DATA FILE PROCESSING

Storing data on the ATARI 410™ Program Recorder

and the ATARI 810™ Disk Drive

1) Storing Data on Cassette
2) An Example of Cassette I/O: Cassette Mailing List
3) Storing Data on Disk
4) Example of Disk I/O: Disk Mailing List
5) Random Access

ATARI INC.
CONSUMER PRODUCT SERVICE
Product Support Group
1312 Crossman Avenue
Sunnyvale, CA 94086

(800) 672-1404 inside CA
(800) 538-8543 outside CA

DEMOPAC #2
Rev. 3 10-83

DISCLAIMER OF WARRANTY ON PROGRAMS CONTAINED HEREIN

All computer programs contained herein are distributed on an "as is" basis by Atari, Inc. ("Atari") without warranty of any kind. Any statements concerning the capabilities or utility of the computer programs are not to be construed as express or implied warranties.

Atari shall have no liability or responsibility to the user or any other person or entity with respect to any claim, loss, liability, or damage caused or alleged to be caused directly or indirectly by the computer programs, contained herein. The entire risk as to the quality and performance of such programs is with the user.

Every effort has been made to ensure the accuracy of this document. However, because of ongoing improvements and updating of our computer software and hardware, Atari cannot guarantee the accuracy of printed material after the date of publication and disclaims any liability for changes, errors, or omissions.

Correspondence regarding this pack should be forwarded to Manager of Technical Support, Consumer Product Service, Atari, Incorporated, 1312 Crossman Avenue, Sunnyvale, CA 94086.

DATA FILE PROCESSING
Storing Data on Cassette
WBB 1/82

A data file is a string of bytes stored on magnetic media independent of any program.

The ATARI 410 Program Recorder stores data on standard audio cassette tapes. The 410 Recorder is called a sequential device because files are distinguished only by their physical location on the tape. Proper positioning of the tape is crucial to insure dependable operation. For this reason, the recommended procedure is to store only one file on each side of a tape. Record the file immediately after the tape leader. The data is transferred over the serial bus to the 410 Recorder at the rate of 600 bits/second or 60 characters/second. The storage capacity of a tape is roughly 1000 characters/minute. Therefore a 60 minute tape would allow storage of about 30,000 bytes on each side.

All data files on tape consist of three sections. There is a 20 second leader of mark tone, followed by any number of data blocks consisting of a pre-record write tone (PRWT), 4 control bytes, 128 data bytes, and a post-record gap (PRG). Finally, there is an end-of-file mark. Each of these sections is audible through the TV speaker during data transmission. The procedure for creating a data file on the 410 Recorder from BASIC is to do an OPEN, a series of outputs (PRINT #N: or PUT #N), and a CLOSE.

The OPEN command establishes a channel from the token file in RAM to the 410 Recorder. There are eight channels in the system numbered 0-7. The OS uses 0,6,7 at various times so you should use 1-5. The correct syntax for the write mode is: OPEN #1,8,0 "C:". When this command is executed, the keyboard buzzes twice to remind you to position the tape and engage 2 keys (PLAY and RECORD) on the 410 Recorder. You need to acknowledge this action by pressing any key on the keyboard (RETURN). The OS then writes 20 seconds of mark tone. It does not automatically shut off the cassette motor. The motor is shut off only after a data block is written on the tape. This is not a problem if all of the data is written out immediately after the file is opened.

The output commands, PRINT #N and PUT #N, transfer data from the token file to the buffer for the tape. When the buffer fills up with 128 bytes, the OS writes a data block to the tape, turns off the motor, and clears the buffer. Two types of I/O can be used to write data to a file, character I/O or record I/O.

Character I/O means that you write data one byte at a time with none of the values interpreted as control characters. The statement PUT #N,X transfers one ASCII byte to the data file.

Record I/O means that you write data one field at a time with the End of Line (EOL) character (ASCII 155) used to delimit the end of each field. The EOL character is automatically generated by the PRINT #N statement. If one field is transferred with each PRINT #N; statement, all fields will be properly separated. The syntax of the PRINT statement should include a semicolon and not a comma. A comma is interpreted as a tab, so 10 blank spaces would be inserted in front of your data. The following statements transfer 10 fields to a data file.

```
DIM NAME$(16)
OPEN #1,8,0,"C:"
FOR I=1 TO 5
PRINT "NAME..."; :INPUT NAME$
PRINT "AGE..."; :INPUT AGE
PRINT #1;NAME$
PRINT #1;AGE
NEXT I
CLOSE #1
```

The CLOSE command writes the current buffer as the last data block and then writes the end of file mark to the tape.

```
1 REM A SIMPLE DATA FILE ON CASSETTE
2 REM PY/JB 2/82
3 REM ****************************
4 REM create a simple file of names on a tape
5 REM read them back and print them
6 REM on the screen, or on a printer
7 REM ************************************
10 DIM BLANK$(128),NAME$(20),ANS$(3)
20 OPEN #1,8,0,"C:":REM open a file for output on the cassette
30 FOR I=0 TO 128:REM create dummy record
32 PUT #1,32:REM of 128 spaces
34 NEXT I:REM to stop motor
35 PRINT #1:REM end dummy record with a carriage return
40 PRINT "NAME";:INPUT NAME$:REM user types in data
50 PRINT #1;NAME$:REM print the data into the cassette file
60 PRINT "MORE DATA (Y/N)";:INPUT ANS$
70 IF ANS$(1,1)="Y" THEN GOTO 40
80 IF ANS$(1,1)="N" THEN GOTO 100
90 GOTO 60
100 CLOSE #1:REM if there is no more data, close the file
110 STOP
150 REM ************************************************
160 REM the tape now contains a list of names.
161 REM to read the names back, rewind the tape,
162 REM and type CONT to continue.
163 REM ************************************************
200 OPEN #1,4,0,"C:":REM open an input file on the cassette
210 TRAP 300:REM in case of error, go close the file
220 INPUT #1,BLANK$:REM get the dummy record and throw it away
230 INPUT #1,NAME$:REM read a name from the tape
240 PRINT NAME$:REM print it on the screen
250 LPRINT NAME$:REM print it on the printer
251 REM (delete line 250 if you don't have a printer)
260 GOTO 230:REM get further names from file
261 REM if there are no more names, an end-of-file error occurs
262 REM and the error-trap goes and closes the file
300 CLOSE #1
```

The procedure for reading the data from a tape data file from BASIC is do an OPEN, a series of inputs (INPUT #N or GET #N), and a CLOSE.

The OPEN command establishes a channel to the 410 Recorder. The correct syntax for the read mode is: OPEN #1,4,0,"C:". When this command is executed, the keyboard buzzes once to remind you to position the tape and engage one key (PLAY) on the 410 Recorder. Acknowledge this action by pressing any key on the keyboard (RETURN). The OS turns on the cassette motor and reads past the mark tone. It does not shut off the tape motor. The motor is shut off only after a data block has been read from the tape. This should never be a problem if you open the tape file only when you are ready to read the data from it.

The data should be read from the file in the same fashion that it was written to the file, record or character I/O. Character I/O reads one byte at a time with none of the values being interpreted as control codes. The GET #1,X transfers one ASCII byte from the data file to the variable X.

Record I/O reads one field at a time with EOL (ASCII 155) used to delimit the end of each field. Many fields can be transferred with each INPUT #N statement. The following statements transfer 10 fields from the data file.

```
DIM NAME$(16)
OPEN #1,4,0,"C:"
FOR I=1 TO 5
INPUT #1,NAME$,AGE
PRINT NAME$,AGE
NEXT I
CLOSE #1
```

There are three ways to read all the data from the file and exit without an error. If you know how many fields were written, you can simply read the same number of fields, as in the example above. If the number of fields changes, you can write a field with a special value at the end of the file and check for this value after each input. If you do not know what is in the file, you can use the TRAP command. When the end-of-file error 136 occurs, the TRAP command will send you to your error routine. The routine should check that location 195 (error status) does contain 136, and then CLOSE the file.

