPLE
NO
WHAT WAS THE VEGETABLE YOU WERE THINKING OF
SCRUB OAK
TYPE A CHARACTERISTIC THAT WOULD DISTINGUISH
SCRUB OAK FROM NORWAY MAPLE
IS IT SHORT AND SCRUFFY

```

```

ARE YOU THINKING OF A VEGETABLE
YES
IS IT A TREE
NO
WHAT WAS THE VEGETABLE YOU WERE THINKING OF
TOMATO
TYPE A CHARACTERISTIC THAT WOULD DISTINGUISH
TOMATO FROM NORWAY MAPLE
IS IT RED AND SQUISHY

```

```

ARE YOU THINKING OF A VEGETABLE
YES
IS IT A TREE
YES
IS IT NORWAY MAPLE
NO
IS IT SHORT AND SCRUFFY
NO
WHAT WAS THE VEGETABLE YOU WERE THINKING OF
BIGTOOTH ASPEN
TYPE A CHARACTERISTIC THAT WOULD DISTINGUISH
BIGTOOTH ASPEN FROM SCRUB OAK
IS IT INDIGENOUS TO COLORADO

```

```

ARE YOU THINKING OF A VEGETABLE
NO
TYPE YES, LIST, SAVE, OR DEBUG THEN <<ENTER>>

```

```

THE VEGETABLES I KNOW ARE...
NORWAY MAPLE SCRUB OAK      TOMATO      BIGTOOTH ASPEN
<<ENTER>>

```

```

ARE YOU THINKING OF A VEGETABLE
DEBUG
I  Q$(I)  A1$(I)  L1(I)  L2(I)
IS IT A TREE  NORWAY MAPLE  2  3
IS IT SHORT AND SCRUFFY  SCRUB OAK  2  4
IS IT RED AND SQUISHY  TOMATO  3  999
IS IT INDIGENOUS TO COLORADO  BIGTOOTH ASPEN  4  999
<<ENTER>>
DEBUG
ARE YOU THINKING OF A VEGETABLE
NO
TYPE YES, LIST, SAVE, OR DEBUG THEN <<ENTER>>

```

```

REC.# 1 2 3 4 ARE YOU THINKING OF A VEGETABLE
YES
IS IT A TREE
YES
IS IT NORWAY MAPLE
NO
IS IT SHORT AND SCRUFFY
YES
IS IT SCRUB OAK
YES

```

...I THOUGHT SO.

The binary tree which is represented by the VEGGIES program is useful as a data management technique for accessing a specific record, or for adding a new record to an existing tree. The algorithm is the same both for access and for addition. But notice that any traversal of the tree, that is any scheme for visiting each record in the tree, does not yield any significant ordering.

Binary Sequence Search Tree

There is a tree structure, however, that yields significant ordering upon its traversal, and that is the *binary sequence search tree*, or *BSST*. You have seen this structure before as the tree sort in Chapter 2. Indeed, the sorted order resulting from the traversal of the BSST turns out to be one of its most useful properties.

The remainder of this chapter will show various BSST structures to indicate its flexibility in a wide selection of settings. We will not discuss the theory behind the tree's properties, however, because the *Systems Applications* book in this series does so in exhaustive detail. You should refer to that volume to understand these aspects of the BSST:

1. Building, accessing, and deleting algorithms: How does a programmer code the procedure for adding new nodes, or records, or for fetching a specific record, or for removing an existing record?
2. Traversal algorithm: What is the process for accessing the tree's records in sorted order?
3. Balancing algorithm: If the tree suffers from an imbalance, that is, significantly more records fall on one sub-branch of the tree than on another, how does the programmer restructure the tree to make it symmetric?

In-Memory, Single-Key BSST

The first program of this section, STRBSST, provides a straightforward example of a program that uses the BSST. The program accesses a sequential file (the same 100 most common words file used previously) and upon each record access, adds to an in-memory tree with data and links, each represented as an array. Remember that the records in this file have two fields: The first field is the common word, not in alphabetical order but rather in order of its frequency of use; the second field is that common word's frequency of occurrence in roughly 250,000 words of English text. The frequency field is in descending order of occurrence.

The purpose of STRBSST is to transfer the two fields to memory (words into W\$, frequencies into F), to build the tree based on W\$ as a key by creating appropriate links (left link is LL, right link is RL), then to traverse the in-memory tree in sorted order. The traversal uses a stack S to keep track of previously hit words so that they can be displayed in sorted order in their proper turn. We include a display of the tree's structure record by record as well as the traversal's sorted output. If you study the first few words, their associated left and right links, and the program listing, you should be able to grasp the essentials of the BSST's properties.

```
10 REM filename: "strbsst"
20 REM purpose: String-Key in-memory BSST
30 REM author: jpg & jdr 10/82 (car)
40 REM -----
50 DIM W$(100),LL(100),RL(100),S(100)'S is the stack
60 ! Get strings from "DSK1.WORDS" sequential file
70 OPEN #2:"DSK1.WORDS",SEQUENTIAL,INTERNAL,INPUT
80 ! Initialize E (end pointer), first word W$
90 E=1 :: W$(1)=" "
100 CALL CLEAR :: INPUT "How many words from file":N
110 FOR K=1 TO N :: INPUT #2:N1$,X :: I=1
120 IF N1$>W$(I)THEN 150
130 IF LL(I)<>0 THEN I=LL(I):: GOTO 120 'Go left
140 LL(I)=E :: GOTO 170
150 IF RL(I)<>0 THEN I=RL(I):: GOTO 120 !Go right
160 RL(I)=E
170 W$(E)=N1$ :: LL(E)=0 :: RL(E)=0 :: E=E+1
180 NEXT K
190 ! Tree is built. Now traverse it in sorted order
200 P=1 :: T=0
210 T=T+1 :: S(T)=P
220 IF P<>0 THEN P=LL(P):: GOTO 210 !Traverse left
230 IF T=0 THEN 260
240 T=T-1 :: P=S(T)
250 IF W$(P)<>" " THEN PRINT W$(P),: T=T-1 :: P=RL(P):: GOTO 210
260 ! Print table of words, their links
270 PRINT
280 INPUT "<<ENTER>> to see table of data and links":A$ :: CALL CLEAR
290 PRINT "Word";TAB(6);"Left link Right link"
300 FOR I=1 TO N
310 IF INT(I/22)*22=I THEN PRINT "<<ENTER>>": INPUT A$ :: CALL CLEAR
320 PRINT W$(I);TAB(6);LL(I);TAB(19);RL(I)
330 NEXT I
340 CLOSE #2 :: END
```


How many words from file

```
25
a          and          are          as          be          but
by         for          have        he          his         i
in         is           it          not         of          on
that      the          to          was        which       with
you
<<ENTER>> to see table of data and links
```

Word	Left link	Right link
the	2	4
of	3	7
and	5	6
to	0	13
a	0	0
in	9	8
that	25	0
is	0	10
i	11	0
it	0	18
for	12	15
as	24	17
with	14	22
was	0	23
his	16	0
he	21	0
be	0	19
not	0	0
by	20	0
but	0	0
have	0	0
<<ENTER>>		
you	0	0
which	0	0
are	0	0
on	0	0

In-Memory, Double-Key BSST

Program DBLKEY is a significant extension of the above BSST example. Its DIM statements include space for two new links, L1 and L2, which will hold left and right links for the second data field in the record, that word's frequency of occurrence. We have purposely scrambled the data in WORDS so that both the words and their frequencies are out of order. If we had not scrambled the file, all right links L2 would have been 0, and all left links L1 would have been one more than the record number. The tree, in tabular form, would have looked like this:

F	L1	L2
15577	2	0
8420	3	0
5671	4	0
4257	5	0
.	.	.
.	.	.
.	.	.

This condition, a tree with all right branches or all left branches, is called a *degenerate tree*. Its utility is no better than that of an ordered sequential file for access purposes. That is why the WORDS file that we use in DBLKEY is scrambled.

```

10 REM filename: "dblkey"
20 REM purpose: Double-Key in-memory BSST
30 REM author: jpg & jdr 8/82 (car)
40 REM -----
50 DIM W$(100),LL(100),RL(100),S(100)!S is the stack
60 DIM L1(100),L2(100),F(100)!Links for frequency F
70 CALL CLEAR
80 INPUT "How many entries from file(to 100)":N
90 ! Scramble DSK1.WORDS into DSK1.WORDS2
100 GOSUB 4000
110 ! Get strings from "DSK1.WORDS" sequential file
120 OPEN #1:"DSK1.WORDS",SEQUENTIAL,INTERNAL,INPUT
130 ! Initialize E (end pointer), first entries W$,F
140 E=1 :: W$(1)=" " :: F(1)=0
150 ! Print the entries
160 FOR K=1 TO N :: INPUT #1:N1$,X :: I=1 :: J=1
170 PRINT K;X;N1$
180 ! First, build the words tree
190 IF N1$>W$(I)THEN 220
200 IF LL(I)<>0 THEN I=LL(I):: GOTO 190 !Go left
210 LL(I)=E :: GOTO 250
220 IF RL(I)<>0 THEN I=RL(I):: GOTO 190 !Go right
230 RL(I)=E
240 ! Now take care of frequency tree
250 IF X<F(J)THEN 280
260 IF L1(J)<>0 THEN J=L1(J):: GOTO 250
270 L1(J)=E :: GOTO 300
280 IF L2(J)<>0 THEN J=L2(J):: GOTO 250
290 L2(J)=E
300 W$(E)=N1$ :: LL(E)=0 :: RL(E)=0
310 F(E)=X :: L1(E)=0 :: L2(E)=0 :: E=E+1
320 NEXT K
330 PRINT
340 ! Tree is built. Now traverse it in sorted order.
350 ! Determine which activity.
360 PRINT :: PRINT
370 PRINT "<<<ENTER>> to continue";: INPUT A$ :: CALL CLEAR
380 PRINT "0=Halt"
390 PRINT "1=Display words in sorted order."
400 PRINT "2=Display frequency in sorted order."
410 PRINT "3=Display table of entries and links."
420 PRINT "Which activity";: INPUT K
430 IF K=0 THEN CALL CLEAR :: GOTO 9999
440 ON K GOSUB 1000,2000,3000
450 GOTO 360
1000 P=1 :: T=0
1010 T=T+1 :: S(T)=P
1020 IF P<>0 THEN P=LL(P):: GOTO 1010 !Traverse left
1030 IF T=0 THEN PRINT :: RETURN
1040 T=T-1 :: P=S(T)
1050 IF W$(P)<>" " THEN PRINT W$(P),: T=T-1 :: P=RL(P):: GOTO 1010
1060 PRINT :: RETURN
2000 ! Traverse tree for frequencies
2010 P=1 :: T=0
2020 T=T+1 :: S(T)=P
2030 IF P<>0 THEN P=L1(P):: GOTO 2020 !Traverse left
2040 IF T=0 THEN PRINT :: RETURN
2050 T=T-1 :: P=S(T)
2060 IF F(P)<>0 THEN PRINT F(P),: T=T-1 :: P=L2(P):: GOTO 2020
2070 PRINT :: RETURN

```

```

3000 ! Print table of words, their links
3010 ! from memory
3020 PRINT "Word";TAB(6);"LL";TAB(12);"RL";
3030 PRINT TAB(15);"Freq.";TAB(22);"L1";TAB(26);"L2"
3040 FOR I=1 TO N
3050 IF INT(I/16)*16=1 THEN PRINT "<<ENTER>>" :: INPUT A$ :: CALL CLEAR
3060 PRINT W$(I);TAB(6);LL(I);TAB(11);RL(I);
3061 PRINT TAB(15);F(I);TAB(22);L1(I);TAB(26);L2(I)
3070 NEXT I
3080 RETURN
4000 !***** Scramble DSK1.WORDS into DSK1.WORDS2 *****
4010 OPEN #1:"DSK1.WORDS",SEQUENTIAL,INTERNAL,INPUT
4020 OPEN #2:"DSK1.WORDS2",SEQUENTIAL,INTERNAL,OUTPUT
4030 FOR I=1 TO N :: INPUT #1:W$(I),F(I):: NEXT I :: CLOSE #1
4040 ! Scramble
4050 FOR I=1 TO N/2 :: A=1+INT(N*RND):: B=1+INT(N*RND)
4060 T$=W$(A):: T=F(A)
4070 W$(A)=W$(B):: F(A)=F(B)
4080 W$(B)=T$ :: F(B)=T
4090 NEXT I
4100 FOR I=1 TO N :: PRINT #2:W$(I),F(I):: NEXT I ::
4110 CLOSE #2 :: RETURN
9999 END

```

How many entries from file(to 100)

```

25
1 15568 the
2 9767 of
3 7638 and
4 5739 to
5 5074 a
6 4312 in
7 3017 that
8 2509 is
9 2292 i
10 2255 it
11 1869 for
12 1853 as
13 1849 with
14 1761 was
15 1732 his
16 1727 he
17 1535 be
18 1496 not
19 1392 by
20 1379 but
21 1344 have
22 1336 you
23 1291 which
24 1222 are
25 1155 on

```

<<ENTER>> to continue

0=Halt
 1=Display words in sorted order.
 2=Display frequency in sorted order.
 3=Display table of entries and links.

Which activity 1

a	and	are	as	be	but
by	for	have	he	his	i
in	is	it	not	of	on
that	the	to	was	which	with
you					

<<ENTER>> to continue

0=Halt
 1=Display words in sorted order.
 2=Display frequency in sorted order.
 3=Display table of entries and links.

Which activity 2

15568	9767	7638	5739	5074	4312
3017	2509	2292	2255	1869	1853
1849	1761	1732	1727	1535	1496
1392	1379	1344	1336	1291	1222
1155					

<<ENTER>> to continue

0=Halt
 1=Display words in sorted order.
 2=Display frequency in sorted order.
 3=Display table of entries and links.

Which activity 3

Word	LL	RL	Freq.	L1	L2
the	2	4	15568	0	2
of	3	7	9767	0	3
and	5	6	7638	0	4
to	0	13	5739	0	5
a	0	0	5074	0	6
in	9	8	4312	0	7
that	25	0	3017	0	8
is	0	10	2509	0	9
i	11	0	2292	0	10
it	0	18	2255	0	11
for	12	15	1869	0	12
as	24	17	1853	0	13
with	14	22	1849	0	14
was	0	23	1761	0	15
his	16	0	1732	0	16
he	21	0	1727	0	17
be	0	19	1535	0	18
not	0	0	1496	0	19
by	20	0	1392	0	20
but	0	0	1379	0	21
have	0	0	1344	0	22
you	0	0	1336	0	23
which	0	0	1291	0	24
are	0	0	1222	0	25
on	0	0	1155	0	0

<<ENTER>> to continue

**In-Memory,
Multi-Key BSST**

The next program, TREE5KEY, exemplifies a multi-key BSST in memory. The data is simply 2-digit random integers between 10 and 99, but it is arranged in a two-dimensional array X(20,5). There are up to five lists of random integers, each list being up to 20 items long. Each list has associated with it two lists of links, such that each list can be accessed in sorted order. The two links for each element of the five sets of twenty elements are in the triply dimensioned array L(20,5,2). L(A,B,1) is the left link and L(A,B,2) is the right link, corresponding to each of the 100 elements X(A,B).

