Chapter 4

FILES

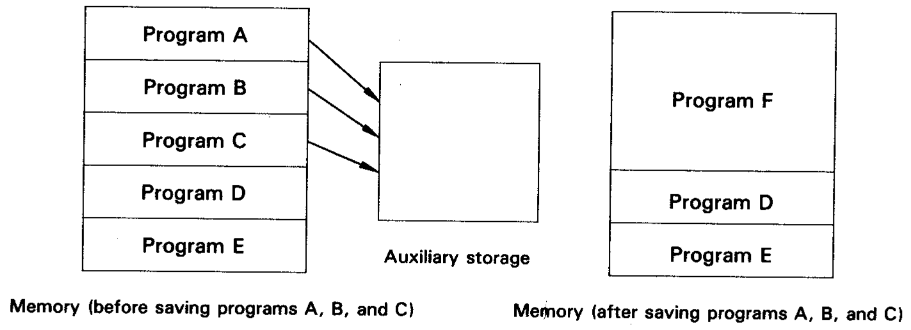
4.1 Program Files

You can load up to five programs in the PX-4 memory by dividing the user area into five.

If there are five programs already existing in memory, or if your new program is too large to fit in the space allotted to it, you may have to delete one of the existing programs to make room for the new one. Before doing this however, you must save on auxiliary storage (such as the RAM disk or microcassette tape.) any programs you want to use again.

The PX-4 supports auxiliary storage methods such as RAM disk, microcassette tape, RAM cartridges, floppy disks, and external cassette tape.

As shown below, you can create program F in the area occupied programs A, B, and C by saving these programs on auxiliary storage then deleting them with the NEW command.



To reexecute Program A, load it back into memory after saving one of the other programs.

Programs saved on auxiliary storage are called program files and treated as files with BASIC. A file is labelled by a file specification consisting of a drive name, a filename, and a file extension. A program file is also identified by its drive name, filename, and file extension. BASIC will use the logged-in drive for CP/M if the drive name is omitted, and automatically appends .BAS if the file extension is omitted. Here is a review of the commands and statements used for program file operations.

(i) SAVE "[<drive name>:]<filename>[.<file extension>]"[, |A P|]

The SAVE command writes a program existing in the program area to the auxiliary storage device specified by <drive name>. The specified filename and file extension must then be assigned to the saved program file.

If the A option is included, the program is saved in the ASCII format; otherwise it is saved in binary format.

The ASCII format allows a program to be saved in the same form as it is displayed on the screen. In the binary format, the program is converted into an intermediate language and written onto a disk in a compressed form. This method of program storage uses less memory space; however, any programs which are to be merged must be saved in the ASCII format.

If the P option is specified, the program is saved in a protected form. This means that program file cannot be edited or listed, and once the option is specified, the protection cannot be cancelled.

(ii) LOAD "[<drive name>:]<filename> [.<file extension>]"[,R]

The LOAD command loads the program from the auxiliary storage device specified by <drive name> to the current program area. Specify the filename and file extension of the program file to be loaded. Add the R option if the program is to be executed as soon as it is loaded. All open files are kept open if this option is specified. Thus, a LOAD command issued with the R option may be used to chain the execution of programs which use the same data file. Executing this command without the R option closes all data files which are currently open.

Example

```
SAVE "A:TEST.BAS",P
Ok
LOAD "A:TEST.BAS"
Ok
LIST
FC Error
Ok
```

(iii) RUN "[<drive name>:]<filename> [.<file extension>]"[,R]

If the file specification is omitted, the RUN command executes the program in the current program area. Using the file specification causes the named program to be loaded into the current program area and to be executed. This operation deletes the contents of any existing loaded program. If the R option is included, all open data files are kept open; otherwise they are closed.

Example

```
5 ' RUN2
10 A$="RUN COMMAND TEST"
20 PRINT#1,A$
30 PRINT A$
40 CLOSE

SAVE "A:RUN2"
Ok
NEW
Ok

5 ' RUN1
10 OPEN "O",#1,"TEST.DAT"
20 RUN "RUN2.BAS",R

RUN
RUN COMMAND TEST
Ok
```

(iv) MERGE "[<drive name>]<filename> [.<file extension>]"

The MERGE command merges a program in the current program area with a program on the auxiliary storage device specified by <drive name>. Once <drive name> is specified, add the filename and file extension of the program file to the file specification. The program files to be merged must have been saved in the ASCII format.

If the two programs use some of same line numbers, the lines of the program in memory (the current program area) are replaced by those from the program on the auxiliary storage device.

BASIC always returns to the command level after executing the MERGE command.