Note 1:
If the PRINT or PUT commands do not immediately follow the OPEN command, the motor stays on and garbage may be written onto the tape, making it unreadable. A solution to this problem is to write a dummy record of 128 blanks immediately after opening the file. The following statements accomplish this.

```
FOR I=1 TO 127 :PUT#1,32:NEXT I :PRINT #1
```

When you then OPEN the file to read it, you must immediately read past this dummy record. An input of any string variable accomplishes this.

```
DIM A$(1) :INPUT #1,A$
```

Note 2:
It is possible to transfer more than one field with each PRINT statement. However, you must write the EOL character after each field.

```
PRINT #1:NAME$;CHR$(155);AGE
  or
DIM CR$(1) :CR$=CHR$(155)
PRINT #1;NAME$;CR$;AGE
```

DATA FILE PROCESSING
AN EXAMPLE OF CASSETTE I/O
WBB/JB 3/82

The following set of programs sets up and maintains a simple mailing list using the 410 Program Recorder. The programs show a method of storing data in data files on the tape. The first program initializes the file by reserving space for each entry. The second updates the information in the file. The third prints out the contents of the file.

The key concepts illustrated are opening a data file with the OPEN statement, and writing to that file using the PRINT #1; statement. In this simple example, only one variable is written at a time, so no extra data separators are necessary.

In order to update a cassette file, the complete file is read into memory, and stored in a long array-string. This process provides a good example of string manipulation, and the long-string method of keeping string-arrays. When the update is complete, the file is written back out to tape.

The following is a procedural list for the three programs which follow:

CASSETTE INIT does only one thing: put blanks on a tape for 100 records. Because it is not possible to add to a data file on cassette tape once the file has been finished, this program gets around that by setting up blank files, which can then be changed to names, addresses, etc. You will only need to use this program once for every 100 entries.

CASSETTE UPDATE is the main program, which adds, changes or deletes records.

CASSETTE PRINT is used only if you want to print out your records on a printer.

1. Type in CASSETTE INIT and CSAVE it on a tape (from now on the tape the programs are saved on will be called the "Program Tape"). Type NEW and then type CASSETTE UPDATE and CSAVE it on the same tape. Decide whether you want CASSETTE PRINT or not. If you do, type NEW and then type this program in and CSAVE it on the same tape.

2. CLOAD CASSETTE INIT into your computer.

3. Remove the Program tape and insert a blank, rewound tape (which will be called the "Data Tape") into your recorder. (You may use the other side of the Program Tape, if you wish.)

4. Type RUN and press RETURN (the recorder will beep twice, and then press RETURN again).

5. CASSETTE INIT will create 100 blank records on the Data Tape.

6. CLOAD CASSETTE UPDATE into your computer. Type RUN. The program will say:

PREPARE TAPE FOR READING,
PRESS 'START' TO CONTINUE...

7. Remove the Program Tape and insert the Data Tape. Rewind it to the beginning.

8. Press RETURN after the recorder beeps twice.

9. The program will then say: READING DATA FOR...(a number) over and over until 100 records have been read. The computer is loading the blanks (in the future it will be names, addresses, etc.) from the tape into its RAM.

10. The program will then say: ITEM (1-100)(0 TO END)...? meaning that you are to type the number of the first record you want to add, OR type 0 if you want to end the program. The first time you do this, you should type 1, since that is the first record you want to add.