```
10 REM filename: "tree5key"
20 REM purpose: To build and traverse a 5-key BSST in mem.
30 REM author: jpg & jdr 80/82 (car)
40 REM -----
50 DIM X(20,5),L(20,5,2),S(20)
60 CALL CLEAR
70 INPUT "How many keys per record (to 5)":M
80 IF M=0 THEN GOTO 9999
90 PRINT "How many records (to 20)" :: INPUT N
100 FOR I=1 TO N
110 FOR J=1 TO M
120 X1=10+INT(90*RND)! generate key, 10 to 99
130 P=1 ! set root pointer to 1
140 GOSUB 4000 ! generate left and right links
150 X(I,J)=X1 ! store this element
160 L(I,J,1)=0 :: L(I,J,2)=0 ! zero its link
170 NEXT J :: NEXT I
180 GOSUB 200 ! determine output from user
190 PRINT :: PRINT :: PRINT "<<ENTER>> for more" :: INPUT A$ :: CALL CLEAR :: GO
TO 70
200 ! Produce output on demand
210 PRINT "Type tables (y or n" :: INPUT A$
220 IF A$="Y" OR A$="y" THEN GOSUB 1000
230 PRINT :: PRINT "<<ENTER>> to continue" :: INPUT A$ :: CALL CLEAR
240 PRINT "Traverse in sorted order (y or n)" :: INPUT A$ ::
250 IF A$="y" OR A$="Y" THEN GOSUB 2000
260 RETURN
1000 !*****PRINT TABLES*****
1010 FOR I=1 TO N
1020 FOR J=1 TO M
1030 PRINT X(I,J);L(I,J,1);L(I,J,2)
1040 NEXT J :: PRINT
1050 NEXT I
1060 RETURN
2000 !***** Traverse Jth list in sorted order
2010 PRINT "What key (0=return)" :: INPUT J
2020 IF J<1 OR J>M THEN RETURN
2030 GOSUB 3000
2040 GOTO 2010
3000 !***** Tree traversal *****
3010 P=1 :: T=0
3020 T=T+1 :: S(T)=P
3030 IF P<>0 THEN P=L(P,J,1):: GOTO 3020
3040 T=T-1 :: P=S(T)
3050 IF X(P,J)<>0 THEN PRINT X(P,J);: T=T-1 :: P=L(P,J,2):: GOTO 3020 ELSE PRIN
T :: RETURN
4000 !***** Build left and right links *****
4010 IF X1>X(P,J)THEN 4050
4020 IF L(P,J,1)<>0 THEN P=L(P,J,1):: GOTO 4010
4030 L(P,J,1)=I
4040 RETURN
4050 IF L(P,J,2)<>0 THEN P=L(P,J,2):: GOTO 4010
4060 L(P,J,2)=I
4070 RETURN
9999 END
```

```

How many keys per record (to 5) 4
How many records (to 20) 4
Type tables (y or n)Y
84 2 3      24 6 2      49 3 2      39 4 2
25 9 4      61 3 4      66 6 4      53 0 3
92 0 5      36 7 0      49 7 0      93 5 0
37 0 6      91 5 12     96 5 0      25 6 12
93 0 0      90 8 0      71 12 9     79 0 8
63 7 10     24 0 0      52 0 0      14 0 7
41 8 11     28 0 0      37 8 0      20 9 10
40 0 0      75 9 11     15 0 0      83 11 0
13 0 0      64 10 0     96 10 0     17 0 0
64 0 0      63 0 0      94 0 11     22 0 0
62 12 0     85 0 0      96 0 0      82 0 0
57 0 0      95 0 0      70 0 0      38 0 0

```

<<ENTER>> to continue

Traverse in sorted order (y or n)Y

What key (0=return) 1

13 25 37 40 41 57 62 63 64 84 92 93

What key (0=return) 2

24 24 28 36 61 63 64 75 85 90 91 95

What key (0=return) 3

15 37 49 49 52 66 70 71 94 96 96 96

What key (0=return) 4

14 17 20 22 25 38 39 53 79 82 83 93

What key (0=return) 0

<<ENTER>> for more

How many keys per record (to 5) 2

How many records (to 20) 2

Type tables (y or n)Y

```

38 7 2      90 3 2
61 6 3      98 0 0
71 8 4      56 5 4
99 5 0      83 7 9
96 10 0     24 6 8
50 11 9     19 12 11
17 0 0      65 14 0
66 0 0      26 0 0
57 0 0      88 10 0
96 12 0     85 15 0
48 0 0      21 0 0
83 14 13    18 13 0
88 0 15     17 0 0
82 0 0      65 0 0
89 0 0      85 0 0

```

<<ENTER>> to continue

Traverse in sorted order (y or n)Y

What key (0=return) 1

17 38 48 50 57 61 66 71 82 83 88 89 96 96 99

What key (0=return) 2

17 18 19 21 24 26 56 65 65 83 85 85 88 90 98

What key (0=return) 0

<<ENTER>> for more

Trees Native to North America

hald cypress	pond cypress	American larch
eastern white pine	longleaf pine	slash pine
pitch pine	shortleaf pine	loblolly pine
red pine	Jack pine	eastern hemlock
red spruce	white spruce	black spruce
balsam fir	northern white-cedar	Atlantic white-cedar
eastern red-cedar	southern red-cedar	cabbage palmetto
northern catalpa	flowering dogwood	white ash
sugar maple	black maple	silver maple
red maple	box-elder	red ash
black ash	yellow buckeye	Ohio buckeye
black walnut	butternut	pecan
bitternut hickory	shagbark hickory	pinus hickory
honey-locust	black locust	sassafras
red mulberry	osage-orange	sweet-gum
American sycamore	tulip tree	southern magnolia
cucumber-tree	common persimmon	water tupelo
black gum	American holly	American basswood
American elm	slippery elm	hackberry
eastern cottonwood	balsam poplar	bistooth aspen
yellow birch	sweet birch	paper birch
gray birch	black willow	black cherry
beech	chestnut	northern red oak
pin oak	black oak	southern red oak
white oak	water oak	willow oak
chinkapin oak	swamp chestnut oak	swamp white oak
bur oak	post oak	western larch
pinon pine	limber pine	lodpole pine
ponderosa pine	Jeffrey pine	western white pine
Pacific yew	sugar pine	western hemlock
blue spruce	sitka pine	white fir
grand fir	redwood	California red fir
Engelmann spruce	Douglas fir	noble fir
giant sequoia	incense-cedar	western red-cedar
Alaska-cedar	Arizona cypress	Rocky Mountain Juniper
western juniper	Pacific dogwood	bigleaf maple
Oregon ash	quaking aspen	plains cottonwood
black cottonwood	red alder	cascara buckhorn
Pacific madrone	golden chinkapin	California black oak
Oregon white oak	California white oak	gambel oak

BSST on Disk

The next program, STRTREE, is a significant departure from the multi-key, in-memory BSST. It stores its keys and links on a direct access file. This means that the file can be accessed on its key and the contents can be displayed in sorted order based on that key. The file was purposely built to contain the same small words string data that was used in previous programs. In this case, however, the records are on a direct access disk file, and a part of each record is made up of the links that provide for sorted order traversal.

The program is written in two major segments:

1. Build file (input data from sequential file WORDS2)
2. Display records (traverse direct access file in sorted order)

```

10 REM filename: "strtree"
20 REM purpose: BSST of string data on a D.A. file
30 REM author: jpg & jdr 8/82 (car)
40 REM -----
50 DIM S(100)
60 OPEN #1:"DSK1.WORDS2",SEQUENTIAL,INTERNAL,INPUT
70 OPEN #2:"DSK1.WORDS3",RELATIVE,INTERNAL,UPDATE
80 CALL CLEAR
90 INPUT "How many words do you wish to transfer ":K
100 ! Read seq.file 1 record at a time, transfer to D.A. file
110 I=R :: INPUT #1:N1$,F1
120 PRINT "Inputted sequ.record=";N1$,F1
130 IF I=0 THEN L1=0 :: L2=0 :: GOTO 210
140 P=I :: GOSUB 2000 !Get links from ith record
145 T$=" %STR$(E):: L1$=SEG$(T$,LEN(T$)-4,5):: L2$=L1$
147 N$=SEG$(X$,6,10)
150 IF N1$>N$ THEN 200
160 IF N1$=N$ THEN PRINT N$;"on file" :: GOTO 110
170 ! If left link is null, redefine it as E
180 IF L1=0 THEN X$=SEG$(X$,1,15)&L1$&SEG$(X$,21,5):: XX$=X$ :: PRINT #2,REC I:X
X$ :: GOTO 210 ELSE I=L1 :: GOTO 140
190 ! If right link is null, redefine it as E
200 IF L2=0 THEN X$=SEG$(X$,1,20)&L2$ :: XX$=X$ :: PRINT #2,REC I:XX$ ELSE I=L2
:: GOTO 140
210 ! Define a buffer
212 T$=" %STR$(F1):: F$=SEG$(T$,LEN(T$)-5,5):: N1$=SEG$(N1$&" ",1,1
0)
214 L1$=" 0" :: L2$=L1$ :: X$=F$&N1$&L1$&L2$
220 PRINT "Outputted D.A. record=";E;X$
230 IF R=0 THEN R=1 :: E=1 :: N=1
250 XX$=X$ :: PRINT #2,REC E:XX$ :: N=N+1 :: E=E+1
260 IF E=K THEN 110
270 PRINT :: INPUT "<<ENTER>> TO CONTINUE":A$ :: CALL CLEAR
280 GOSUB 1000
330 PRINT "Sorted order traversal"
340 P=1 :: T=0
350 GOSUB 2000
360 Q=0
370 T=T+1 :: S(T)=P 'Stack P, increment top pointer P
380 !traverse left branch
390 IF P>0 THEN GOSUB 2000 :: P=L1 :: GOTO 370
400 T=T-1 'Decrement top pointer, process record
410 IF T=0 THEN PRINT "all done" :: GOTO 9999
420 P=S(T):: GOSUB 2000 'Pop the stack
430 Q=Q+1
440 IF Q-INT(Q/4)*4=0 THEN PRINT
450 PRINT SEG$(X$,6,5);SEG$(X$,1,5),
460 T=T-1 :: P=L2 :: GOTO 370 !Traverse right branch
1000 ! Subroutine to print out records
1010 PRINT "Records as stored on D.A. file"
1020 FOR I=1 TO K
1030 INPUT #2,REC I:XX$ :: X$=XX$ :: PRINT I;X$
1040 NEXT I
1050 RETURN
2000 ! Subroutine to return links
2010 INPUT #2,REC P:XX$ :: X$=XX$
2020 L1$=SEG$(X$,16,5):: L1=VAL(L1$)
2030 L2$=SEG$(X$,21,5):: L2=VAL(L2$)
2040 RETURN
9999 CLOSE #1 :: CLOSE #2 :: END

```