Example

```
5 ' MERGE1
10 DIM B(10)
20 FOR I=1 TO 10
30   READ A : PRINT A;
40   B(I)=A
50 NEXT : PRINT
60 DATA 72,61,43,100,86,37,56,55,65,45

SAVE "A:MERGE1",A
Ok
```

```
NEW
Ok

100 ' merge2
110 S=0
120 FOR I=1 TO 10
130    S=S+B(I)
140 NEXT
150 PRINT "SUMMING = ";S
160 PRINT "AVERAGE = ";S/10
170 END

MERGE "A:MERGE1"
Ok
RUN

 72  61  43  100  86  37  56  55  65  45
SUMMING =  620
AVERAGE =  62
Ok

LIST
5 'MERGE1
10 DIM B(10)
20 FOR I=1 TO 10
30    READ A : PRINT A;
40    B(I)=A
50 NEXT : PRINT
60 DATA 72,61,43,100,86,37,56,55,65,45
100 ' merge2
110 S=0
120 FOR I=1 TO 10
130    S=S+B(I)
140 NEXT
150 PRINT "SUMMING = ";S
160 PRINT "AVERAGE = ";S/10
170 END
```

(v) KILL "[<drive name>:]<filename> [.<file extension>]"

The KILL command deletes a file from the auxiliary storage device named in the file specification. Care must be taken when using this command because it will delete any file having the specified name regardless of whether it is a system, program, or data file.

(vi) NAME <old file specification> AS <new file specification>

The NAME command changes the name of a file on the auxiliary storage device. This command, as with the KILL command, can be used for both program and data files.

Specify the existing filename for <old file specification> and the new filename to be assigned for <new file specification>. Both filenames must designate the same drive.

Example

```
FILES
STOCKLST.BAS  ACCOUNT .BAS  DEMO     .BAS
STOCK   .DAT  CHART   .DAT  TEST1    .DAT
ITEM    .DAT  SALES1  .BAS  GRAPH    .BAS
Ok
```

```
KILL "DEMO.BAS"
Ok
FILES
STOCKLST.BAS  ACCOUNT .BAS  STOCK    .DAT
CHART   .DAT  TEST1   .DAT  ITEM     .DAT
SALES1  .BAS  GRAPH   .BAS
Ok
```

```
NAME "SALES1.BAS" AS "RESULT.BAS"
Ok
FILES
STOCKLST.BAS  ACCOUNT .BAS  STOCK    .DAT
CHART   .DAT  TEST1   .DAT  ITEM     .DAT
RESULT  .BAS  GRAPH   .BAS
Ok
```

"FE Error" will be displayed if the new filename specified already exists.

4.2 Sequential Date Files

The sequential data file contains data that can only be read in the sequence that it is written. You cannot update part of a sequential data file once it has been created. However, sequential files are easy to create, requiring no special consideration about the length or structure of the data to be entered.

4.2.1 Creating sequential data files

The steps involved in creating a sequential data file are as follows:

(1) Execute an OPEN statement using the "O" mode. The file number specified in this statement will be assigned to the output file.

Example: `OPEN "O",#1,"A:OUTDT.DAT"`

This example creates a data file named "OUTDT.DAT" on the RAM disk and assigns file number 1 to that file.

(2) Write data into the file using the PRINT# or WRITE# statement.

Example:
```
PRINT #1,A$;",";B$;",";C$
WRITE #1,A$,B$,C$
```

Both examples write the contents of A$, B$, and C$ into the file.

Contents of A$	Contents of B$	Contents of C$

(3) End the write operation with a CLOSE statement when all the data has been written.

Example: `CLOSE #1`

This statement closes file number 1.

4.2.2 Reading sequential data files

The steps for reading a sequential data file are as follows:

(1) Execute an OPEN statement in the "I" mode. The file number specified in this statement will be assigned as an input file, in a similar manner to the output file.

Example: `OPEN "I",#1,"A:OUTDT.DAT"`

This statement declares that BASIC will find the file named "OUTDT.DAT" and start reading the file.

(2) Read data from the file into variables in memory by executing either an INPUT # or LINE INPUT # statement.

Example:
```
INPUT #1,A$,B$,C$
LINE INPUT #1,A$
```

Both examples assign the data items in the file to the variables A$, B$, and C$ in the same order as they are written.

(3) End the read operation by executing a CLOSE statement after all the data has been read.

Example: `CLOSE #1`

This example closes file number 1.

4.2.3 Sample programs

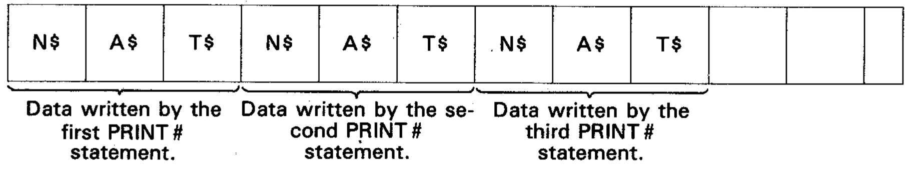
```
5 'SEQUENTIAL FILE1
10 OPEN "O",#1,"A:ADRSBOOK.DAT"
20 INPUT "NAME     :";N$
30 IF N$="*" THEN 80
40 INPUT "ADDRESS :";A$
50 INPUT "PHONE    :";P$
60 PRINT #1,N$;",";A$;",";P$
70 PRINT : GOTO 20
80 CLOSE
90 END
```

```
NAME     :JOHN......
ADDRESS :LONDON....
PHONE    :01-000-0000

NAME     :SALLY.....
ADDRESS :NY........
PHONE    :02-0000-0000

NAME     :*
Ok
```