11. The program will then say: NAME? and you type in the first name you want to record.

From here on, simply follow the directions. The next time you want to add some names, follow the procedures from item #6. If you want to use CASSETTE PRINT, simply type or load it in if you have saved it, insert the rewound Data Tape, and RUN the program.

```
1 REM CASSETTE INIT
  REM WBB/JB 3/82
3 REM run this program first to reserve file space on the tape
4 REM ************************************************
10 DIM NAME$(100*24),ADDR$(100*24),CITY$(100*16),STATE$(100*2),ZIP$(100*5)
20 DIM PHONE$(100*8),BLANK$(24)
25 REM each field is stored in a long-string variable--
26 REM there is space for 100 records.
30 BLANK$="                        ":REM a string of 24 spaces
40 FOR I=1 TO 100
50 PRINT "INITIALIZING SPACE FOR...";I
60 NAME$(I*24-23,I*24)=BLANK$
70 ADDR$(I*24-23,I*24)=BLANK$
80 CITY$(I*16-15,I*16)=BLANK$
90 STATE$(I*2-1,I*2)=BLANK$
100 ZIP$(I*5-4,I*5)=BLANK$
110 PHONE$(I*8-7,I*8)=BLANK$
120 NEXT I
130 REM all of the records now contain the correct number of blanks --
140 REM the blank records now get saved to tape
150 PRINT :PRINT "PREPARE TAPE FOR WRITING,"
160 PRINT "PRESS 'START' TO CONTINUE..."
165 IF PEEK(53279)<>6 THEN 165:REM wait for start key
170 OPEN #1,8,0,"C:":REM press play and record on cassette unit
180 FOR I=1 TO 100
190 PRINT "WRITING FILE SPACE FOR...";I
200 PRINT #1;NAME$(I*24-23,I*24)
210 PRINT #1;ADDR$(I*24-23,I*24)
220 PRINT #1;CITY$(I*16-15,I*16)
230 PRINT #1;STATE$(I*2-1,I*2)
240 PRINT #1;ZIP$(I*5-4,I*5)
250 PRINT #1;PHONE$(I*8-7,I*8)
260 NEXT I
270 CLOSE #1
280 REM the file space is now reserved on the tape
290 PRINT :PRINT "REWIND THE TAPE"
300 PRINT "** END OF INITIALIZATION **"
310 END
```

```
1 REM CASSETTE UPDATE
2 REM WBB/JB 3/82
3 REM use this program to enter or change information in the file.
4 REM ************************************************************
10 PRINT CHR$(125):REM clear screen
20 PRINT "ENTER OR REPLACE RECORDS"
25 REM set up long-string variables
30 DIM NAME$(100*24),ADDR$(100*24),CITY$(100*16),STATE$(100*2),ZIP$(100*5)
40 DIM PHONE$(100*8),BLANK$(24),X$(24)
50 BLANK$="                        ":REM string of 24 spaces
60 REM ************************************************************
70 REM read in existing file from the tape
80 PRINT "PREPARE TAPE FOR READING,"
90 PRINT "PRESS 'START' TO CONTINUE..."
100 IF PEEK(53279)<>6 THEN 100:REM wait for start key
110 OPEN #1,4,0,"C:":REM press play on cassette unit
120 FOR I=1 TO 100
130 PRINT "READING DATA FOR...";I
140 INPUT #1,X$:NAME$(I*24-23,I*24)=X$
150 INPUT #1,X$:ADDR$(I*24-23,I*24)=X$
160 INPUT #1,X$:CITY$(I*16-15,I*16)=X$
170 INPUT #1,X$:STATE$(I*2-1,I*2)=X$
180 INPUT #1,X$:ZIP$(I*5-4,I*5)=X$
190 INPUT #1,X$:PHONE$(I*8-7,I*8)=X$
200 NEXT I
210 CLOSE #1:REM the string-arrays now hold the data from the saved file
215 REM ************************************************************
220 REM -- update the file --
221 REM ask user which record to look at,
222 REM then call subroutine which displays that record
223 REM and replaces it with new data as entered by user.
230 PRINT :TRAP 240:REM in case of input error, keep trying
240 PRINT "ITEM (1-100)(0 TO END)...";:INPUT I
245 TRAP 40000:REM turn off error trap
250 IF I=0 THEN 300:REM if no more records, go write them out
260 IF I<1 OR I>100 THEN 230:REM bad number, try again
270 GOSUB 1000:REM call subroutine which displays and updates data
280 GOTO 230:REM get next record number
290 REM ************************************************************
295 REM -- write the updated file back out to the tape --
300 PRINT :PRINT "PREPARE TAPE FOR WRITING,":REM rewind or turn over tape
310 PRINT "PRESS 'START' TO CONTINUE..."
320 IF PEEK(53279)<>6 THEN 320:REM wait for start key
330 OPEN #1,8,0,"C:":REM press play and record on cassette unit
340 FOR I=1 TO 100
350 PRINT "WRITING DATA FOR...";I
360 PRINT #1;NAME$(I*24-23,I*24)
370 PRINT #1;ADDR$(I*24-23,I*24)
380 PRINT #1;CITY$(I*16-15,I*16)
390 PRINT #1;STATE$(I*2-1,I*2)
400 PRINT #1;ZIP$(I*5-4,I*5)
410 PRINT #1;PHONE$(I*8-7,I*8)
420 NEXT I
430 CLOSE #1:REM the updated file is now saved on the tape
440 PRINT :PRINT "REWIND THE TAPE"
450 PRINT "  ** END OF PROGRAM **"
460 END
470 REM ************************************************************
990 REM the following subroutine displays the desired record,
```

```
991 REM asks the user whether it should be changed,
992 REM and performs the change if requested.
  00 PRINT :PRINT "RECORD NUMBER...";I
1010 PRINT "NAME",NAME$(I*24-23,I*24)
1020 PRINT "ADDRESS",ADDR$(I*24-23,I*24)
1030 PRINT "CITY",CITY$(I*16-15,I*16)
1040 PRINT "STATE",STATE$(I*2-1,I*2)
1050 PRINT "ZIP",ZIP$(I*5-4,I*5)
1060 PRINT "PHONE",PHONE$(I*8-7,I*8)
1070 PRINT :PRINT "DO YOU WISH TO REPLACE (Y/N)...";:INPUT X$
1080 IF X$<>"Y" THEN RETURN
1085 REM ************************************************************
1090 REM the following section calls a subroutine which
1091 REM gets the new data, and blank-fills if necessary,
1092 REM so that all fields are the proper length.
1093 REM the input field, with the blank-fill, is then
1094 REM put into the string-array in the correct place.
2000 PRINT "NAME",
2005 GOSUB 3000:NAME$(I*24-23,I*24)=X$
2010 PRINT "ADDRESS",
2015 GOSUB 3000:ADDR$(I*24-23,I*24)=X$
2020 PRINT "CITY",
2025 GOSUB 3000:CITY$(I*16-15,I*16)=X$
2030 PRINT "STATE",
2035 GOSUB 3000:STATE$(I*2-1,I*2)=X$
2040 PRINT "ZIP",
2045 GOSUB 3000:ZIP$(I*5-4,I*5)=X$
2050 PRINT "PHONE",
  55 GOSUB 3000:PHONE$(I*8-7,I*8)=X$
2060 RETURN
2070 REM ************************************************************
2080 REM here is the subroutine that gets the new data
2090 REM and blank-fills if necessary
3000 INPUT X$
3010 IF LEN(X$)<24 THEN X$(LEN(X$)+1)=BLANK$:REM concatenate spaces
3020 RETURN
```

```
1 REM CASSETTE PRINT
2 REM WBB/JB 3/82
3 REM this program gets the data file from the tape
4 REM and prints it out on a printer
5 REM ************************************************************
10 PRINT CHR$(125):REM clear screen
20 PRINT "MAKE SURE YOUR PRINTER IS TURNED ON,"
30 PRINT " AND PREPARE THE TAPE FOR READING..."
40 PRINT "PRESS 'START' TO CONTINUE..."
50 IF PEEK(53279)<>6 THEN 50:REM wait for start key
60 POKE 201,2:REM set comma print zone at 2 spaces
70 DIM X$(24):REM only one variable is used
80 OPEN #1,4,0,"C:":REM press play on cassette unit
90 OPEN #2,8,0,"P:":REM open printer for output
98 REM the following section gets each field from the tape file
99 REM and prints it to the printer file
100 FOR I=1 TO 100:REM read and print 100 records
110 PRINT "READING DATA FOR...";I
120 PRINT #2;"READING...";I
130 INPUT #1,X$:PRINT #2;X$
140 INPUT #1,X$:PRINT #2;X$
150 INPUT #1,X$:PRINT #2;X$,
160 INPUT #1,X$:PRINT #2;X$,
170 INPUT #1,X$:PRINT #2;X$,
180 INPUT #1,X$:PRINT #2;X$
190 NEXT I
200 PRINT "REWIND THE TAPE"
210 PRINT " -- END OF PROGRAM --"
220 CLOSE #1
230 CLOSE #2
240 END
```

DATA FILE PROCESSING
Storing Data on Disk
WBB 4/82

The ATARI 810 Disk Drive stores data on 5-1/4" floppy diskettes. A diskette is formatted into 40 concentric tracks, each with 18 sectors giving a total of 720 sectors. Each sector is 128 bytes long (single density). Therefore, the total storage capacity on each diskette is 92,160 bytes.

The 810 Disk Drive uses a boot file to control the power-up initialization procedure. This usually means that the Disk Operating System File Management System (DOS FMS) is loaded into RAM. The DOS FMS is responsibile for allocating the available sectors on a disk as needed for file storage. The boot file uses 3 of the 720 sectors.

DOS maintains a directory of the files that have been created on the disk up to the maximum of 64 files. The files are located in the directory by having a unique name and they are identified by their sequential position in the directory numbered from 0-63. The directory takes up 8 of the 720 sectors.

DOS maintains a bit map of the sectors that have been allocated for file storage. When DOS needs to allocate a sector to a file, it uses the lowest numbered free sector as indicated in the bit map. This is commonly referred to as random access. The bit map takes 1 of the 720 sectors.

DOS uses 3 of the 128 bytes in each sector to identify the file it belongs to and the next sequential sector in the file. Therefore, the file storage space available to the user with DOS 2.0S is 707 sectors x 125 bytes = 88,375 bytes.

OPEN

DOS maintains a pointer for each file opened. The pointer is the location in the file the next I/O command will access. The OPEN command establishes a channel from the token file in RAM to the 810 Disk Drive. There are eight channels in the system numbered 0-7. The OS uses 0,6,7 at various times so you should use 1-5. There are four modes for the OPEN command.

OPEN #1,4,0,"D:FILENAME" opens a file in READ mode. DOS locates FILENAME in the directory and positions the pointer at the first byte of the file. INPUT#1 or GET #1 are the only legal commands.

OPEN #1,8,0,"D:FILENAME" always opens a new file in WRITE mode. DOS first searches the directory for FILENAME and, if it exists, deletes it. DOS then creates FILENAME in the directory, allocates a free sector as the first sector of the file and position the pointer at the first byte of that sector. PRINT#1 or PUT#1 are the only legal commands.

OPEN #1,9,0,"D:FILENAME" opens a file in APPEND mode. DOS first locates FILENAME in the directory. It then allocates a free sector appended to the end of the file and positions the pointer at the first byte of that sector. Mode 9 is write only, so PRINT#1 or PUT#1 are the only legal commands.

OPEN #1,12,0,"D:FILENAME" opens a file in READ/WRITE mode. DOS locates FILENAME in the directory and positions the pointer at the first byte of the file. PRINT#1, PUT#1, INPUT#1, and GET#1 are all legal commands in this mode. However, the output commands write over the values that are currently in the file so the user should be careful to replace the exact number of bytes when updating an existing field. It is not possible to append new data to a file in this mode. Mode 12 can also be thought of as RANDOM ACCESS mode because it is the only mode that supports the commands NOTE#1 and POINT#1.