```

Outputted D.A. record= 2 185as How many words do you wish to transfer
10
Inputted sequ.record=the 15568
Outputted D.A. record= 0 1556the 0 0
Inputted sequ.record=as 1853
Outputted D.A. record= 2 185as 0 0
Inputted sequ.record=and 7638
Outputted D.A. record= 3 763and 0 0
Inputted sequ.record=have 1344
Outputted D.A. record= 4 134have 0 0
Inputted sequ.record=i 2292
Outputted D.A. record= 5 229i 0 0
Inputted sequ.record=in 4312
Outputted D.A. record= 6 431in 0 0
Inputted sequ.record=that 3017
Outputted D.A. record= 7 301that 0 0
Inputted sequ.record=is 2509
Outputted D.A. record= 8 250is 0 0
Inputted sequ.record=for 1869
Outputted D.A. record= 9 186for 0 0
Inputted sequ.record=it 2255
Outputted D.A. record= 10 225it 0 0

```

<<ENTER>> TO CONTINUE

Records as stored on D.A. file

1	1556the	2	0
2	185as	3	4
3	763and	0	0
4	134have	9	5
5	229i	0	6
6	431in	0	7
7	301that	8	0
8	250is	0	10
9	186for	0	0
10	225it	0	0

Sorted order traversal

and	763	as	185	for	186	have	134	i	229	in	431
is	250	it	225	that	301	the	1556				

all done

Tree and Circularly Linked List

The last program in this chapter, BOSTON, is an example of a program that uses both the linked list and the BSST data management techniques. It includes the building of a BSST and doubly linked lists within the tree. It also provides other links for access to additional information.

This application uses stations on Boston's subway system, also known as the "T", as data elements. There are four lines, Red, Green, Blue, and Orange. All stations are in the tree only once. Each line is represented as a linked list in which the head points to the first element in the list. Figure 7.4 is a sketch of the subway's system of stops and crossings.

Each station is included in the BSST only once. When duplicates are encountered, the program generates a crossing link, which identifies the additional line on which this station appears. Some stations allow for the departure to another line by yet a third line. This condition creates a "get to" link which at present is only flagged when appropriate.

The program's DATA contains the stations in the order that they appear on the line. The tree is built in standard fashion using a binary search comparison to determine whether the data already exists. This technique also allows for the speedy access to any station. Additions to

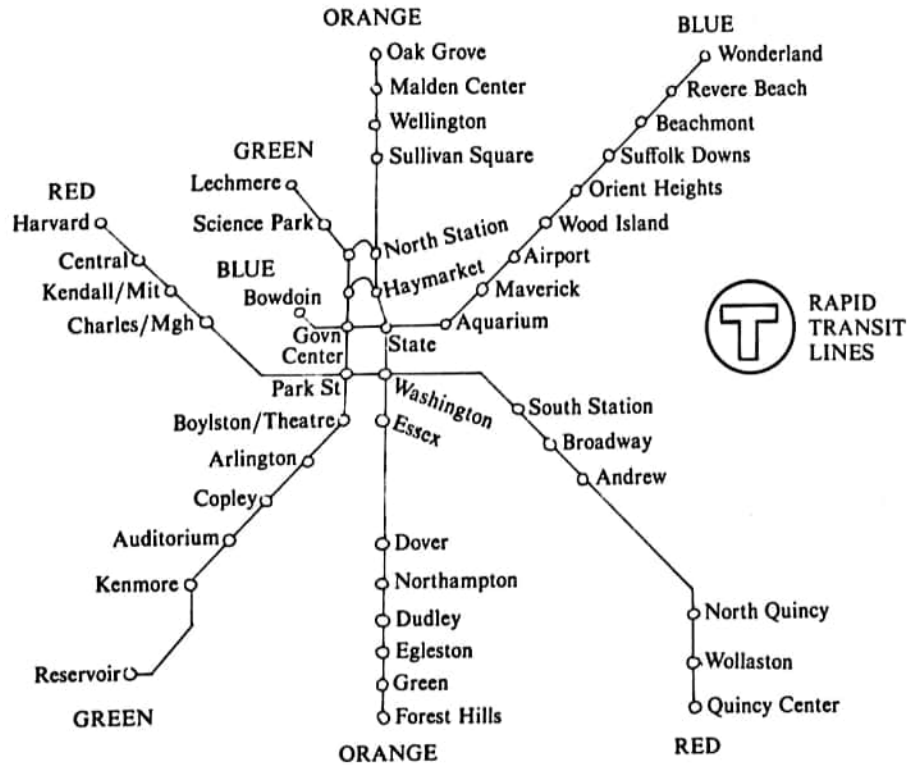


Figure 7.4 The Boston "T"

the line are of course possible, and you can list stations in alphabetical order with the usual traversal procedure.

The program builds the BSST using the T's stations as keys for the left and right pointers. This structure also has four associated linked lists, one for each line. These lists and corresponding pointers are also built as a new station is inserted into the tree.

As we have pointed out a number of times, the programs we include in this book are intended to be skeletal in nature. This one is no exception, and it could be improved with additional features. A major improvement to this application would be a "travel path query processor". You could access the source and destination stations through a binary search to determine the line to which each belongs.

If the source and destination stations are on the same line, you could generate a movement through either the right or left pointers of the appropriate list. You would get the correct direction by comparing the right pointers of the source and destination.

If source and destination are on different lines, the "travel path" algorithm must incorporate additional information. As in the above system, first locate the source and destination on the appropriate line. Then perform a table lookup, noting those stations on the source line which either appear on the destination line (examine the cross link also) or note the "get to" links which connect to the destination line.

When you find a crossing point or chain link, get the correct direction from the right pointers of the source station and the cross or chain station. Then you know which way to travel on the source line.

Then compare the right pointer of the crossed or chained-to station and the destination right pointer. This will tell you the direction on the destination line.

The structure implemented above could be described as a primitive inverted file. The subject of inverted files is discussed in greater detail in Chapter 8. The structure can provide "chain" or "get to" information noting how to get from one line to another if there are no common stops. You must note the entrance point onto the destination line for speedy path generation. You can use additional links to provide more information, such as sub-lines, and time schedules.

Although at first glance this application may appear to exhibit only problem solving techniques, there may be a good, practical use for this program. Many travelers in a variety of transit stations could use the ability to determine travel paths. Transit stations are notorious in not providing easily accessible travel information.

We are indebted to Celia Robertson for her analysis, programming, and documentation of this problem. It shows the use of the tree structure very well, and in addition incorporates a practical use of a doubly linked list. In the next chapter, we will include another major application, again incorporating a variety of data management techniques.

```

10 REM filename: "boston"
20 REM purpose: BSST and linked lists to deal with subway
30 REM author: jpg & jdr B/B2 (car)
40 REM -----
50 !***This program build a BSST and linked lists*****
60 !***The program allows information gathering*****
70 !*****about the Boston "T"*****
80 !*****PART I*****
90 DIM N$(51),L(51),LL(51),RL(51),CL(51),GL(51)
100 DIM H(4),LP(51),RP(51),C$(4),ST(51)
110 !*** N$ contains the name of the station
120 !*** L contains the line number
130 !*** LL is the BSST left link, RL is the right link
140 !*** CL is a crossing link built while tree built
150 !*** H points to the head of each linked list
160 !*** LP is the left pointer in then linked lists
170 !*** RP is the right pointer in the linked lists
180 !*** C$ holds line colors, ST is a stack for traversal
190 FOR I=1 TO 47
200 N$(I)=" " : L(I)=0 : LL(I)=0 : RL(I)=0 : GL(I)=0
210 LP(I)=0 : RP(I)=0 : ST(I)=0
220 NEXT I
230 FOR I=1 TO 4
240 H(I)=0 : C$(I)=" "
250 NEXT I
260 CALL CLEAR
270 !***K will always point to the root of the BSST
280 K=0
290 FOR I=1 TO 51
300 READ N$(I),L(I),GL(I)
310 PRINT SEG$(N$(I))&" " ",1,15);" ";
320 IF I-INT(I/4)*4=0 THEN PRINT
330 !*** M contains line number of station
340 M=L(I)
350 IF K=0 THEN GOTO 470
360 J=K
370 IF N$(I)<N$(J) THEN 540
380 IF N$(I)>N$(J) THEN 550

```

```

390 *** Tree comparisons for possible insertion completed
400 *** If a duplicate is found, the line is added to
410 ***           the original in the tree
420 CL(J)=M
430 CL(I)=L(J)
440 GOTO 580
450 NEXT I
460 GOTO 660
470 K=I
480 LL(K)=0
490 RL(K)=0
500 LP(K)=0
510 RP(K)=0
520 *** The above handles the first entry in the tree
530 GOTO 580
540 IF LL(J)=0 THEN LL(I)=0 :: RL(I)=0 :: LL(J)=I :: GOTO 580 ELSE J=LL(J):: GOT
O 370
550 IF RL(J)=0 THEN LL(I)=0 :: RL(I)=0 :: RL(J)=I :: GOTO 580 ELSE J=RL(J):: GOT
O 370
560 *** The above inserts in a tree if possible
570 ***           and BSST links set
580 IF H(M)=0 THEN H(M)=I :: LP(I)=0 :: RP(I)=0 :: GOTO 450
590 P=H(M)
600 IF RP(P)=0 THEN RP(P)=I :: LP(I)=P :: RP(I)=0 :: GOTO 450 ELSE P=RP(P):: GOT
O 600
610 *** The above sets linked list pointers
620 ***           when insertion1 made
630 ***
640 ***           PART II
650 ***
660 C$(1)="RED" :: C$(2)="GREEN" :: C$(3)="BLUE"
670 C$(4)="ORANGE"
680 PRINT :: PRINT :: PRINT "PRESS <<ENTER>> TO CONTINUE" :: INPUT Z$ :: CALL CL
EAR
690 PRINT "THE MENU OF QUERY OPTIONS FOLLOWS"
700 PRINT "ENTER THE APPROPRIATE SYMBOL AT THE INPUT PROMPT."
710 PRINT :: PRINT
720 PRINT "PRESS @ TO FIND WHERE A STATION IS LOCATED"
730 PRINT "PRESS # TO SEE ALL STATIONS ON A LINE"
740 PRINT "PRESS $ TO SEE ALL STATIONS IN ALPHABETICAL ORDER"
750 PRINT "PRESS % TO SEE ALL STATIONS WHICH ARE CROSSINGS"
760 PRINT "PRESS & TO SEE CROSSINGS ON A LINE"
770 PRINT "PRESS . TO EXIT"
780 CALL KEY(O,V,S):: IF S=0 THEN 780 :: Q#=CHR$(V):: CALL CLEAR
790 IF Q#="@" THEN 860
800 IF Q#="#" THEN 1020
810 IF Q#="$" THEN 1170
820 IF Q#="%" THEN 1170
830 IF Q#="&" THEN 1020
840 IF Q#="." THEN 1400
850 PRINT "INVALID CHARACTER" :: GOTO 680
860 PRINT "Enter then station you wish to locate."
870 PRINT "Please enter valid station on attached map."
880 INPUT S$ ::
890 J=K
900 IF S#=N$(J) THEN 980
910 IF S#<N$(J) THEN J=LL(J)
920 IF S#>N$(J) THEN J=RL(J)
930 IF J=0 THEN 950 ELSE 900
940 *** Above handles binary search of tree
950 PRINT "STATION ";S$;" IS NOT ON THE LINE."
960 PRINT "CHECK THE ATTACHED MAP FOR A VALID STATION."
970 GOTO 680
980 A=L(J)
990 PRINT S$;" IS ON THE ";C$(A);" LINE"
1000 IF CL(J)<>0 THEN A=CL(J):: PRINT S$;" IS ALSO ON ";C$(A);" LINE"
1010 GOTO 680

```