CLOSE

The CLOSE command frees the channel from the program to the device. It is always a good idea in the interest of good programming technique to CLOSE every file that is opened in a program. However, it is imperative that a file opened in update modes 8 or 9 be closed. If not, it is probable that the bit map or forward pointers will not be updated correctly. The results is an error 164, requiring the disk to be reformatted.

PRINT/PUT

The output commands, PRINT#n and PUT#n, transfer data from the token file to the buffer for the disk file. When the buffer fills up with 125 bytes, DOS writes the buffer to the file, allocates another free sector and clears the buffer. Two types of I/O can be used to write data to a file; character I/O or record I/O.

Character I/O means that you write data one byte at a time with none of the values interpreted as control characters. The statement PUT #n,X transfers one ATASCII byte to the disk file.

Record I/O means that you write data one field at a time with the End of Line (EOL) character (ATASCII 155) used to delimit the end of the fields. The EOL character is automatically generated by the PRINT statement. Therefore, to avoid having to put in your own delimiters, simply transfer each field on a separate PRINT statement. The PRINT statement should include a semicolon and not a comma (PRINT #1;A$). A comma is interpreted as a tab, causing 10 blank spaces to be inserted in front of your data. The size of the field should be limited to less than 119 bytes to avoid using reserved memory space on page 6 of RAM.

INPUT/GET

The input commands, INPUT #1 and GET #1, transfer data from the disk file to the token file. The data should be read from the file in the same fashion that it was written, character or record I/O.

Character I/O reads one byte at a time with none of the values being interpreted as control codes. The statement GET#1,X transfers one ATASCII byte from the disk file to the variable X.

Record I/O reads one field at a time with EOL (ATASCII 155) used to delimit the end of each field (INPUT #1,A$). Many fields can be transferred with each INPUT statement (INPUT #1,A$,B$,C).

There are three ways to read all the data from the file and exit without an error. If you know how many fields were written, you can simply read that number of fields. If the number of fields varies, you can write a special value at the end of the file and check for this value after each input. If you don't know what's in the file, you should use the TRAP command. When the end-of-file error 136 occurs, the TRAP will send you to an error routine. The routine should check that location 195 (error status) does contain a 136, and then CLOSE the file.

NOTE/POINT

NOTE#1,S,B stores the current sector and byte location of the pointer in the variables S and B. Conversely, POINT#1,S,B moves the pointer directly to the sector and byte specified in the variables. These commands are used together in mode 12 to provide the user with random access to the disk file. In general the procedure is NOTE#1,INPUT#1,POINT#1, and PRINT#1. This should only be done with fixed length records. If you update a 4-byte field with a 3-byte field, an extra EOL is added to the file and the number of fields is incorrectly incremented by 1. If you update a 4-byte field with a 5-byte field, the next sequential field in the file is overwritten. Care should be taken to replace the exact number of bytes when updating an existing field.

DATA FILE PROCESSING

An Example of DISK I/O

WBB/JB 3/82

The following set of programs sets up and maintains a simple mailing list using the 810 Disk Drive. The programs provide an example of one method of storing data in data files. The first program sets up the file by adding records. The second updates the information in the file. The third prints out the contents of the file.

The key concepts illustrated are opening a data file with the OPEN statement, and writing to that file using the PRINT#1; statement. In this simple example, only one variable is written at a time, so no extra data separators are necessary,

A temporary file is used to keep the updated information. When the update is complete, the temporary file is renamed, and the old file becomes the temporary file. Some error trapping is provided.

```
1 REM DISKADD
2 REM WBB/JB 4/82
3 REM --use this program to create a file, or to add new records--
4 REM the program creates a temporary file, adds records to it,
5 REM then deletes the permanent file and renames the temporary file
6 REM so that it becomes the permanent file.
7 REM ************************************************************
10 PRINT CHR$(125):REM clear screen
20 PRINT "THIS PROGRAM ADDS RECORDS"
30 PRINT "FOR NEW CUSTOMERS.":PRINT
40 PRINT "INSERT THE PROPER DISKETTE,"
50 PRINT "PRESS 'START' TO CONTINUE..."
60 IF PEEK(53279)<>6 THEN 60:REM wait for start key
65 REM --set up variables and filenames--
70 DIM ID$(9),NAME$(24),ADDR$(24),CITY$(16),STATE$(2),ZIP$(5),PHONE$(12)
80 DIM FILE1$(16),FILE2$(16)
90 FILE1$="D:CUSTOMER.DAT":FILE2$="D:CUSTOMER.TMP"
95 REM --open the files--
100 CLOSE #1:CLOSE #2:REM close any currently open files
110 TRAP 200:OPEN #1,4,0,FILE1$:TRAP 40000:REM check for 'no file found' error
120 OPEN #2,8,0,FILE2$:REM if no error, open write file
130 GOTO 300:REM skip over error routine
180 REM ************************************************************
190 REM --error routine--
195 REM if the file is not on this disk then create one, or replace disk
200 PRINT CHR$(253);"CUSTOMER FILE NOT ON THIS DISK,":PRINT :REM sound buzzer
210 PRINT "PRESS 'START' TO TRY ANOTHER DISK-"
220 PRINT "PRESS 'SELECT' TO CREATE ON THIS DISK-"
230 IF PEEK(53279)=6 THEN 100:REM if start is pressed, try again
240 IF PEEK(53279)<>5 THEN 230:REM check for select
250 CLOSE #1:OPEN #1,8,0,FILE1$:REM if creating, open write file
260 PRINT #1;"ENDOFFILE":REM write file with no records
270 CLOSE #1:GOTO 100:REM now that there is a file, go try again
280 REM ************************************************************
290 REM --transfer existing records to new file--
300 INPUT #1,ID$:REM get record number
305 IF ID$="ENDOFFILE" THEN 400:REM last record, go to add-record routine
310 PRINT #2;ID$:REM transfer number to new file
320 PRINT "TRANSFERRING TO TEMP FILE...";ID$
330 INPUT #1,NAME$:PRINT #2;NAME$
340 INPUT #1,ADDR$:PRINT #2;ADDR$
350 INPUT #1,CITY$:PRINT #2;CITY$
360 INPUT #1,STATE$:PRINT #2;STATE$
370 INPUT #1,ZIP$:PRINT #2;ZIP$
380 INPUT #1,PHONE$:PRINT #2;PHONE$
390 GOTO 300:REM get next record
395 REM ************************************************************
396 REM --add new records to file--
400 PRINT CHR$(125);"SPECIFY RECORD TO ADD";CHR$(29):REM (move cursor down)
410 PRINT "ID NUMBER OR 'END'...":INPUT ID$:IF ID$="END" THEN 600
420 PRINT "NAME...";:INPUT NAME$
425 PRINT "ADDRESS...";:INPUT ADDR$
430 PRINT "CITY...";:INPUT CITY$
435 PRINT "STATE...";:INPUT STATE$
440 PRINT "ZIP...";:INPUT ZIP$
445 PRINT "PHONE...";:INPUT PHONE$
450 PRINT :PRINT "PRESS 'SELECT' TO ADD RECORD..."
460 PRINT "PRESS 'OPTION' TO RE-ENTER..."
470 IF PEEK(53279)=3 THEN 400:REM option is pressed, re-enter record
480 IF PEEK(53279)<>5 THEN 470:REM check for select key
485 REM ************************************************************
```

```
490 REM ************************************************************
  5 REM --write new record to temporary file--
500 PRINT #2;ID$
510 PRINT #2;NAME$
520 PRINT #2;ADDR$
530 PRINT #2;CITY$
540 PRINT #2;STATE$
550 PRINT #2;ZIP$
560 PRINT #2;PHONE$
570 GOTO 400:REM go get more new records
580 REM ************************************************************
590 REM --closing routine--
600 PRINT #2;"ENDOFFILE":REM write end-of-file marker
610 CLOSE #1:CLOSE #2
620 PRINT CHR$(125);"DELETING OLD FILE..."
630 XIO 33,#1,0,0,FILE1$:REM delete old file
640 PRINT :PRINT "RENAMING NEW FILE..."
650 XIO 32,#1,0,0,"D:CUSTOMER.TMP,CUSTOMER.DAT"
660 PRINT :PRINT "--END OF PROGRAM--"
670 END
```

```
1 REM DISK UPDATE
2 REM WBB/JB 3/82
3 REM --use this program to change or delete existing records in the file--
4 REM read records from permanent file, update temporary file,
5 REM then delete old file and rename new one to be new permanent file.
6 REM ************************************************************
10 PRINT CHR$(125):REM clear screen
20 PRINT "THIS PROGRAM CHANGES OR DELETES"
30 PRINT "EXISTING RECORDS IN THE DATA FILE.":PRINT
40 PRINT "PRESS 'START' TO CONTINUE...":PRINT
50 IF PEEK(53279)<>6 THEN 50:REM wait for start key
60 REM set up variables and file names
70 DIM ID$(9),NAME$(24),ADDR$(24),CITY$(16),STATE$(2),ZIP$(5),PHONE$(12)
80 DIM FILE1$(16),FILE2$(16)
90 FILE1$="D:CUSTOMER.DAT":FILE2$="D:CUSTOMER.TMP"
95 REM -- open the files --
100 CLOSE #1:CLOSE #2:REM close any files which are open
110 TRAP 200:OPEN #1,4,0,FILE1$:TRAP 40000:REM trap file-not-found error
120 OPEN #2,8,0,FILE2$:REM if no error, open write file
130 GOTO 300:REM skip error routine
140 REM ************************************************************
150 REM -- error routine --
160 REM if file is not on this diskette, try another one
200 PRINT CHR$(253);"CUSTOMER FILE NOT ON THIS DISK,":PRINT :REM sound buzzer
210 PRINT "PRESS 'START' TO TRY ANOTHER DISK..."
220 IF PEEK(53279)<>6 THEN 220:REM wait for start key
230 GOTO 100:REM try again
240 REM ************************************************************
250 REM -- read a record from the old file --
300 INPUT #1,ID$:REM get record number
305 IF ID$="ENDOFFILE" THEN 600:REM last record, go to closing routine
310 INPUT #1,NAME$,ADDR$,CITY$,STATE$,ZIP$,PHONE$:REM read rest of record
315 REM -- display the record --
320 PRINT CHR$(125);"DATA IN OLD FILE";CHR$(29):REM (move cursor down)
330 PRINT "ID",ID$
335 PRINT "NAME",NAME$
340 PRINT "ADDRESS",ADDR$
345 PRINT "CITY",CITY$
350 PRINT "STATE",STATE$
355 PRINT "ZIP",ZIP$
360 PRINT "PHONE",PHONE$
370 PRINT :PRINT "PRESS 'OPTION' TO MODIFY RECORD-"
375 PRINT "PRESS 'SELECT' TO KEEP RECORD AS IS-"
380 PRINT "PRESS 'START' TO DELETE RECORD-"
390 IF PEEK(53279)=6 THEN 300:REM get another record, don't save this one
391 IF PEEK(53279)=5 THEN 500:REM add this record to the new file
392 IF PEEK(53279)<>3 THEN 390:REM check for option key
395 REM ************************************************************
397 REM -- modify data in record --
400 PRINT :PRINT "ENTER NEW DATA FOR RECORD":PRINT
410 PRINT "ID NUMBER...";:INPUT ID$
420 PRINT "NAME...";:INPUT NAME$
430 PRINT "ADDRESS...";:INPUT ADDR$
440 PRINT "CITY...";:INPUT CITY$
450 PRINT "STATE...";:INPUT STATE$
460 PRINT "ZIP...";:INPUT ZIP$
470 PRINT "PHONE...";:INPUT PHONE$
480 PRINT :PRINT "PRESS 'SELECT' TO ADD RECORD..."
485 PRINT "PRESS 'OPTION' TO RE-ENTER..."
490 IF PEEK(53279)=3 THEN 400:REM re-enter the data
491 IF PEEK(53279)<>5 THEN 490:REM check for select key
495 REM ************************************************************
```

```
495 REM ************************************************************
 6 REM -- add the new record to the temporary file --
500 PRINT #2;ID$
510 PRINT #2;NAME$
520 PRINT #2;ADDR$
530 PRINT #2;CITY$
540 PRINT #2;STATE$
550 PRINT #2;ZIP$
560 PRINT #2;PHONE$
570 GOTO 300:REM go read another record from old file
580 REM ************************************************************
590 REM -- closing routine --
595 REM write end-of-file marker, close files,
596 REM delete old file and rename new one to permanent file.
600 PRINT #2;"ENDOFFILE"
610 CLOSE #1:CLOSE #2
620 PRINT CHR$(125);"DELETING OLD FILE..."
630 XIO 33,#1,0,0,FILE1$:REM delete old file
640 PRINT "RENAMING TEMP TO PERMANENT FILE..."
650 XIO 32,#1,0,0,"D:CUSTOMER.TMP,CUSTOMER.DAT"
660 PRINT :PRINT "-- END OF PROGRAM --"
670 END
```

```
1 REM DISK PRINT
2 REM WB8/JB 3/82
3 REM --use this program to print out the customer list from the file--
6 REM ************************************************************
10 PRINT CHR$(125):REM clear screen
20 PRINT "THIS PROGRAM PRINTS ALL RECORDS"
30 PRINT "FROM THE DATA FILE ON A PRINTER.":PRINT
40 PRINT "PRESS 'START' TO CONTINUE...":PRINT
50 IF PEEK(53279)<>6 THEN 50:REM wait for start key
60 REM set up variables and file names
70 DIM ID$(9),NAME$(24),ADDR$(24),CITY$(16),STATE$(2),ZIP$(5),PHONE$(12)
80 DIM FILE1$(16),FILE2$(16)
90 FILE1$="D:CUSTOMER.DAT":FILE2$="P:"
95 REM -- open the files --
100 CLOSE #1:CLOSE #2:REM close any files which are open
110 TRAP 200:OPEN #1,4,0,FILE1$:TRAP 40000:REM trap file-not-found error
120 TRAP 250:OPEN #2,8,0,FILE2$:TRAP 40000:REM trap printer-not-ready error
130 GOTO 300:REM skip error routine
140 REM ************************************************************
150 REM -- error routine, file not found --
200 PRINT CHR$(253);"CUSTOMER FILE NOT ON THIS DISK,":PRINT :REM sound buzzer
210 PRINT "PRESS 'START' TO TRY ANOTHER DISK..."
220 IF PEEK(53279)<>6 THEN 220:REM wait for start key
230 GOTO 100:REM try again
235 REM ************************************************************
240 REM --error routine, printer not ready--
250 PRINT CHR$(253);"PRINTER NOT READY,":REM sound buzzer
260 PRINT "PRESS 'START' TO TRY AGAIN...":PRINT
270 IF PEEK(53279)<>6 THEN 270:REM wait for start key
280 GOTO 100:REM try again
285 REM ************************************************************
290 REM -- read a record from the disk file --
300 INPUT #1,ID$:REM get record number
305 IF ID$="ENDOFFILE" THEN 500:REM last record, go to closing routine
310 INPUT #1,NAME$,ADDR$,CITY$,STATE$,ZIP$,PHONE$:REM read rest of record
315 REM -- display the record --
320 PRINT CHR$(125);"DATA IN FILE";CHR$(29):REM (move cursor down)
330 PRINT "ID",ID$
335 PRINT "NAME",NAME$
340 PRINT "ADDRESS",ADDR$
345 PRINT "CITY",CITY$
350 PRINT "STATE",STATE$
355 PRINT "ZIP",ZIP$
360 PRINT "PHONE",PHONE$
370 PRINT "PRESS 'SELECT' TO PRINT RECORD..."
380 PRINT "PRESS 'START' TO READ NEXT RECORD..."
390 IF PEEK(53279)=6 THEN 300:REM get another record
391 IF PEEK(53279)<>5 THEN 390:REM check for select key
395 REM ************************************************************
397 REM -- print out record on printer --
400 PRINT #2;ID$
410 PRINT #2;NAME$
420 PRINT #2;ADDR$
430 PRINT #2;CITY$
440 PRINT #2;STATE$
450 PRINT #2;ZIP$
460 PRINT #2;PHONE$
470 GOTO 390:REM go wait for ok to read new record
480 REM ************************************************************
490 REM -- closing routine --
500 CLOSE #1:CLOSE #2
510 PRINT CHR$(125):REM clear screen
520 PRINT "-- END OF PROGRAM --"
530 END
```