```

1020 PRINT "Enter which line for which you wish stations"
1030 PRINT "Enter R for red, G for green, B for blue, O for orange."
1040 CALL KEY(O,V,S):: IF S=0 THEN 1040 :: S%=CHR$(V)
1050 IF S%="R" THEN P=H(1):: GOTO 1100
1060 IF S%="G" THEN P=H(2):: GOTO 1100
1070 IF S%="B" THEN P=H(3):: GOTO 1100
1080 IF S%="O" THEN P=H(4):: GOTO 1100
1090 PRINT "INVALID CHARACTER" :: GOTO 1020
1100 IF P=0 THEN PRINT "END OF LINE" :: GOTO 680
1110 IF Q%="#" THEN PRINT N$(P)
1120 IF Q%="&" THEN 1150
1130 P=RP(P)
1140 GOTO 1100
1150 A=CL(P):: IF CL(P)<>0 THEN PRINT "STATION-->";N$(P);" ALSO ON-->";C$(A);" L
LINE"
1160 GOTO 1130
1170 IF Q%="$" THEN PRINT "ALL STATIONS WILL BE LISTED IN ORDER"
1180 IF Q%="%" THEN PRINT "ALL CROSSINGS WILL BE PRINTED"
1190 S=0
1200 P=K
1210 IF P=0 THEN 1260
1220 S=S+1
1230 ST(S)=P
1240 P=LL(P)
1250 GOTO 1210
1260 IF S=0 THEN 680
1270 P=ST(S)
1280 S=S-1
1290 A=L(P):: B=CL(P)
1300 IF Q%="%" THEN 1350
1310 IF Q%="$" THEN PRINT "STATION-->";N$(P);" LINE-->";C$(A);
1320 IF Q%="$" AND B<>0 THEN PRINT " AND ";C$(B)ELSE PRINT
1330 P=RL(P)
1340 GOTO 1210
1350 A=L(P):: B=CL(P)
1360 IF B<>0 THEN PRINT "STATION-->";N$(P);" LINE-->";C$(B)
1370 GOTO 1330
1380 !*** Above is the inorder traversal of the BSST
1390 !*** What is printed depends on what symbol is input
1400 PRINT "Process complete"
1410 DATA HARVARD,1,0,CENTRAL,1,0
1420 DATA KENDALL/MIT,1,0,CHARLES/MGH,1,0
1430 DATA PARK STREET,1,-9,WASHINGTON,1,-9
1440 DATA SOUTH STATION,1,0,BROADWAY,1,0
1450 DATA ANDREW,1,0,NORTH QUINCY,1,0
1460 DATA WOLLASTON,1,0,QUINCY CENTER,1,0
1470 DATA LECHMERE,2,0,SCIENCE PARK,2,0
1480 DATA NORTH STATION,2,0,HAYMARKET,2,0
1490 DATA GOVERNMENT CENTER,2,-9,PARK STREET,2,-9
1500 DATA BOYLSTON/THEATER,2,0,ARLINGTON,2,0
1510 DATA COPLY,2,0,AUDITORIUM,2,00
1520 DATA KENMORE,2,0,RESERVOIR,2,0
1530 DATA BOUDION,3,0,GOVERNMENT CENTER,3,-9
1540 DATA STATE,3,-9,AQUARIUM,3,0
1550 DATA MAVERICK,3,0,AIRPORT,3,0
1560 DATA WOOD ISLAND,3,0,ORIENT HEIGHTS,3,0
1570 DATA SUFFOLK DOWNS,3,0,BEACHMONT,3,0
1580 DATA REVERE BEACH,3,0,WONDERLAND,3,0
1590 DATA OAK GROVE,4,0,MALDEN CENTER,4,0
1600 DATA WELLINGTON,4,0,SULLIVAN SQUARE,4,0
1610 DATA NORTH STATION,4,0,HAYMARKET,4,0
1620 DATA STATE,4,-9,WASHINGTON,4,-9
1630 DATA ESSEX,4,0,DOVER,4,0
1640 DATA NORTHHAMPTON,4,0,DUDLEY,4,0
1650 DATA EGLESTON,4,0,GREEN,4,0
1660 DATA FOREST HILLS,4,0
1670 END

```

HARVARD	CENTRAL	KENDALL/MIT	CHARLES/MGH
PARK STREET	WASHINGTON	SOUTH STATION	BROADWAY
ANDREW	NORTH QUINCY	WOLLASTON	QUINCY CENTER
LECHMERE	SCIENCE PARK	NORTH STATION	HAYMARKET
GOVERNMENT CENT	PARK STREET	BOYLSTON/THEATE	ARLINGTON
COPLY	AUDITORIUM	KENMORE	RESERVOIR
BOUDION	GOVERNMENT CENT	STATE	AQUARIUM
MAVERICK	AIRPORT	WOOD ISLAND	ORIENT HEIGHTS
SUFFOLK DOWNS	BEACHMONT	REVERE BEACH	WONDERLAND
OAK GROVE	MALDEN CENTER	WELLINGTON	SULLIVAN SQUARE
NORTH STATION	HAYMARKET	STATE	WASHINGTON
ESSEX	DOVER	NORTHHAMPTON	DUDLEY
EGLESTON	GREEN	FOREST HILLS	

PRESS <<ENTER>> TO CONTINUE
 THE MENU OF QUERY OPTIONS FOLLOWS
 ENTER THE APPROPRIATE SYMBOL AT THE INPUT PROMPT.

PRESS @ TO FIND WHERE A STATION IS LOCATED
 PRESS # TO SEE ALL STATIONS ON A LINE
 PRESS \$ TO SEE ALL STATIONS IN ALPHABETICAL ORDER
 PRESS % TO SEE ALL STATIONS WHICH ARE CROSSINGS
 PRESS & TO SEE CROSSINGS ON A LINE
 PRESS . TO EXIT

%
 ALL CROSSINGS WILL BE PRINTED
 STATION-->GOVERNMENT CENTER LINE-->BLUE
 STATION-->HAYMARKET LINE-->ORANGE
 STATION-->NORTH STATION LINE-->ORANGE
 STATION-->PARK STREET LINE-->GREEN
 STATION-->STATE LINE-->ORANGE
 STATION-->WASHINGTON LINE-->ORANGE

PRESS <<ENTER>> TO CONTINUE
 THE MENU OF QUERY OPTIONS FOLLOWS
 ENTER THE APPROPRIATE SYMBOL AT THE INPUT PROMPT.

PRESS @ TO FIND WHERE A STATION IS LOCATED
 PRESS # TO SEE ALL STATIONS ON A LINE
 PRESS \$ TO SEE ALL STATIONS IN ALPHABETICAL ORDER
 PRESS % TO SEE ALL STATIONS WHICH ARE CROSSINGS
 PRESS & TO SEE CROSSINGS ON A LINE
 PRESS . TO EXIT

@
 Enter then station you wish to locate.
 Please enter valid station on attached map.
 KENMORE
 KENMORE IS ON THE GREEN LINE

PRESS <<ENTER>> TO CONTINUE
 THE MENU OF QUERY OPTIONS FOLLOWS
 ENTER THE APPROPRIATE SYMBOL AT THE INPUT PROMPT.

PRESS @ TO FIND WHERE A STATION IS LOCATED
 PRESS # TO SEE ALL STATIONS ON A LINE
 PRESS \$ TO SEE ALL STATIONS IN ALPHABETICAL ORDER
 PRESS % TO SEE ALL STATIONS WHICH ARE CROSSINGS
 PRESS & TO SEE CROSSINGS ON A LINE
 PRESS . TO EXIT

#

Enter which line for which you wish stations
Enter R for red, G for green, B for blue, O for orange.

R
HARVARD
CENTRAL
KENDALL/MIT
CHARLES/MGH
PARK STREET
WASHINGTON
SOUTH STATION
BROADWAY
ANDREW
NORTH QUINCY
WOLLASTON
QUINCY CENTER
END OF LINE

PRESS <<ENTER>> TO CONTINUE
THE MENU OF QUERY OPTIONS FOLLOWS
ENTER THE APPROPRIATE SYMBOL AT THE INPUT PROMPT.

PRESS @ TO FIND WHERE A STATION IS LOCATED
PRESS # TO SEE ALL STATIONS ON A LINE
PRESS \$ TO SEE ALL STATIONS IN ALPHABETICAL ORDER
PRESS % TO SEE ALL STATIONS WHICH ARE CROSSINGS
PRESS & TO SEE CROSSINGS ON A LINE
PRESS . TO EXIT

&

Enter which line for which you wish stations
Enter R for red, G for green, B for blue, O for orange.

O

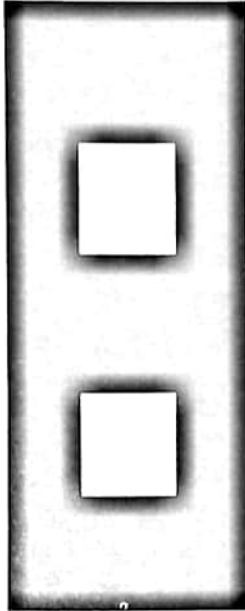
STATION-->NORTH STATION ALSO ON-->GREEN LINE
STATION-->HAYMARKET ALSO ON-->GREEN LINE
STATION-->STATE ALSO ON-->BLUE LINE
STATION-->WASHINGTON ALSO ON-->RED LINE
END OF LINE

PRESS <<ENTER>> TO CONTINUE
THE MENU OF QUERY OPTIONS FOLLOWS
ENTER THE APPROPRIATE SYMBOL AT THE INPUT PROMPT.

PRESS @ TO FIND WHERE A STATION IS LOCATED
PRESS # TO SEE ALL STATIONS ON A LINE
PRESS \$ TO SEE ALL STATIONS IN ALPHABETICAL ORDER
PRESS % TO SEE ALL STATIONS WHICH ARE CROSSINGS
PRESS & TO SEE CROSSINGS ON A LINE
PRESS . TO EXIT

.

Process complete



Inverted Files

There are many occasions in file processing when the retrieval of information from more than one record is required. The information desired is based on a relationship among the records and the request for information might take the form of a *query* like these:

“List all 1950’s movies”.

“What ’60s movies on file are Westerns?”

“List all Good-rated comedies from the 1970’s that are on file”

Secondary Keys

Rather than using a *unique key*, the movie title, the user wants filed information based on what are called *secondary keys* that are not necessarily unique. We will build such a system in this chapter. Of course you must realize that the technique used here for home entertainment is used often in the working world in industry, education, small businesses, banking, and many other areas where retrieval systems have become computerized.

The entire chapter is devoted to a single inverted file system comprised of five programs, a menu driver and four quite large and sophisticated applications. This system was written by Steve Grillo at our request for inclusion in this book, and we are indebted to him for this substantial effort.

Record Structure

The records on this multiple-key retrieval system are made up of the following fields:

Name	Description
LL\$	left link for B tree
RL\$	right link for B tree
NA\$	title of movie (unique key)
AC\$	starring actors
YR\$	decade of release
CR\$	critic ratings
TY\$	type of film link
Y1\$	link for decade of release
C1\$	link for critic ratings
T1\$	link for type ratings

The primary and unique key is the movie title. The three other keys are used by the system to focus on a multi-key query. They allow the user to specify, say, the list of movies released in the 1960's that were good or better comedies.

The year of release is by decade, such that for a movie made in either 1952, or 1959, the year of release is 1950. The critic rating is coded as follows: E = Excellent, G = Good, F = Fair to poor.

The type of movie is coded as three characters:

- ADV = Adventure
- BIB = Biblical epic
- BIO = Biography
- CHI = Children
- COM = Comedy
- CRI = Crime-detective
- DIS = Disaster
- DOC = Documentary
- DRA = Drama
- HOR = Horror
- MUS = Musical
- SCI = Science fiction
- TRA = Travelogue
- WAR = War
- WES = Western

Record Contents

The three secondary keys each have corresponding links Y, C, and T. These links show the record number of the next logical record that contains the same code. Consider this chart of sample records generated from a random sampling of movies.

Record #	Title NA\$	Cast ST\$	B-tree links		Secondary links			Secondary keys			
			LL	RL	Y	C	T	YR	CR	TY	
1	Paris Playboys	Bowery Boys	2	5	3	4	9	50	F	COM	
2	Little Miss Marker	S. Temple	4	3		3		30	G	CHI	
3	Magic Box	A. Menjou R. Donal M. Schell				7	5	6	50	G	BIO
4	Hundred Cries of Terror	A. Welter	7					60	F	HOR	
5	Spikes Gang	J. Corder L. Marvin R. Howard		6	6	8		70	G	WES	
6	Wilma	S. Finney C. Tyson	9		9	7		70	E	BIO	
7	Band Wagon	F. Astaire C. Charisse		8	10*	10*		50	E	MUS	
8	Bombardier	P. O'Brien R. Scott				9	10*	40	G	WAR	
9	Star Spangled Girl	S. Duncan T. Robert		10*				70	G	COM	
10	War and Peace	A. Heburn M. Ferrer H. Fonda						50	E	WAR	

In the chart above, the last movie to be entered as record 10 (War and Peace) changes the right link of record 9. Its secondary keys (a 1956 highly rated movie about the Napoleonic wars) are 53, E, and WAR. The last time a 50's movie was entered as a Year secondary key was in record 7, so its Y link is changed from null to 10. The last time an E was entered as a Critics secondary key was by coincidence also in record 7, so its C link is changed from null to 10. The last time a WAR movie was entered was in record 8, so its Type link T is changed to 10. Note that if a new type of movie had been entered, it would not have been necessary to alter a previous type link. We have placed an "*" next to these changes to draw your attention to them.

File Access Using
Pointer Table

When the file is accessed, the first Year link for each year from 1 to 8 (1910s to 1980s), as well as for each Critic and Type, may be recorded into a pointer table. In the preceding example, the pointer table would look like this:

Occurrence Table for Secondary Links

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Year 19--	10	20	30	40	50	60	70	80							
			2	8	1	4	5								
Critic	P-F	G	E												
	1	2	6												
Type	ADV	BIB	BIO	CHI	COM	CRI	DIS	DRA	DOC	HOR	MUS	SCI	TRA	WAR	WES
			3	2	1					4	7			8	5

The accessing program takes the user's keys, say 1960s war movies, and checks to see if the 1960s decade has a pointer value other than null. It does, and record 4 is the first occurrence of a 1960s movie. Then the program would check to see if the type WAR has a pointer which is not null. Yes again, and record 8 is the first occurrence of a war movie. Finally, the program would perform its file search in this manner:

1. Get record 8, based on the fact that Type pointers are more numerous and therefore more selective (fewer entries per category) than the Year pointers.
2. Check the record Year key. If it is 1960s, print it.
3. Check record's Type link.
 - a. If it is null, the search is complete.
 - b. If it is not null, fetch the next WAR record because that Type link points to it, and go back to step 2.

This pointer table is stored as the first record of the file.

Main Program
Driver

The first program in this retrieval system is the one that links to all others, and the one to which all others return. It displays a menu of the activities that can be performed on the file and runs the chosen program. The program's listing shows all of the activities that this system can perform.

```

10 REM filename: "movie"
20 REM purpose: Main driver for movie program
30 REM author: spg, jpg & jdr
40 REM -----
50 CALL CLEAR
60 PRINT :: PRINT
70 PRINT " Movie Program"
80 PRINT TAB(10); "Menu"
90 PRINT :: PRINT
100 PRINT " <1>-insert movies into disk      file"
110 PRINT " <2>-access and look at          certain movies"
120 PRINT " <3>-list all of the movies      in any of several order"
130 PRINT " <4>-list some or all of the    data records"
140 PRINT " <5>-stop the execution of      this program"
150 INPUT "Which activity?":A
160 PRINT
170 IF A<1 OR A>5 THEN 50
180 ON A GOTO 190,200,210,220,230
190 RUN "DSK1.INSERT"
200 RUN "DSK1.ACCESS"
210 RUN "DSK1.LISTLG"
220 RUN "DSK1.LISTER"
230 END

```

Deletion and Balancing

Deletion and balancing have been omitted from this system because in this kind of database the records will never be deleted, only added, and their order of addition will probably be random enough that the balancing algorithm is also (fortunately) unnecessary.

Record Insertion

The first program to which the driver links is the one responsible for building and adding to the database. It builds a BSST structure and it maintains three separate secondary key linked lists.