RANDOM ACCESS with DOS 2.OS
WB 10/82

A. CONCEPT

There are two methods of accessing a data file stored on the 810 Disk Drive. Sequential access is characterized by DIRECT access to the data file, and is accomplished by searching through a data file from beginning to end, looking for a desired record. Random access is characterized by INDIRECT access to the data file by way of an index file, looking for a key which points to the desired record's location in the data file. Simply put, random access is the ability to read a particular record in a file without having to first read every previous record in the file.

B. ADVANTAGES

1. Speed: Because an index file is much smaller than the data file, and usually resides in RAM, rather than on the storage device, the time to locate a record using random access is a mere fraction of sequential access time.

C. DISADVANTAGES

1. Programming techniques are more difficult.

2. Disk storage space may be required for the index file.

3. Using the DOS COPY command requires that the index file be rebuilt.

D. METHODS

Random access is achieved by creating an INDEX for a data file. An index entry consists of a key and pointer for each record in the data file. Entries are kept in ascending order of the key's ATASCII value. The key is some small part of the whole record such as LAST NAME or ACCOUNT NUMBER. The pointer is the actual storage location on the disk (sector and byte number) of the record.

1. Index file created and maintained in the application program:

At the beginning of the program, build the index file by reading each record in the file and saving the key and pointer in a program variable. The advantage with this method is the ability to easily define a new key. The disadvantage is the time (up to several minutes) required to read through the file and create the index.

2. Index file maintained on the disk:

Store the index file as a unique file on the disk which can be read into any program that needs to randomly access the data file. The advantage is that only an insignificant delay is required at the beginning of the program to load the index file. The disadvantage is the requirement for more complicated programming to update the index file whenever the data file is updated.

E. ATARI 8K BASIC

Random access capability is provided in the ATARI 8K BASIC programming language with the NOTE and POINT commands. The NOTE command identifies the location on the disk (sector and byte number) where the next I/O operation will occur. The POINT command is used to position the pointer to a desired location on the disk where the next I/O operation should occur.

The following rules should be observed when programming random access applications in ATARI 8K BASIC.

1. The data file must be OPEN in mode 12 (OPEN #1,12,0,"D:filename").

2. Additional data cannot be appended to the end of a file opened in mode 12.

3. You can only do random access (POINT) to a sector that is allocated to the file.

F. ATARI MICROSOFT BASIC

Random access capability is provided in the ATARI Microsoft BASIC programming language with the NOTE and AT commands. The NOTE command in Microsoft is exactly the same as in 8K BASIC. The AT (sector, byte) command can be added to either an INPUT or PRINT command, to cause the I/O operation to occur at a desired location on the disk.

The following rules should be observed when programming random access applications in ATARI Microsoft BASIC.

1. The data file must be opened in UPDATE mode (OPEN #1,"D:filename" UPDATE).

2. Additional data cannot be appended to the end of a file opened in UPDATE mode.

3. You can only do random access (AT) to a sector that is allocated to the file.

G. EXAMPLE PROGRAMS

The programs on the following pages provide simple examples of random access to a data file in either Atari 8K BASIC or ATARI Microsoft BASIC. The first two programs are constructed as in Method 1. under D. (on the previous page). Each record consists of the first name, last name and phone number. The first two bytes of the file are used to store the number of records in the file in lobyte/hibyte format. The second program provides random access inquiry into the data file. The program variable used for the index file is INDEX$. The key field is the last name. The program searches sequentially through the index looking for a match to the desired last name, and then points to the actual data record.