```

10 REM filename: "insert"
20 REM purpose: Insert movie records into BSST inverted file
30 REM author: hmz, spg, jpg & jdr 10/83
40 REM -----
50 OPEN #1:"DSK1.MOVDAT",RELATIVE,INTERNAL,UPDATE,FIXED 254
60 DIM M$(16),N$(4),T2$(15),Y2$(15),Y2(8)
70 DIM YL(50),CL(50),TL(50)
80 FOR I=1 TO 15 :: READ M$(I):: NEXT I
90 DATA adventure,biblical epic,biography,childern
100 DATA documentary,horror,musical,science fiction
110 DATA comedy,crime-dective,disaster,drama
120 DATA travel,war,western
130 FOR I=0 TO 3 :: READ R$(I):: NEXT I
140 DATA unknown,fair to poor,good,excellent
150 ! Skip next section if file exists
160 INPUT "DOES FILE EXIST(Y OR N)":A$ :: IF A$="Y" OR A$="y" THEN 210
170 RC=1 :: FOR I=0 TO 3 :: C2$(I)="0" :: NEXT I
180 FOR I=1 TO 15 :: T2$(I)="0" :: NEXT I
190 FOR I=1 TO 8 :: Y2$(I)="0" :: NEXT I
200 GOSUB 1390
210 CALL CLEAR
220 PRINT " Gathering data from disk.."
230 GOSUB 1340
240 ! Set up the arrays of secondary links
250 ! using the data on the first record
260 FOR J=0 TO 3 :: C2(J)=VAL(C2$(J)):: NEXT J
270 FOR J=1 TO 15 :: T2(J)=VAL(T2$(J)):: NEXT J
280 FOR J=1 TO 8 :: Y2(J)=VAL(Y2$(J)):: NEXT J
290 ! Skip next section if file has no movie data

```

```

300 IF RC=1 THEN 400 ! Check the record count
310 DISPLAY AT(6,1):"( records left to be read)";
320 ! set up the YL, CL, and TL arrays using
330 ! the data on the disk
340 FOR IN=2 TO RC :: GOSUB 1440
350 DISPLAY AT(6,2):RC-IN;
360 YL(IN)=VAL(YR$)
370 CL(IN)=VAL(CR$)
380 TL(IN)=VAL(TY$)
390 NEXT IN
400 ! this section gets all of the needed
410 ! information concernig the movie
420 ! from the user
430 CALL CLEAR :: PRINT
440 RN=RC+1
450 IN=2
460 PRINT "To cancel this movie data at any time,"
470 PRINT "just enter '*'." :: PRINT
480 PRINT "Movie title ('%' to stop)" :: LINPUT X$
490 IF X$="*" THEN 430
500 IF X$="%" THEN GOTO 1500
510 PRINT
520 LINPUT "Actors? (separated by commas<:":S$
530 IF S$="*" THEN 430
540 PRINT :: PRINT
550 PRINT " Enter the decade of release as follows:"
560 FOR J=1 TO 8
570 PRINT " "&STR$(J)&" ==> 19"&STR$(J)&"0's"
580 NEXT J
590 LINPUT "Decade of release; ":YT$
600 IF YT$<"1" OR YT$>"8" THEN 430
610 PRINT
620 PRINT "Enter critic's rating as follows:"
630 PRINT " 0-unknown rating"
640 PRINT " 1-fair to poor rating"
650 PRINT " 2-good rating"
660 PRINT " 3-excellent rating"
670 LINPUT "Please enter critic's rating:":CT$
680 IF CT$<"0" OR CT$>"3" THEN 430
690 PRINT :: PRINT "Type of movie:"
700 FOR J=1 TO 15 :: PRINT J;M$(J):: NEXT J
710 PRINT
720 INPUT "What type of movie is this (1-15):":TT$
730 IF VAL(TT$)<1 OR VAL(TT$)>15 THEN G50
740 PRINT :: PRINT "personal comments concerning this movie"
750 LINPUT "(less than 33 characters please):":P$
760 CALL CLEAR :: PRINT
770 PRINT "Record#";RN
780 PRINT "Left link - 0";
790 PRINT TAB(16);"right link - 0"
800 PRINT
810 PRINT "Movie title: " :: PRINT TAB(6);X$
820 PRINT "Actors:" :: PRINT TAB(6);S$
830 PRINT "Released in :";"19";YT$;"0's"
840 PRINT "Consumer union rating: ";R$(VAL(CT$))
850 PRINT "Type of movie: ";M$(VAL(TT$))
860 PRINT "Personal comment: ";P$
870 PRINT
880 PRINT "Press <<ENTER>> if this data is O.K.,"
890 PRINT "otherwise enter '*' to re-enter"

```

```

900 INPUT "this record.":A$
910 IF A$="*" THEN 430
920 IF RC<2 THEN 1040
930 ! this is the BSST section which sets the
940 ! right and left links of the other records
950 GOSUB 1440 :: LT=VAL(LL$):: RT=VAL(RL$):: NT$=NA$
960 IF X$>NT$ THEN 1000
970 IF X$=NT$ THEN PRINT " Already on file (<<ENTER>> to continue)" :: INPUT A$
:: GOTO 430
980 IF LT<>0 THEN IN=LT :: GOTO 950
990 LL$=STR$(RN):: GOTO 1020 ! change LL of record I
1000 IF RT<>0 THEN IN=RT :: GOTO 950
1010 RL$=STR$(RN)! change RL of record I
1020 GOSUB 1470 ! replace the new record with good links
1030 ! now fill the buffer with new data
1040 LL$="0" :: RL$="0" :: NA$=X$
1050 AC$=S$ :: Y1$=YT$ :: C1$=CT$ :: T1$=TT$
1060 PC$=F$
1070 YR$="0" :: CR$="0" :: TY$="0"
1080 ! write this record on disk (finally)
1090 IN=RN :: GOSUB 1470 :: GOSUB 1340 :: RC=RC+1 :: GOSUB 1390
1100 !Set critic's rating occurrence links if needed
1110 CR=VAL(CT$):: TY=VAL(TT$):: YR=VAL(YT$)
1120 IF C2(CR)=0 THEN C2(CR)=RN :: GOSUB 1340 :: C2$(CR)=STR$(RN):: GOSUB 1390 :
: GOTO 1190
1130 ! set "next record with this rating" link
1140 T=C2(CR)
1150 IF CL(T)<>0 THEN T=CL(T):: GOTO 1150
1160 CL(T)=RN
1170 IN=T :: GOSUB 1440 :: CR$=STR$(RN):: GOSUB 1470
1180 ! set movie type occurrence links if needed
1190 IF T2(TY)=0 THEN T2(TY)=RN :: GOSUB 1340 :: T2$(TY)=STR$(RN):: GOSUB 1390 :
: GOTO 1260
1200 ! set "next record with this type" link
1210 T=T2(TY)
1220 IF TL(T)<>0 THEN T=TL(T):: GOTO 1220
1230 TL(T)=RN
1240 IN=T :: GOSUB 1440 :: TY$=STR$(RN):: GOSUB 1470
1250 ! set decade of release occurrence links if needed
1260 IF Y2(YR)=0 THEN Y2(YR)=RN :: GOSUB 1340 :: Y2$(YR)=STR$(RN):: GOSUB 1390 :
: GOTO 1330
1270 ! set "next record with this decade" link
1280 T=Y2(YR)
1290 IF YL(T)<>0 THEN T=YL(T):: GOTO 1290
1300 YL(T)=RN
1310 IN=T :: GOSUB 1440 :: YR$=STR$(RN):: GOSUB 1470
1320 ! go back to get another movie
1330 GOTO 430
1340 !To read the pointer record
1350 INPUT #1,REC 1:RC,:: FOR I=0 TO 3 :: INPUT #1:C2$(I),:: NEXT I
1360 FOR I=1 TO 15 :: INPUT #1:T2$(I),:: NEXT I
1370 FOR I=1 TO 8 :: INPUT #1:Y2$(I),:: NEXT I ::
1380 RETURN
1390 !To write the pointer record
1400 PRINT #1,REC 1:RC,:: FOR I=0 TO 3 :: PRINT #1:C2$(I),:: NEXT I
1410 FOR I=1 TO 15 :: PRINT #1:T2$(I),:: NEXT I
1420 FOR I=1 TO 8 :: PRINT #1:Y2$(I),:: NEXT I ::
1430 RETURN
1440 !To read a data record
1450 INPUT #1,REC IN:LL$,RL$,NA$,AC$,YR$,CR$,TY$,Y1$,C1$,T1$,PC$
1460 RETURN
1470 !To write a data record
1480 PRINT #1,REC IN:LL$,RL$,NA$,AC$,YR$,CR$,TY$,Y1$,C1$,T1$,PC$
1490 RETURN
1500 CLOSE #1 :: RUN "DSK1.MOVIE"

```

Movie Program
Menu

<1>-insert movies into disk file
<2>-access and look at certain movies
<3>-list all of the movies in any of several order
<4>-list some or all of the data records
<5>-stop the execution of this program

Which activity? 1

DOES FILE EXIST(Y OR N)

YES

Gathering data from disk..

To cancel this movie data at any time,
just enter '*'.
Movie title ('%' to stop)

Risky Business

Actors? (separated by commas):T.Cruise

Enter the decade of release as follows:

1 ==> 1910's

2 ==> 1920's

3 ==> 1930's

4 ==> 1940's

5 ==> 1950's

6 ==> 1960's

7 ==> 1970's

8 ==> 1980's

Decade of release: 8

Random Access

The program to access the file for more than one record based on the user's queries is also quite complex. First it must analyze the user's query, then retrieve only those records that match it. The technique for query analysis in this program is kept somewhat simple so that it doesn't detract from the essential features of inverted file processing. The user enters the responses upon request from the program.

The following program, ACCESS, implements the query management and the database access.

```

10 REM filename: "access"
20 REM purpose: Insert movie records into BSST inverted file
30 REM author: hnz, spg, jpg & jdr 10/83
40 REM -----
50 OPEN #1:"DSK1.MOVDAT",RELATIVE,INTERNAL,UPDATE,FIXED 254
60 DIM M$(16),N$(4),T2$(15),Y2$(8),Y5(20),Y5(8)
70 FOR I=1 TO 15 :: READ M$(I):: NEXT I
80 DATA adventure,biblical epic,biography,children
90 DATA documentary,horror,musical,science fiction
100 DATA comedy,crime-detective,disaster,drama
110 DATA travel,war,western
120 FOR I=0 TO 3 :: READ N$(I):: NEXT I
130 DATA unknown,fair to poor,good,excellent
140 GOSUB 1020
150 ! set up the arrays of secondary links
160 ! using the data on the first record
170 FOR J=0 TO 3 :: C2(J)=VAL(C2$(J)):: NEXT J
180 FOR J=1 TO 15 :: T2(J)=VAL(T2$(J)):: NEXT J
190 FOR J=1 TO 8 :: Y2(J)=VAL(Y2$(J)):: NEXT J
200 CALL CLEAR :: PRINT :: PRINT
210 PRINT TAB(5);"M o v i e A c c e s s i n g"
220 PRINT TAB(9);"P r o g r a m"
230 PRINT :: PRINT
240 PRINT " This program access the "
250 PRINT "movie data file on disk and gives descriptions of movies"
260 PRINT "that you want to see."
270 PRINT " You tell me which"
280 PRINT "categories you want, and I "
290 PRINT "will find and display the"
300 PRINT "movies (if there are any)"
310 PRINT "which satisfy your restric- tions." :: PRINT :: PRINT :: INPUT "<<ENTER
ER>>":A$
320 Y=0 :: C=0 :: T=0
330 CALL CLEAR
340 INPUT "Do you want a specific decade (<<ENTER>>=No)":A$
350 IF SEG$(A$,1,1)="Y" OR SEG$(A$,1,1)="y" THEN 370
360 FOR TC=1 TO 8 :: Y5(TC)=1 :: NEXT TC :: GOTO 540
370 CALL CLEAR
380 PRINT "<1>-search for one specific decade"
390 PRINT "<2>-search for a range of decades"
400 PRINT " (e.g. 1920's - 1960's)"
410 PRINT :: INPUT "Which activity:":A
420 IF A<1 OR A>2 THEN 330
430 IF A=2 THEN 480
440 PRINT
450 INPUT "Enter one digit decade(e.g. 3=1930's)":Y5$
460 IF Y5$<"1" OR Y5$>"8" THEN 330
470 Y5(VAL(Y5$))=1 :: GOTO 540
480 PRINT :: PRINT "Enter one digit decades (e.g. 3=1930's)"
490 INPUT "Lower boundry(1 digit decade)":Y8$
500 INPUT "Upper boundry(1 digit decade)":Y9$
510 IF Y8$<"1" OR Y8$>"8" OR Y9$<"1" OR Y9$>"8" THEN 330
520 IF Y8$>Y9$ THEN 330
530 FOR TC=VAL(Y8$)TO VAL(Y9$):: Y5(TC)=1 :: NEXT TC
540 CALL CLEAR
550 INPUT "Do you want specific types (<<ENTER>>=No)":A$
560 IF SEG$(A$,1,1)="Y" OR SEG$(A$,1,1)="y" THEN 580
570 FOR TC=1 TO 15 :: T5(TC)=1 :: NEXT TC :: GOTO 680
580 CALL CLEAR
590 PRINT "Enter the desired types as follows:"
600 FOR TC=1 TO 15
610 PRINT TC;M$(TC)
620 NEXT TC

```



```

630 PRINT "Enter 999 if no more restrictions."
640 PRINT "Enter a type you want";
650 INPUT TC
660 IF TC=999 THEN 680
670 T5(TC)=1 :: TB=TB+5 :: GOTO 650
680 CALL CLEAR :: TB=0
690 ' now for ratings
700 INPUT "Do you want specific ratings (<<ENTER>>=No):":A$
710 IF SEG$(A$,1,1)="Y" OR SEG$(A$,1,1)="y" THEN 730
720 FOR TC=0 TO 3 :: C5(TC)=1 :: NEXT TC :: GOTO 830
730 PRINT "Enter your desited ratings as follows:"
740 FOR TC=0 TO 3
750 PRINT TAB(5);TC;"-";N$(TC)
760 NEXT TC
770 PRINT TAB(4);"999 - no more restrictions"
780 INPUT TC
790 IF TC<>-1 OR TC=999 THEN 810
800 FOR C=0 TO 3 :: C5(C)=1 :: NEXT C :: GOTO 830
810 IF TC=999 THEN 830
820 C5(TC)=1 :: TB=TB+5 :: GOTO 780
830 FOR TC=1 TO 15
840 IF T5(TC)=0 OR T2(TC)=0 THEN 900
850 A=T2(TC)
860 IN=A :: GOSUB 1070
870 IF Y5(VAL(Y1$))=0 OR C5(VAL(C1$))=0 THEN 890
880 GOSUB 1100
890 IF VAL(TY$)<>0 THEN A=VAL(TY$):: GOTO 860
900 NEXT TC
910 CALL CLEAR
920 PRINT "Those were all of the movies"
930 PRINT "that I have on file with those restrictions."
940 PRINT :: PRINT
950 INPUT "Do you want to access the file again?":A$
960 IF SEG$(A$,1,1)="Y" OR SEG$(A$,1,1)="y" THEN 980
970 GOTO 1240
980 FOR TC=1 TO 8 :: Y5(TC)=0 :: NEXT TC
990 FOR TC=0 TO 3 :: C5(TC)=0 :: NEXT TC
1000 FOR TC=1 TO 15 :: T5(TC)=0 :: NEXT TC
1010 GOTO 320
1020 !To read the pointer record
1030 INPUT #1,REC 1:RC,:: FOR I=0 TO 3 :: INPUT #1:C2$(I),:: NEXT I
1040 FOR I=1 TO 15 :: INPUT #1:T2$(I),:: NEXT I
1050 FOR I=1 TO 8 :: INPUT #1:Y2$(I),:: NEXT I ::
1060 RETURN
1070 !To read a data record
1080 INPUT #1,REC IN:LL$,RL$,NA$,AC$,YR$,CR$,TY$,Y1$,C1$,T1$,PC$
1090 RETURN
1100 CALL CLEAR
1110 PRINT NA$
1120 PRINT "Rec. #";IN
1130 PRINT "Actors:";AC$
1140 PRINT "Released in:";" 19";SEG$(Y1$,1,1);"0's"
1150 PRINT "Rating:";N$(VAL(C1$))
1160 PRINT "Type of movie:";M$(VAL(T1$))
1170 PRINT "Personal comment:";PC$
1180 PRINT
1190 PRINT "Next record of same:"
1200 PRINT "Rating-";VAL(CR$)
1210 PRINT "Decade-";VAL(YR$)
1220 PRINT "Type-";VAL(TY$)
1230 PRINT "<<ENTER>>";: INPUT A$ :: RETURN
1240 CLOSE #1 :: RUN "DSK1.MOVIE"

```

Sorted Order
Display

The last two programs provide for two different procedures to produce output. The first, LISTLG, lists all movies in the database in any one of several orders. See the output following the listing.