This is not the ultimate random access method, but it does provide substantial speed improvements over sequential access directly to the data file. The inquiry program could be improved by sorting INDEX$ and using binary search techniques to locate a desired key.

```
10 REM   RANDOM1
20 REM This program adds records to the file D1:TESTDATA.
   REM Each record has the fields FIRST NAME, LAST NAME, PHONE #
   REM The first 2 bytes in the file contain the # of records in
23 REM the file in the format lobyte, hibyte.
100 REM   SETUP
110 GRAPHICS 0:POKE 82,0
120 DIM FILE$(20),FIELD1$(20),FIELD2$(30),FIELD3$(12)
130 FILE$="D1:TESTDATA"
140 RECS=0
200 REM   OPEN FILE
210 TRAP 230
220 CLOSE #1:OPEN #1,4,0,FILE$:TRAP 40000
221 GET #1,LO:GET #1,HI:RECS=LO+HI*256
222 CLOSE #1:OPEN #1,9,0,FILE$
223 GOTO 300
230 CLOSE #1:OPEN #1,8,0,FILE$
231 PUT #1,0:PUT #1,0:REM 0 RECORDS
300 REM   ADD RECORDS
310 ? CHR$(125);"PRESS SELECT TO END THE PROGRAM..."
320 ? "PRESS START TO ADD ANOTHER RECORD..."
330 IF PEEK(53279)=5 THEN 400
331 IF PEEK(53279)<>6 THEN 330
340 ? CHR$(125);"FIRST NAME...";:INPUT FIELD1$
341 ? " LAST NAME...";:INPUT FIELD2$
342 ? "   PHONE #...";:INPUT FIELD3$
350 ? #1;FIELD1$
351 ? #1;FIELD2$
352 ? #1;FIELD3$
353 RECS=RECS+1
360 GOTO 300
400 REM   END
410 CLOSE #1
420 HI=INT(RECS/256):LO=RECS-HI*256
430 OPEN #1,12,0,FILE$
431 PUT #1,LO:PUT #1,HI
432 CLOSE #1
440 ? CHR$(125);"END OF PROGRAM"
450 END
```

```
10 REM   RANDOM2
20 REM This program creates an index file in RAM (INDEX$)
21 REM and allows random access to records by last name.
100 REM   SETUP
110 GRAPHICS 0:POKE 82,0
120 DIM FILE$(20),FIELD1$(20),FIELD2$(30),FIELD3$(12)
130 FILE$="D1:TESTDATA"
200 REM   OPEN FILE
220 CLOSE #1:OPEN #1,12,0,FILE$
230 GET #1,LO:GET #1,HI:RECS=LO+HI*256
240 IF RECS=0 THEN ? "NO RECORDS IN THE FILE!":GOTO 900
250 ? "STANDBY WHILE THE INDEX IS BUILT..."
300 REM   BUILD INDEX
310 DIM INDEX$(RECS*33):REM 30 FOR LAST NAME + 2 FOR SECTOR + 1 FOR BYTE
311 DIM PAD$(33):PAD$="                                ":REM used to blank fill
312 DIM DUM$(33):REM DUMMY VARIABLE
320 FOR I=1 TO RECS
330 NOTE #1,SECTOR,BYTE
331 INPUT #1,FIELD1$,FIELD2$,FIELD3$
332 HI=INT(SECTOR/256):LO=SECTOR-HI*256
340 DUM$=FIELD2$:DUM$(LEN(DUM$)+1)=PAD$
341 DUM$(31)=CHR$(LO):DUM$(32)=CHR$(HI):DUM$(33)=CHR$(BYTE)
350 INDEX$(LEN(INDEX$)+1)=DUM$
351 NEXT I
400 REM   INDEX SEARCH
410 ? CHR$(125);"ENTER LAST NAME OR 'END'...";:INPUT DUM$:IF DUM$="END" THEN 900
411 DUM$(LEN(DUM$)+1)=PAD$:FIELD2$=DUM$
420 FOR I=1 TO RECS
421 IF INDEX$(I*33-32,I*33-3)=FIELD2$ THEN 500
422 NEXT I
430 ? "MATCH NOT FOUND!":FOR I=1 TO 250:NEXT I:GOTO 400
500 REM   RANDOM ACCESS TO RECORD
510 POP
520 LO=ASC(INDEX$(I*33-2)):HI=ASC(INDEX$(I*33-1)):BYTE=ASC(INDEX$(I*33))
521 SECTOR=LO+HI*256
530 POINT #1,SECTOR,BYTE
531 INPUT #1,FIELD1$,FIELD2$,FIELD3$
532 ? "FIRST NAME:";FIELD1$
533 ? " LAST NAME:";FIELD2$
534 ? "   PHONE #:";FIELD3$
540 ? :? "PRESS START WHEN DONE..."
541 IF PEEK(53279)=6 THEN 400
542 GOTO 541
900 REM   END
910 CLOSE #1
920 ? CHR$(125);"END OF PROGRAM."
930 END
```

```
10 REM   RANDOM1.MSB
20 REM This program adds records to the file D1:TESTDATA.
21 REM Each record has the fields FIRST NAME, LAST NAME, PHONE #
22 REM The first 2 bytes in the file contain the number of records in
23 REM the file in the format lobyte/hibyte.
100 REM   SETUP
110 GRAPHICS 0:POKE 82,0
140 RECS=0
200 REM   OPEN FILE
210 ON ERROR 230
220 CLOSE #1:OPEN #1,"D:TESTDATA" INPUT:ON ERROR 0
221 GET #1,LO:GET #1,HI:RECS=LO+HI*256
222 CLOSE #1:OPEN #1,"D:TESTDATA" APPEND
223 GOTO 300
230 CLOSE #1:OPEN #1,"D:TESTDATA" OUPUT
231 PUT #1,0:PUT #1,0:REM 0 RECORDS
300 REM   ADD RECORDS
310 PRINT  CHR$(125);"PRESS SELECT TO END THE PROGRAM..."
320 PRINT  "PRESS START TO ADD ANOTHER RECORD..."
330 IF PEEK(53279)=5 THEN 400
331 IF PEEK(53279)<>6 THEN 330
340 PRINT  CHR$(125);:INPUT "FIRST NAME...",FIELD1$
341 INPUT  " LAST NAME...",FIELD2$
342 INPUT  "   PHONE #...",FIELD3$
350 PRINT  #1,FIELD1$
351 PRINT  #1,FIELD2$
352 PRINT  #1,FIELD3$
353 RECS=RECS+1
360 GOTO 300
400 REM   END
410 CLOSE #1
420 HI=INT(RECS/256):LO=RECS-HI*256
430 OPEN #1,"D:TESTDATA" UPDATE
431 PUT #1,LO:PUT #1,HI
432 CLOSE #1
440 PRINT  CHR$(125);"END OF PROGRAM"
450 END
```

```
10 REM   RANDOM2.MSE
20 REM This program creates an index file in RAM (INDEX$) and allows
21 REM random access to records by last name.
100 REM   SETUP
110 GRAPHICS 0:POKE 82,0
200 REM   OPEN FILE
220 CLOSE #1:OPEN #1,"D:TESTDATA" UPDATE
230 GET #1,LO:GET #1,HI:RECS=LO+HI*256
240 IF RECS=0 THEN PRINT  "NO RECORDS IN THE FILE!":GOTO 900
250 PRINT  "STANDBY WHILE THE INDEX IS BUILT..."
300 REM   BUILD INDEX
310 OPTION BASE 1: DIM INDEX$(RECS,3):REM KEY/SECTOR/BYTE
320 FOR I=1 TO RECS
330 NOTE #1,SECTOR,BYTE
331 INPUT #1,FIELD1$,FIELD2$,FIELD3$
340 INDEX$(I,1)=FIELD2$
341 INDEX$(I,2)=STR$(SECTOR)
342 INDEX$(I,3)=STR$(BYTE)
350 NEXT I
400 REM   INDEX SEARCH
410 PRINT CHR$(125);
411 INPUT "ENTER LAST NAME OR 'END'...";FIELD2$:IF FIELD2$="END" THEN 900
420 FOR I=1 TO RECS
421 IF INDEX$(I,1)=FIELD2$ THEN 500
422 NEXT I
430 PRINT  "MATCH NOT FOUND!":PRINT:PRINT  "PRESS START TO CONTINUE..."
431 WAIT 53279,7,6
432 GOTO 400
500 REM   RANDOM ACCESS TO RECORD
510 SECTOR=VAL(INDEX$(I,2)):BYTE=VAL(INDEX$(I,3))
530 INPUT #1,AT(SECTOR,BYTE) FIELD1$,FIELD2$,FIELD3$
532 PRINT  "FIRST NAME:";FIELD1$
533 PRINT  " LAST NAME:";FIELD2$
534 PRINT  "   PHONE #:";FIELD3$
540 PRINT  :PRINT  "PRESS START WHEN DONE..."
541 WAIT 53279,7,6
542 GOTO 400
900 REM   END
910 CLOSE #1
920 PRINT  CHR$(125);"END OF PROGRAM."
930 END
```

The programs following this page set up random access to data files in which the index "map" is a separate file from the data file proper (lines 70-80). The relative advantages to this procedure are described under section D.2. earlier.

The following is a sectional description of the program, which describes the different sections so that you may easily modify and adapt portions to your own needs.

10	Sets the total number of records.
12	Sets the length of each record.
14-16	Set the length of the name and address. The total length must not be less than the name + the address.
25	Clears out the strings.
30	An error will occur if the index file has never been entered.
40-50	Open both files to make sure they exist.
60	Re-opens the data file for UPDATE.
70-80	Get the index to the data file.
85	Converts the hi-lo bytes to one number.
90	Stores the numbers in the subscripted variables.
1000-2000	Demonstrative portion of program.
1100	Sets R equal to the record number to use.
1110-1170	Printing to the data file
1140	GOSUB string pad subroutine
1150-1170	POINT to the correct location and print the name and address.
1200-1220	Inputs from the data file.
1200-1220	POINT to the correct location and inputs the name and address.
3000-3040	Subroutine for padding string with spaces so that the sector pointers on the diskette do not get scrambled.
4000-4100	Establish an index file for the data file.
4060	NOTEs the location about to be written to.
4080	PRINTs the name and the address to the data file.
4085	Converts the SECT to hi-lo byte numbers.
4090	Puts the pointers into the index file.
4100	Does next record and re-runs the program when done.

```
1 REM Random Access - Atari BASIC Version
2 REM by Kent Smith
10 RCDS=100
12 LNGTH=40
14 NM=20
16 AD=20
19 PRINT CHR$(125);
20 DIM S(RCDS) , C(RCDS) , SPACE$(LNGTH) , NAME$(NM) , ADDR$(AD)
25 SPACE$ = " " : SPACE$(LNGTH)= " " : SPACE$(2) = SPACE$ : NAME$ = SPACE$(1,NM) :
ADDR$ = SPACE$(1,AD)
30 TRAP 4000
35 PRINT CHR$(125);:POSITION 7,3:PRINT "READING INDEX FILE"
40 OPEN #2,4,0,"D:INDEX.DAT"
50 OPEN #1,4,0,"D:FILE.DAT":CLOSE #1:TRAP 40000
60 OPEN #1,12,0,"D:FILE.DAT"
70 FOR A=1 TO RCDS
75 POSITION 5,5:PRINT A
80 GET #2,SECTH:GET #2,SECTL:GET #2,CHAR
85 SECT = SECTH * 256 + SECTL
90 S(A) = SECT : C(A) = CHAR:NEXT A:CLOSE #2:GOTO 1000
1000 REM Enter or retrieve data here
1020 PRINT CHR$(125);
1100 R = 45
1110 NAME$ = "JOYCE PERRY"
1130 ADDR$ = "1234 ANY STREET"
1140 GOSUB 3000
1150 POINT #1,S(R),C(R)
1160 PRINT #1;NAME$
1170 PRINT #1;ADDR$
1180 NAME$ = SPACE$ : ADDR$ = SPACE$
1200 POINT #1,S(R),C(R)
1210 INPUT #1,NAME$
1220 INPUT #1,ADDR$
1230 PRINT NAME$
1240 PRINT ADDR$
2000 CLOSE #1:CLOSE #2
2010 END
3000 REM Subroutine for NAME$ & ADDR$
3010 NAME$(LEN(NAME$) + 1) = SPACE$
3030 ADDR$(LEN(ADDR$) + 1) = SPACE$
3040 RETURN
4000 TRAP 40000:PRINT CHR$(125);:POSITION 7,3:PRINT "CREATING NEW INDEX FILE"
4010 CLOSE #1:CLOSE #2
4020 OPEN #2,8,0,"D:INDEX.DAT"
4030 OPEN #1,8,0,"D:FILE.DAT"
4040 FOR A=1 TO RCDS
4050 POSITION 5,5:PRINT A
4060 NOTE #1,SECT,CHAR
4070 S(A) = SECT : C(A) = CHAR
4080 PRINT #1;NAME$:PRINT #1;ADDR$
4085 SECTH = INT(SECT/256) : SECTL = INT(SECT - SECTH * 256)
4090 PUT #2,SECTH:PUT #2,SECTL:PUT #2,CHAR
4100 NEXT A:CLOSE #1:CLOSE #2:GOTO 30
```

10	Sets the total number of records.
12	Sets the length of each record.
14-16	Set the length of the name and address. The total length must not be less than the name + the address.
25	Clears out the strings.
30	An error will occur if the index file has never been entered.
40-50	Open both files to make sure they exist.
60	Re-opens the data file for UPDATE.
70-80	Get the index to the data file.
85	Converts the hi-lo bytes to one number.
90	Stores the numbers in the subscripted variables.
1000-2000	Demonstrative portion of program.
1100	Sets R equal to the record number to use.
1110-1170	Printing to the data file.
1140	GOSUB string pad subroutine.
1150-1170	Point to the correct location with the AT command and print the name and address.
1200-1240	Inputing from the data file.
1200-1220	Point to the correct location with the AT command and print the name and address.
3000-3040	Subroutine for padding string with spaces so that the sector pointers on the diskette do not get scrambled.
4000-4100	Establish an index file for the data file.
4060	NOTEs the location about to be written to.
4080	PRINTs the name and the address to the data file.
4085	Converts the SECT to hi-lo byte numbers.
4090	Puts the pointers into the index file
4100	Does next record and re-runs the program when done.

```
1 REM Random Access - Microsoft BASIC Version
2 REM by Kent Smith
10 RCDS=100
12 LNGTH=40
14 NM=20
16 AD=20
19 PRINT  CHR$(125);
20 DIM S(RCDS),C(RCDS)
25 NME$ = STRING$(NM,32) : ADDR$ = STRING$(AD,32)
30 ON ERROR 4000
35 PRINT CHR$(125);:PRINT AT(7,3)"READING INDEX FILE"
40 OPEN #2,"D:INDEX.DAT" INPUT
50 OPEN #1,"D:FILE.DAT" INPUT:CLOSE #1:ON ERROR 0
60 OPEN #1,"D:FILE.DAT" UPDATE
70 FOR A=1 TO RCDS
75 PRINT AT(5,5)A
80 GET #2,SECTH:GET #2,SECTL:GET #2,CHAR
85 SECT = SECTH * 256 + SECTL
90 S(A) = SECT : C(A) = CHAR :NEXT A:CLOSE #2:GOTO 1000
1000 REM Enter or retrieve data here
1020 PRINT CHR$(125);
1100 R = 45
1110 NME$ = "JOYCE PERRY"
1130 ADDR$ = "1234 ANY STREET"
1140 GOSUB 3000
1150 PRINT #1,AT(S(R),C(R))NME$
1170 PRINT #1,ADDR$
1180 NME$ = STRING$(NM,32) : ADDR$ = STRING$(AD,32)
1200 INPUT #1,AT(S(R),C(R))NME$
1220 INPUT #1,ADDR$
1230 PRINT NME$
1240 PRINT ADDR$
2000 CLOSE #1:CLOSE #2
2010 END
3000 REM Subroutine for NME$ & ADDR$
3010 NME$ = NME$ + STRING$(NM - LEN(NME$),32)
3030 ADDR$ = ADDR$ + STRING$(AD - LEN(ADDR$),32)
3040 RETURN
4000 PRINT CHR$(125);:PRINT AT(7,3)"CREATING NEW INDEX FILE"
4010 CLOSE #1:CLOSE #2
4020 OPEN #2,"D:INDEX.DAT" OUTPUT
4030 OPEN #1,"D:FILE.DAT" OUTPUT
4040 FOR A=1 TO RCDS
4050 PRINT AT(5,5)A
4060 NOTE #1,SECT,CHAR
4070 S(A) = SECT : C(A) = CHAR
4080 PRINT #1,NME$:PRINT #1,ADDR$
4085 SECTH = INT(SECT / 256) : SECTL = INT(SECT - SECTH * 256)
4090 PUT #2,SECTH:PUT #2,SECTL:PUT #2,CHAR
4100 NEXT A:CLOSE #1:CLOSE #2:RUN
```

Many of you have experienced entering a program which asks you to input data which the program then manipulates in some way (such as Check Balancer in the Basic Reference Manual). You then save the program to cassette or diskette, only to find that on reloading the program, the items previously entered were no longer there. This is because all variables are cleared when you transfer a program to memory. To save the entered items, or variables, you must set up special files, called data files, following the procedures illustrated in the following pages.

Enclosed is a collection of programs which serve as instructions for creating data files on cassette or diskette. A program which uses data files has the same elements as an office. The program which decides how to categorize and manipulate the data is like the secretary or other human agent who sets up various projects. The data files are like the file cabinet in which the necessary material for these projects is kept.

A word to the wise regarding cassette files: the main difference between cassette and diskette storage, is that files on diskette may be accessed by name. This means you may keep a large number of files on a disk and the computer will do the searching for a specific file when you desire to load it. Several differently named disk files may be set up within a program, accessed, changed and resaved, due to this freedom. Cassette files on the other hand, are much more limiting, as it is up to you to know where a file is according to your setting of the mechanical counter. Therefore, they cannot be added to except by the device of setting up blank files which you can later replace with data.

A second difference between cassette and disk files is the nature of the medium used. A disk is read by the disk heads floating over the surface searching for the correct sectors. A cassette tape, on the other hand, moves back and forth mechanically, constantly rubbing against the heads. This often leads to stretching and marring of the tape, generating the common "tape loading errors" 138, 140 and 143. Tape is, simply stated, not an ideal medium for data files.

BIBLIOGRAPHY (CASSETTE FILES)

Your Atari Computer, by Lon Poole, published by Osborne/McGraw-Hill. Re Chapter 5.

Antic Magazine, September 1983, "Cardfile For Cassettes," by Vern Mastel.

Microcomputing Magazine, October 1982, "Trouble Free Cassette Use On The Atari," by Marty Carmichael.

Softside Magazine, September through December 1980, "Developing Data Base (Parts 1-4)," by Pelczarski and Bouchard.

Compute! Magazine, May 1980, "Atari Tape Data Files," by Al Baker.