```
10 REM filename: "listlg"
20 REM purpose: Print the list of movies in different order
30 REM author: hmz, spg, jpg & jdr
40 REM -----
50 OPEN #1:"DSK1.MOV DAT",RELATIVE,INTERNAL,UPDATE,FIXED 254
60 DIM M$(16),R$(4),T2$(15),T2(15),Y2$(8),Y2(8),STK(20)
70 OPEN #9:"RS232"
80 FOR I=1 TO 15 :: READ M$(I):: NEXT I
90 DATA adventure,biblical epic,biography,childern
100 DATA documentary,horror,musical,science fiction
110 DATA comedy,crime-dective,disaster,drama
120 DATA travel,war,western
130 FOR I=0 TO 3 :: READ R$(I):: NEXT I
140 DATA -,Fair to poor,Good,Excellent
150 GOSUB 740
160 ! set up the arrays of secondary links
170 ! using the data on the first record
180 FOR J=0 TO 3 :: C2(J)=VAL(C2$(J)):: NEXT J
190 FOR J=1 TO 15 :: T2(J)=VAL(T2$(J)):: NEXT J
200 FOR J=1 TO 8 :: Y2(J)=VAL(Y2$(J)):: NEXT J
210 CALL CLEAR :: PRINT "List the movies in..."
220 PRINT "<1>-the order entered"
230 PRINT "<2>-alphabetical order"
240 PRINT "<3>-order of their type"
250 PRINT "<4>-order of their rating"
260 PRINT "<5>-order of decade released"
270 PRINT "<6>-return to main menu"
280 PRINT :: INPUT "Which activity?":A
290 IF A<1 OR A>5 THEN 1190
300 ON A GOSUB 320,370,470,560,650
310 GOTO 210
320 ! LIST MOVIES IN THE ENTERED ORDER
330 GOSUB 820 ! get and set the printing keys
340 FOR TC=2 TO RC
350 GOSUB 970
360 NEXT TC :: RETURN
370 ! list movies in alphabetical order
380 GOSUB 820
390 TC=2 :: T=0 :: IN=2 :: GOSUB 790
400 T=T+1
410 STK(T)=TC
420 IF TC<>0 THEN IN=TC :: GOSUB 790 :: TC=VAL(LL*): GOTO 400
430 T=T-1
440 IF T=0 THEN RETURN
450 TC=STK(T):: GOSUB 970
460 TC=VAL(RL*): GOTO 410
470 ! list movies in order of type
480 GOSUB 820
490 FOR T8=1 TO 15
500 IF T2(T8)=0 THEN 540
510 TC=T2(T8)
520 GOSUB 970
530 IF VAL(TY* )<>0 THEN TC=VAL(TY*): GOTO 520
540 NEXT T8
550 RETURN
560 ! list movies in order of rating
570 GOSUB 820
580 FOR T8=0 TO 3
590 IF C2(T8)=0 THEN 630
600 TC=C2(T8)
610 GOSUB 970
620 IF VAL(CR* )<>0 THEN TC=VAL(CR*): GOTO 610
630 NEXT T8
```

```

640 RETURN
650 ! list movies in order of decade released
660 GOSUB 820
670 FOR TB=1 TO 8
680 IF Y2(TB)=0 THEN 720
690 TC=Y2(TB)
700 GOSUB 970
710 IF VAL(YR#)<>0 THEN TC=VAL(YR#):: GOTO 700
720 NEXT TB
730 RETURN
740 'To read the pointer record
750 INPUT #1,REC 1:RC,,: FOR I=0 TO 3 :: INPUT #1:C2$(I),,: NEXT I
760 FOR I=1 TO 15 :: INPUT #1:T2$(I),,: NEXT I
770 FOR I=1 TO 8 :: INPUT #1:Y2$(I),,: NEXT I ::
780 RETURN
790 'To read a data record
800 INPUT #1,REC IN:LL$,RL$,NA$,AC$,YR$,CR$,TY$,Y1$,C1$,T1$,PC$
810 RETURN
820 CALL CLEAR :: PRINT
830 PRINT "<1>-output only to screen"
840 PRINT "<2>-output only to printer"
850 PRINT "<3>-to both screen and printer"
860 PRINT :: INPUT "Which do you want?":A
870 IF A<1 OR A>3 THEN 820
880 IF A=1 THEN SC=1 :: PR=0 :: GOTO 960
890 PRINT #9:" #   Date   CR Type           Description of movie"
900 PRINT #9:" -   ----   - - - - -   -----"
910 IF A=2 THEN SC=0 :: PR=1
920 IF A=3 THEN SC=1 :: PR=1
930 PRINT :: PRINT
940 INPUT "Print the comments?":A$
950 IF SEG$(A$,1,1)="Y" OR SEG$(A$,1,1)="y" THEN PC=1 ELSE PC=0
960 RETURN
970 IN=TC :: GOSUB 790
980 IF PR=1 THEN GOSUB 1120
990 IF SC=1 THEN GOSUB 1010
1000 RETURN
1010 CALL CLEAR :: PRINT
1020 PRINT NA$
1030 PRINT "Rec #":TC
1040 PRINT "Actors:":AC$
1050 PRINT "Released in: 19":SEG$(Y1$,1,1);"0's"
1060 PRINT "Rating:":R$(VAL(C1$))
1070 PRINT "Type of movie:":M$(VAL(T1$))
1080 PRINT "Personal comment:":PC$
1090 PRINT
1100 PRINT
1110 INPUT "<<ENTER>>":A$ :: RETURN
1120 Y=VAL(Y1$):: N$=NA$
1130 PRINT #9:TC;TAB(5);"19";Y1$;"0'S";TAB(12);SEG$(R$(VAL(C1$)),1,1);TAB(15);SE
G$(M$(VAL(T1$)),1,3);
1140 PRINT #9:TAB(21);N$;
1150 PRINT #9:"--";AC$
1160 IF PC=0 THEN 1180
1170 PRINT #9:TAB(25);"Comment: ";PC$
1180 RETURN
1190 CLOSE #9 :: CLOSE #1 :: RUN "DSK1.MOVIE"

```

#	Date	CR	Type	Description of movie
2	1980'S	G	hor	Shinning,The--J.Nicholson, S.Duvall
3	1980'S	G	com	Airplane--Star studded cast
4	1970'S	G	com	Heaven Can Wait--W.Beatty
5	1970'S	G	com	Oh, God!--J.Denver, G.Burns
6	1980'S	F	com	How to Beat the High Cost of Living--
7	1970'S	G	com	Love at First Bite--G.Hamilton
8	1980'S	F	hor	Blood Eaters--
9	1970'S	G	hor	Invasion of the Body Snatchers--
10	1980'S	E	sci	Empire Strikes Back,The--M.Hamill, H.Ford
11	1970'S	E	sci	Star Wars--M.Hamill, A.Guiness
12	1980'S	G	com	Cheech and Chong's Next Movie --Cheech and Chong
13	1980'S	G	com	Blues Brothers--J.Belushi, D.Akroyd
14	1970'S	G	adv	Deep,The--R.Shaw, J.Bisset
15	1970'S	G	adv	Jaws--R.Scheider, R.Dryfuss
16	1970'S	G	adv	King Kong--J.Bridges, J.Lang
17	1970'S	G	com	Monty Python and the Holy Grail--Monty Python
18	1970'S	G	com	Murder by Death--P.Sellers, T.Capote
19	1970'S	G	sci	Logan's Run--M.York, R.Jordan
20	1970'S	E	dra	Rocky--S.Stallone, T.Shire
21	1970'S	G	mus	Semi-Tough--B.Reynolds, K.Kristofferson
22	1970'S	G	chi	Shaggy D.A.--S.Pleshette, D.Jones
23	1970'S	E	wes	Shootist,The--J.Wayne, L.Bacall
24	1970'S	G	com	Smokey and the Bandits--B.Reynolds, J.Reed
25	1970'S	G	com	W.W. and the Dixie Dancekings --B.Reynolds
26	1970'S	G	sci	Silent Running--B.Dern, C.Potts
27	1960'S	E	com	M.A.S.H.--E.Gould
28	1980'S	E	bio	Coal Miner's Daughter--S.Spacek
29	1980'S	E	com	Being There--P.Sellers
30	1980'S	G	com	Ten--B.Derek
31	1970'S	G	chi	101 Dalmations--Animated
32	1970'S	E	chi	Benji--Benji the dog
33	1940'S	E	chi	Fantasia--Animated
34	1970'S	E	dra	Rocky II--S.Stallone, T.Shire
35	1970'S	G	com	Young Frankenstein--G.Wilder, P.Boyle
36	1970'S	G	adv	Superman--C.Reeves
37	1980'S	E	com	Life of Brian,The--Monty Python troupe
38	1940'S	E	chi	Wizard of Oz,The--J.Garland
39	1960'S	E	dra	To Sir, with Love--S.Poitier
40	1970'S	G	hor	Exorcist,The--L.Blair
41	1970'S	G	com	Animal House--J.Belushi
42	1960'S	E	sci	2001: A Space Odyssey--HAL, K.Dullea
43	1970'S	G	com	Blazing Saddles--C.Little, R.Morse
44	1960'S	G	hor	Birds,The--J.Tandy, R.Taylor
45	1970'S	G	chi	Boatniks,The--S.Powers, R.Morse
46	1970'S	E	dra	Bless the Beasts and the Childern--R.Mummy
47	1970'S	G	com	Frisco Kid,The--G.Wilder, H.Ford
48	1980'S	G	com	Foolin' Around--G.Busey, T.Randall
49	1960'S	G	dra	Guns of Navarone--G.Peck, D.Niven

#	Date	CR	Type	Description of movie
31	1970'S	G	chi	101 Dalmations--Animated
42	1960'S	E	sci	2001: A Space Odyssey--HAL, K.Dullea
3	1980'S	G	com	Airplane--Star studded cast
41	1970'S	G	com	Animal House--J.Belushi
29	1980'S	E	com	Being There--P.Sellers
32	1970'S	E	chi	Benji--Benji the dog
44	1960'S	G	hor	Birds,The--J.Tandy, R.Taylor
43	1970'S	G	com	Blazing Saddles--C.Little, R.Morse
46	1970'S	E	dra	Bless the Beasts and the Childern--R.Mummy
8	1980'S	F	hor	Blood Eaters--
13	1980'S	G	com	Blues Brothers--J.Belushi, D.Akroyd
45	1970'S	G	chi	Boatniks,The--S.Powers, R.Morse
12	1980'S	G	com	Cheech and Chong's Next Movie --Cheech and Chong
28	1980'S	E	bio	Coal Miner's Daughter--S.Spacek
14	1970'S	G	adv	Deep,The--R.Shaw, J.Bisset
10	1980'S	E	sci	Empire Strikes Back,The--M.Hamill, H.Ford
40	1970'S	G	hor	Exorcist,The--L.Blair
33	1940'S	E	chi	Fantasia--Animated
48	1980'S	G	com	Foolin' Around--G.Busey, T.Randall
47	1970'S	G	com	Frisco Kid,The--G.Wilder, H.Ford
49	1960'S	G	dra	Guns of Navarone--G.Peck, D.Niven
4	1970'S	G	com	Heaven Can Wait--W.Beatty
6	1980'S	F	com	How to Beat the High Cost of Living--
9	1970'S	G	hor	Invasion of the Body Snatchers--
15	1970'S	G	adv	Jaws--R.Scheider, R.Dryfuss
16	1970'S	G	adv	King Kong--J.Bridges, J.Lang
37	1980'S	E	com	Life of Brian,The--Monty Python troupe
19	1970'S	G	sci	Logan's Run--M.York, R.Jordan
7	1970'S	G	com	Love at First Bite--G.Hamilton
27	1960'S	E	com	M.A.S.H.--E.Gould
17	1970'S	G	com	Monty Python and the Holy Grail--Monty Python
18	1970'S	G	com	Murder by Death--P.Sellers, T.Capote
5	1970'S	G	com	Oh, God!--J.Denver, G.Burns
20	1970'S	E	dra	Rocky--S.Stallone, T.Shire
34	1970'S	E	dra	Rocky II--S.Stallone, T.Shire
21	1970'S	G	mus	Semi-Tough--B.Reynolds, K.Kristofferson
22	1970'S	G	chi	Shaggy D.A.--S.Pleshette, D.Jones
2	1980'S	G	hor	Shinning,The--J.Nicholson, S.Duvall
23	1970'S	E	wes	Shootist,The--J.Wayne, L.Bacall
26	1970'S	G	sci	Silent Running--B.Dern, C.Potts
24	1970'S	G	com	Smokey and the Bandits--B.Reynolds, J.Reed
11	1970'S	E	sci	Star Wars--M.Hamill, A.Guinness
36	1970'S	G	adv	Superman--C.Reeves
30	1980'S	G	com	Ten--B.Derek
39	1960'S	E	dra	To Sir, with Love--S.Poitier
25	1970'S	G	com	W.W. and the Dixie Dancekings --B.Reynolds
38	1940'S	E	chi	Wizard of Oz,The--J.Garland
35	1970'S	G	com	Young Frankenstein--G.Wilder, P.Boyle

#	Date	CR	Type	Description of movie
14	1970'S	G	adv	Deep, The--R. Shaw, J. Bisset
15	1970'S	G	adv	Jaws--R. Scheider, R. Dryfuss
16	1970'S	G	adv	King Kong--J. Bridges, J. Lang
36	1970'S	G	adv	Superman--C. Reeves
28	1980'S	E	bio	Coal Miner's Daughter--S. Spacek
22	1970'S	G	chi	Shaggy D.A.--S. Pleshette, D. Jones
31	1970'S	G	chi	101 Dalmations--Animated
32	1970'S	E	chi	Benji--Benji the dog
33	1940'S	E	chi	Fantasia--Animated
38	1940'S	E	chi	Wizard of Oz, The--J. Garland
45	1970'S	G	chi	Boatniks, The--S. Powers, R. Morse
2	1980'S	G	hor	Shinning, The--J. Nicholson, S. Duvall
8	1980'S	F	hor	Blood Eaters--
9	1970'S	G	hor	Invasion of the Body Snatchers--
40	1970'S	G	hor	Exorcist, The--L. Blair
44	1960'S	G	hor	Birds, The--J. Tandy, R. Taylor
21	1970'S	G	mus	Semi-Tough--B. Reynolds, K. Kristofferson
10	1980'S	E	sci	Empire Strikes Back, The--M. Hamill, H. Ford
11	1970'S	E	sci	Star Wars--M. Hamill, A. Guinness
19	1970'S	G	sci	Logan's Run--M. York, R. Jordan
26	1970'S	G	sci	Silent Running--B. Dern, C. Potts
42	1960'S	E	sci	2001: A Space Odyssey--HAL, K. Dullea
3	1980'S	G	com	Airplane--Star studded cast
4	1970'S	G	com	Heaven Can Wait--W. Beatty
5	1970'S	G	com	Oh, God!--J. Denver, G. Burns
6	1980'S	F	com	How to Beat the High Cost of Living--
7	1970'S	G	com	Love at First Bite--G. Hamilton
12	1980'S	G	com	Cheech and Chong's Next Movie --Cheech and Chong
13	1980'S	G	com	Blues Brothers--J. Belushi, D. Akroyd
17	1970'S	G	com	Monty Python and the Holy Grail--Monty Python
18	1970'S	G	com	Murder by Death--P. Sellers, T. Capote
24	1970'S	G	com	Smokey and the Bandits--B. Reynolds, J. Reed
25	1970'S	G	com	W.W. and the Dixie Dancekings --B. Reynolds
27	1960'S	E	com	M.A.S.H.--E. Gould
29	1980'S	E	com	Being There--P. Sellers
30	1980'S	G	com	Ten--B. Derek
35	1970'S	G	com	Young Frankenstein--G. Wilder, P. Boyle
37	1980'S	E	com	Life of Brian, The--Monty Python troupe
41	1970'S	G	com	Animal House--J. Belushi
43	1970'S	G	com	Blazing Saddles--C. Little, R. Morse
47	1970'S	G	com	Frisco Kid, The--G. Wilder, H. Ford
48	1980'S	G	com	Foolin' Around--G. Busey, T. Randall
20	1970'S	E	dra	Rocky--S. Stallone, T. Shire
34	1970'S	E	dra	Rocky II--S. Stallone, T. Shire
39	1960'S	E	dra	To Sir, with Love--S. Poitier
46	1970'S	E	dra	Bless the Beasts and the Children--R. Mummy
49	1960'S	G	dra	Guns of Navarone--G. Peck, D. Niven
23	1970'S	E	wes	Shootist, The--J. Wayne, L. Bacall

#	Date	CR	Type	Description of movie
6	1980'S	F	com	How to Beat the High Cost of Living--
8	1980'S	F	hor	Blood Eaters--
2	1980'S	G	hor	Shinning,The--J.Nicholson, S.Duvall
3	1980'S	G	com	Airplane--Star studded cast
4	1970'S	G	com	Heaven Can Wait--W.Beatty
5	1970'S	G	com	Oh, God!--J.Denver, G.Burns
7	1970'S	G	com	Love at First Bite--G.Hamilton
9	1970'S	G	hor	Invasion of the Body Snatchers--
12	1980'S	G	com	Cheech and Chong's Next Movie --Cheech and Chong
13	1980'S	G	com	Blues Brothers--J.Belushi, D.Akroyd
14	1970'S	G	adv	Deep,The--R.Shaw, J.Bisset
15	1970'S	G	adv	Jaws--R.Scheider, R.Dryfuss
16	1970'S	G	adv	King Kong--J.Bridges, J.Lang
17	1970'S	G	com	Monty Python and the Holy Grail--Monty Python
18	1970'S	G	com	Murder by Death--P.Sellers, T.Capote
19	1970'S	G	sci	Logan's Run--M.York, R.Jordan
21	1970'S	G	mus	Semi-Tough--B.Reynolds, K.Kristofferson
22	1970'S	G	chi	Shaggy D.A.--S.Pleshette, D.Jones
24	1970'S	G	com	Smokey and the Bandits--B.Reynolds, J.Reed
25	1970'S	G	com	W.W. and the Dixie Dancekings --B.Reynolds
26	1970'S	G	sci	Silent Running--B.Dern, C.Potts
30	1980'S	G	com	Ten--B.Derek
31	1970'S	G	chi	101 Dalmations--Animated
35	1970'S	G	com	Young Frankenstein--G.Wilder, P.Boyle
36	1970'S	G	adv	Superman--C.Reeves
40	1970'S	G	hor	Exorcist,The--L.Blair
41	1970'S	G	com	Animal House--J.Belushi
43	1970'S	G	com	Blazing Saddles--C.Little, R.Morse
44	1960'S	G	hor	Birds,The--J.Tandy, R.Taylor
45	1970'S	G	chi	Boatniks,The--S.Powers, R.Morse
47	1970'S	G	com	Frisco Kid,The--G.Wilder, H.Ford
48	1980'S	G	com	Foolin' Around--G.Busey, T.Randall
49	1960'S	G	dra	Guns of Navarone--G.Peck, D.Niven
10	1980'S	E	sci	Empire Strikes Back,The--M.Hamill, H.Ford
11	1970'S	E	sci	Star Wars--M.Hamill, A.Guinness
20	1970'S	E	dra	Rocky--S.Stallone, T.Shire
23	1970'S	E	wes	Shootist,The--J.Wayne, L.Bacall
27	1960'S	E	com	M.A.S.H.--E.Gould
28	1980'S	E	bio	Coal Miner's Daughter--S.Spacek
29	1980'S	E	com	Being There--P.Sellers
32	1970'S	E	chi	Benji--Benji the dog
33	1940'S	E	chi	Fantasia--Animated
34	1970'S	E	dra	Rocky II--S.Stallone, T.Shire
37	1980'S	E	com	Life of Brian,The--Monty Python troupe
38	1940'S	E	chi	Wizard of Oz,The--J.Garland
39	1960'S	E	dra	To Sir, with Love--S.Poitier
42	1960'S	E	sci	2001: A Space Odyssey--HAL, K.Dullea
46	1970'S	E	dra	Bless the Beasts and the Childern--R.Mummy

#	Date	CR	Type	Description of movie
33	1940'S	E	chi	Fantasia--Animated
38	1940'S	E	chi	Wizard of Oz,The--J.Garland
27	1960'S	E	com	M.A.S.H.--E.Gould
39	1960'S	E	dra	To Sir, with Love--S.Poitier
42	1960'S	E	sci	2001: A Space Odyssey--HAL, K.Dullea
44	1960'S	G	hor	Birds,The--J.Tandy, R.Taylor
49	1960'S	G	dra	Guns of Navarone--G.Peck, D.Niven
4	1970'S	G	com	Heaven Can Wait--W.Beatty
5	1970'S	G	com	Oh, God!--J.Denver, G.Burns
7	1970'S	G	com	Love at First Bite--G.Hamilton
9	1970'S	G	hor	Invasion of the Body Snatchers--
11	1970'S	E	sci	Star Wars--M.Hamill, A.Guinness
14	1970'S	G	adv	Deep,The--R.Shaw, J.Bisset
15	1970'S	G	adv	Jaws--R.Scheider, R.Dryfuss
16	1970'S	G	adv	King Kong--J.Bridges, J.Lang
17	1970'S	G	com	Monty Python and the Holy Grail--Monty Python
18	1970'S	G	com	Murder by Death--P.Sellers, T.Capote
19	1970'S	G	sci	Logan's Run--M.York, R.Jordan
20	1970'S	E	dra	Rocky--S.Stallone, T.Shire
21	1970'S	G	mus	Semi-Tough--B.Reynolds, K.Kristofferson
22	1970'S	G	chi	Shaggy D.A.--S.Pleshette, D.Jones
23	1970'S	E	wes	Shootist,The--J.Wayne, L.Bacall
24	1970'S	G	com	Smokey and the Bandits--B.Reynolds, J.Reed
25	1970'S	G	com	W.W. and the Dixie Dancekings --B.Reynolds
26	1970'S	G	sci	Silent Running--B.Dern, C.Potts
31	1970'S	G	chi	101 Dalmations--Animated
32	1970'S	E	chi	Benji--Benji the dog
34	1970'S	E	dra	Rocky II--S.Stallone, T.Shire
35	1970'S	G	com	Young Frankenstein--G.Wilder, P.Boyle
36	1970'S	G	adv	Superman--C.Reeves
40	1970'S	G	hor	Exorcist,The--L.Blair
41	1970'S	G	com	Animal House--J.Belushi
43	1970'S	G	com	Blazing Saddles--C.Little, R.Morse
45	1970'S	G	chi	Boatniks,The--S.Powers, R.Morse
46	1970'S	E	dra	Bless the Beasts and the Children--R.Mummy
47	1970'S	G	com	Frisco Kid,The--G.Wilder, H.Ford
2	1980'S	G	hor	Shinning,The--J.Nicholson, S.Duvall
3	1980'S	G	com	Airplane--Star studded cast
6	1980'S	F	com	How to Beat the High Cost of Living--
8	1980'S	F	hor	Blood Eaters--
10	1980'S	E	sci	Empire Strikes Back,The--M.Hamill, H.Ford
12	1980'S	G	com	Cheech and Chong's Next Movie --Cheech and Chong
13	1980'S	G	com	Blues Brothers--J.Belushi, D.Akroyd
28	1980'S	E	bio	Coal Miner's Daughter--S.Spacek
29	1980'S	E	com	Being There--P.Sellers
30	1980'S	G	com	Ten--B.Derek
37	1980'S	E	com	Life of Brian,The--Monty Python troupe
48	1980'S	G	com	Foolin' Around--G.Busey, T.Randall

Physical Record
Display

The last program, LISTER, is a simplified version of the previous program, LISTLG. Its purpose is to display selected records neatly. The user selects the records to be displayed according to physical record numbers which correspond to their original order of entry in the file. It can also display linking information by showing the first occurrence of a secondary link.

```
10 REM filename: "lister"
20 REM purpose: Print the selected movies neatly
30 REM author: hmz, spg, jpg & jdr
40 REM -----
50 OPEN #1:"DSK1.MOVDAT",RELATIVE,INTERNAL,UPDATE,FIXED 254
60 DIM M$(16),T$(15),Y2$(8),Y2(8),B$(2)
70 FOR I=1 TO 15 :: READ M$(I):: NEXT I
80 DATA adventure,biblical epic,biography,childern
90 DATA documentary,horror,musical,science fiction
100 DATA comedy,crime-dective,disaster,drama
110 DATA travel,war,western
120 FOR I=0 TO 3 :: READ R$(I):: NEXT I
130 DATA unknown,Fair to poor,Good,Excellent
140 CALL CLEAR :: PRINT :: PRINT
150 PRINT "<1>-view all occurrence data"
160 PRINT "<2>-look at one movie"
170 PRINT "<3>-look at some movies"
180 PRINT "<4>-look at all movies"
190 PRINT "<5>-return to main program"
200 PRINT :: INPUT "Which activity?":A
210 IF A<1 OR A>5 THEN 140
220 ON A GOSUB 240,410,460,540,730
230 GOTO 140
240 GOSUB 740
250 FOR J=0 TO 3 :: C2(J)=VAL(C2$(J)):: NEXT J
260 FOR J=1 TO 15 :: T2(J)=VAL(T2$(J)):: NEXT J
270 FOR J=1 TO 8 :: Y2(J)=VAL(Y2$(J)):: NEXT J
280 CALL CLEAR :: PRINT :: PRINT
290 PRINT TAB(5);"O c c u r r e n c e  D a t a"
300 PRINT TAB(5);"-----"
310 PRINT :: PRINT
320 PRINT TAB(13);"Subscript #"
330 FOR J=0 TO 15 :: PRINT USING "###":J:: NEXT J
340 PRINT :: PRINT RPT$("=",28):: PRINT "C2(j)";
350 FOR J=0 TO 3 :: PRINT C2(J):: NEXT J
360 PRINT :: PRINT "T2(j) ";
370 FOR J=1 TO 15 :: PRINT T2(J):: NEXT J
380 PRINT :: PRINT "Y2(j) ";
390 FOR J=1 TO 8 :: PRINT Y2(J):: NEXT J
400 PRINT :: PRINT :: INPUT "<<enter>>":A$ :: RETURN
410 GOSUB 740 :: PRINT :: PRINT "There are";RC-1;" movie records."
420 PRINT "They are numbered 2 through";RC;". "
430 INPUT "Which do you want to see?":A
440 IF A<2 OR A>RC THEN RETURN
450 GOSUB 560 :: RETURN
460 GOSUB 740 :: PRINT "There are";RC-1;" movie records."
470 PRINT "They are numbered 2 through";RC;". "
480 PRINT "This routine allows you to"
490 PRINT "see records X through Y."
500 INPUT "What are X and Y(in the form X,Y)?:":X,Y
510 IF X<2 OR Y<2 OR X>Y OR Y>RC OR X>RC THEN 500
520 FOR A=X TO Y :: GOSUB 560 :: NEXT A
530 RETURN
540 GOSUB 740 :: FOR A=2 TO RC :: GOSUB 560 :: NEXT A
550 RETURN
```

```

560 GOSUB 740
570 ! get a data record
580 IN=A :: GOSUB 790 :: CALL CLEAR
590 PRINT "Record #";A
600 PRINT "Left link-";VAL(LL$);
610 PRINT TAB(14);"Right link-";VAL(RL$)
620 PRINT "Next record of same:"
630 PRINT "Rating-";VAL(CR$);" Decade-";VAL(YR$)
640 PRINT "Type-";VAL(TY$):: PRINT
650 PRINT "Movie title: ";NA$
660 PRINT "Actors: ";AC$
670 PRINT "Decade of release: 19";SEG$(Y1$,1,1);"0's"
680 PRINT "Consumer union rating:";
690 PRINT R$(VAL(C1$))
700 PRINT "Type of movie: ";M$(VAL(T1$))
710 PRINT "Personal comment: ";PC$
720 PRINT :: INPUT "<<ENTER>>":A$ :: RETURN
730 CLOSE #1 :: RUN "DSK1.MOVIE"
740 !To read the pointer record
750 INPUT #1,REC 1:RC,:: FOR I=0 TO 3 :: INPUT #1:C2$(I),:: NEXT I
760 FOR I=1 TO 15 :: INPUT #1:T2$(I),:: NEXT I
770 FOR I=1 TO 8 :: INPUT #1:Y2$(I),:: NEXT I ::
780 RETURN
790 !To read a data record
800 INPUT #1,REC IN:LL$,RL$,NA$,AC$,YR$,CR$,TY$,Y1$,C1$,T1$,PC$
810 RETURN
820 END

```

Final Thoughts

It is instructive at this time to note that many other data management techniques exist. In every chapter we have endeavored to indicate to you some of the other methods. We feel, though, that a thorough understanding of the techniques we have shown here are more than enough to give you the essential skills for practically all industrial applications programming. If there is one significant difference between the programs in this book and those in industry, it is that the latter are more customized to a particular application and client. Ours have tended toward the skeletal because we feel that given these bones you can flesh out any one or a combination of them to satisfy the requirements of the most demanding applications.

We wish you success in your efforts at managing information with a computer. The techniques are not simple, and because they tend toward the complex they are all the more interesting. The future holds more discoveries in data management, and like you, we look forward to using them.

Index

- ACCESS 138
- ACDC 13
- Acey-ducey 13
- ANIMAL 108
- Aphorism generator 10
- Arrays 1, 6
- Artificial intelligence 2

- B-tree 31
- BASIC compiler 102
- BBLSORT 21, 25
- BELLCURV 17
- BLIP 10
- BOSTON 124
- BRFRSORT 21, 22
- BSST on disk 122
- BSST sort 31
- BSST stack 56
- BSST 22, 114, 124
- Bell-shaped curve 17
- Binary Sequence Search Tree (BSST) 114
- Binary search, maximum accesses 91
- Binary search 21, 90
- Binary sorts 22, 27
- Binary trees 108
- Block, H. D. 2
- Blocking records 91
- Boston subway 124
- Bottom (of stack) 55
- Brute force sorts 22
- Bubble sort 25

- CALL KEY 3
- COMPSORT 35
- Circular doubly linked list application 62
- Circular queue 58
- Circularly linked list and tree 124
- Circularly linked lists 62
- Clusters (IBM DOS) 103
- Codes 46
- Collision (hashing) 93
- Compiler, BASIC 102
- Compilers 40
- Computer games 39
- Critchfield, M. 26
- Cryptogram generation 46

- DACCSORT 96
- DBLKEY 116
- DELXSORT 22, 26
- DOS physical characteristics 103
- Daily transaction file 78
- Degenerate tree 117
- Delayed exchange (selection) sort 26
- Deletion and balancing, inverted files 135
- Deletion, stack 56
- Deque deletion 59
- Deque insertion 59
- Dequeues 55, 59
- Detached key sort 98
- Direct access 89
- Direct access files 69, 89
- Directed scan, ordered list 70
- Directed scan, unordered list 70
- Disk drives 89
- Disk sort 96
- Distribution, normal 17
- Double-ended queue (deque) 59
- Double-key BSST 116
- Doubly linked lists 61
- Dwyer, T. 26

- ENTER 3
- EXCHSORT 22, 26
- Encryption 46
- Estate distribution 8
- Exchange sorts 22, 24, 26

- FIFO list 57
- FOR-NEXT 2
- FORTRAN 1, 2
- File access using pointer table 134
- File pointer 90
- File record number 90
- File searching 90
- First In First Out (FIFO) list 57
- Five-letter word game 40
- Front pointer (queue) 57

- GIA 2, 3
- GET 90
- GPTOT 76
- GPTOTSUB 79
- GRAFCODE 46
- Graphing word size 50
- Grillo, J. 37
- Grillo, S. 82, 131
- Group totals 75

- HASHING 93
- HEAPSORT 22, 30
- Hash address processing 93
- Hashing functions 93
- Heap sort 30
- Heuristic programming 2

- IBM DOS 103
- INDEXBLD 81
- INDEXPRT 85
- INHERIT 8
- INPUT mode 69
- INSEARCH 93
- INSERT 135

- INSERT 21, 24
- ISAM access 104
- ISAM file processing 102
- ISAM insertion 105
- ISAM storage areas 103
- ISAM structuring 103
- ISAM 89
- In between game 13
- In-memory, double-key BSST 116
- In-memory, multi-key BSST 120
- In-memory, single-key BSST 115
- Index area (ISAM) 103
- Index printing 85
- Index production 81
- Indexed sequential access method (ISAM) 89, 102
- Indirect addressing 2
- Insertion sort 24
- Insertion, stack 56
- Interpolation search 92
- Inverted file record contents 133
- Inverted file record structure 132
- Inverted files 131

- JOBSTEPS 6
- JOTTO 40
- Jargon generator (SIMP) 12

- KWIC index 47
- KWICINDX 47
- Knuth, D. E. 22, 37

- LIFO list 57
- LINKLIST 60
- LISTER 148
- LISTLG 141
- Last in first out (LIFO) list 57
- Linear lists 55
- Linked list insertion 61
- Linked lists 55, 60
- Links, Y, C, and T 133
- Links, YES and NO 111
- Links, left and right 31
- Listing, ACCESS 139
- Listing, ACDC 13
- Listing, BBLSORT 25
- Listing, BELLCURV 18
- Listing, BLIP 10
- Listing, BRFRSORT 23
- Listing, COMPSORT 35
- Listing, DACCSORT 96
- Listing, DBLKEY 117
- Listing, DELXSORT 27
- Listing, EXCHSORT 26
- Listing, GIA 3
- Listing, GPTOT 76
- Listing, GPTOTSUB 79
- Listing, HASHING 94
- Listing, HEAPSORT 30

- Listing, INDEXBLD 82
- Listing, INDEXPRT 85
- Listing, INHERIT 8
- Listing, INSEARCH 93
- Listing, INSERT 135
- Listing, INRSORT 24
- Listing, JOBSTEPS 7
- Listing, JOTTO 40
- Listing, KWICINDX 48
- Listing, LISTER 148
- Listing, LISTLG 141
- Listing, MASTRMND 44
- Listing, MOVIE 135
- Listing, MUSHSORT 33
- Listing, PICOFORM 42
- Listing, QUIKSORT 31
- Listing, SEARCH 92
- Listing, SEQWORDS 72
- Listing, SHELSORT 28
- Listing, SIMP 12
- Listing, SMETSORT 29
- Listing, SORTMERG 79
- Listing, STRBSST 115
- Listing, STRTREE 123
- Listing, SUBCODE 46
- Listing, TREE5KEY 120
- Listing, TREESORT 32
- Listing, VEGGIES 112
- Listing, WORDFREQ 50
- Logical order 61

- MASTRMND 42, 44
- MOVIE 135
- MUSHSORT 33
- Main program driver, movie system 134
- Merging 78
- Monte Carlo technique 6
- Movie system 135
- Multi-key BSST 120
- Multikey sorts 32
- Mushroom data 33

- Nijenhuis, A. 25, 37
- Nim 2
- Normal distribution of values 17
- Normal variates 17

- OPEN (file operation) 69
- OUTPUT mode 69
- Ordered list directed scan 70
- Overflow area (hashing) 94
- Overflow area (ISAM) 103, 105

- PICOFORM 42
- Pattern matching 42
- Physical order 61
- Physical record display 148
- Pico-Fermi-Bagels 42
- Playfair code 46
- Pointer scrambling 8
- Pointer tables 134
- Pointer, front (queue) 57
- Pointer, rear (queue) 57
- Pointer, stack 55

- Pointers 1
- Pop (a stack) 55
- Prime area (ISAM) 103
- Punch-card-oriented systems 22
- Push (a stack) 55

- QUIKSORT 22, 31
- Query 132
- Query processing, inverted files 138
- Query, multi-key 132
- Queue, circular 58
- Queues 55, 57
- Quicksort 56
- Quicksort stack 56

- Radix sort 22
- Random access, inverted files 138
- Random message selection 13
- Random selection from DATA 13
- Random text 10
- Random word selection 40
- Rear pointer (queue) 57
- Record address 90
- Record insertion, inverted files 135
- References on sorts 37
- Robertson, C. 62, 126
- Robertson, J. D. 10

- SCRIPSIT 40
- SEARCH 91, 92
- SEQWORDS 72
- SHELSORT 22, 28
- SIMP 10, 12
- SMETSORT 22, 28
- SORTMERG 79
- STRBSST 114, 115
- STRTREE 123
- SUBCODE 46
- Search, binary 90
- Search, interpolation 92
- Search, sequential 70
- Secondary keys 131
- Sectors (IBM DOS) 103
- Segmented detached key sort 100
- Selection sort 26
- Sequential access files 69
- Sequential file access 71
- Sequential file merging 78
- Sequential search techniques 70
- Shell sort 28
- Shell, D. 28
- Shell-Metzner sort 28, 32
- Shell-Metzner, direct access file 96
- Single-key BSST 115
- Singly linked lists 60
- Sort size 21
- Sort, BSST 31
- Sort, binary 27
- Sort, brute force 22
- Sort, bubble 25
- Sort, delayed exchange 26
- Sort, delayed selection 26
- Sort, detached key 22, 98
- Sort, disk 96
- Sort, exchange 22, 24, 26
- Sort, heap 30
- Sort, insertion 24
- Sort, multikey 32
- Sort, mushroom 33
- Sort, Quicksort 30
- Sort, radix 22
- Sort, segmented detached key 100
- Sort, Shell-Metzner 32, 96
- Sort, Shell 28
- Sort, string array 33
- Sort, tree 22, 31
- Sorted order display, inverted files 141
- Sorting categories 21
- Sorting comparison 35
- Sorting effectiveness 21
- Sorting efficiency 21, 35
- Sorting large files 96
- Sorting speed 21
- Sorting subroutines 35
- Sorts comparison chart 37
- Sorts, references 37
- Source program 40
- Stack array 30
- Stack pointer 55
- Stack, BSST 31
- Stack, Quicksort 30
- Stacks 55
- String array sorting 33
- Strings 39
- Sublists 27, 30
- Subscripted variables 2
- Subscripts 1
- Substitution code 46

- TREE5KEY 120
- TREESORT 22, 31
- Text analysis 50
- Text encoding 46
- Text reordering 47
- Top (of stack) 55
- Tracks (IBM DOS) 103
- Transaction files 78
- Tree and circularly linked list 124
- Tree sorts 22, 31
- Tree structures 107
- Tree, degenerate 117
- Trees 107
- Tries 107
- Trinary trees 107

- Underflow, stack 55, 56
- Unique keys 131
- Unordered list directed scan 70

- Volatile files 93

- WORDFREQ 50
- Word processing 39
- Word size frequency 50
- Worker scheduling 6

- YES and NO links 111
- Yob, G. 47

ABOUT THE AUTHORS

John P. Grillo

Dr. Grillo earned his Ph.D. in 1971 from the University of New Mexico. His professional experience includes consulting in the areas of word processing, mail-order systems, structured programming techniques, and database systems. He is a contributor to trade and professional journals, as well as coauthor of numerous books. He is a member of the Bentley College faculty in the Computer Information Systems Department.

J. D. Robertson

Dr. Robertson earned his Ph.D. in Computer Science in 1976 from the University of Southwestern Louisiana. He has been a consultant for book publishers, state agencies, various small businesses, and computerized farm management systems. He has written articles for business and professional journals and coauthored numerous books. He is a member of the Bentley College faculty in the Computer Information Systems Department

Henry M. Zbyszynski

Henry Zbyszynski received his Doctorate in Education from Boston University and his Masters in Education from Harvard. Currently he is Senior Faculty Consultant and Adjunct Assistant Professor of Computer Information Systems at Bentley College in Massachusetts.

wcb

Wm. C. Brown Publishers
Dubuque, Iowa

\$15.95

ISBN 0-697-00245-4