The above program writes the data form the INPUT statement into a sequential data file named "ADRSBOOK.DAT" on the RAM disk. Type in a name, address, and phone number after "Name:?", "Address:?", and "Phone:?", respectively. The data written into the file, as shown in the figure below, each time the PRINT # statement on line 60 is executed. Typing "*" in response to the "Name:?" message closes the data file and terminates the program.


N$
A$
T$
N$
A$
T$
N$
A$
T$
Data written by the first PRINT # statement.
Data written by the second PRINT # statement.
Data written by the third PRINT # statement.

You may use the PRINT #USING or WRITE # statement instead of the PRINT # statement.

The next program reads the data file "ADRSBOOK.DAT" created in the previous program and displays it on the screen. The program checks for an EOF (indicating the end of file) at line 20. On detecting an EOF, it jumps to line 90 and closes the file since there is no data left to be read. If no EOF is detected, the program goes on to line 30 where it reads the data, and displays it on lines 40 to 60. An "IE Error" message will appear if an attempt is made to read further data after an EOF has been detected.

The program executes the statements on lines 20 to 50 repeatedly until it encounters an EOF.

```
5  SEQUENTIAL FILE2
10 OPEN "I",#1,"A:ADRSBOOK.DAT"
20 IF EOF(1) THEN 90
30 INPUT #1,N$,A$,P$
40 PRINT "NAME    :";N$
50 PRINT "ADDRESS :";A$
60 PRINT "PHONE   :";P$
70 INPUT "READ NEXT DATA (Y/N)";Y$
80 IF Y$="Y" OR Y$="y" THEN 20
90 CLOSE
100 END
```

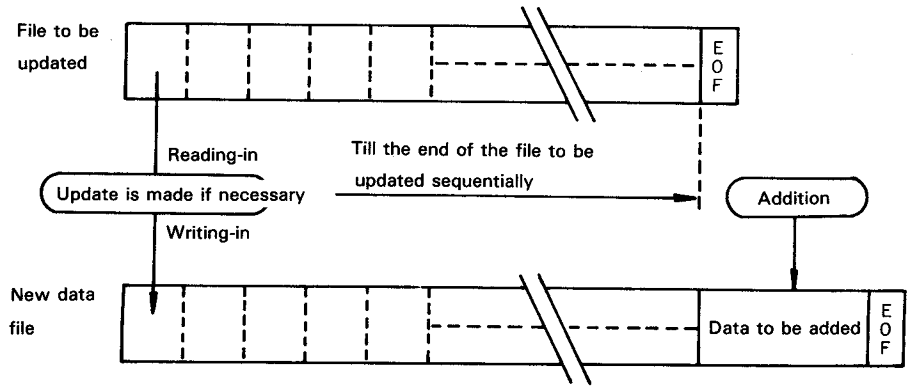
```
NAME    :JOHN......
ADDRESS :LONDON....
PHONE   :01-000-0000
READ NEXT DATA (Y/N)? Y
NAME    :SALLY.....
ADDRESS :NY........
PHONE   :02-0000-0000
READ NEXT DATA (Y/N)? Y
Ok
```

4.2.4 Updating sequential data files

After a sequential file has been created, it its not possible to update data in that file once it has been closed. The contents of a sequential file are destroyed whenever that file is opened in the "O" mode. To overcome this, the following procedure can be used:

(1) Open the data file to be updated in the "I" mode.
(2) Open a second data file in the "O" mode.
(3) Read in data from the original data file and write it to the new file, making necessary updates. .... (Updating of data.)
(4) After all the data included in the original data file has been written to the second data file, delete the original data file with the KILL command.
(5) Write the data to be added to the second data file... (Addition of data.)
(6) Close the second data file after all the data to be added has been written. Using the NAME command, rename the second data file with the name previously assigned to the original data file.

The result is a new data file which has the same file name as the original file, and which includes both the original data and the new data.



```
5 'SEQUENTIAL FILE3
100 OPEN "I",#1,"A:ADRSBOOK.DAT"
105 'Opens old file in input mode
110 OPEN "O",#2,"A:WORK.DAT"
115 'Opens temporary file in output mode
120 IF EOF(1) THEN 270
130 INPUT #1,N$,A$,P$:'Reads old data
140 PRINT "NAME    :";N$
150 PRINT "ADDRESS :";A$
160 PRINT "PHONE   :";P$
170 INPUT "  <CORRECT DATA (Y/N)>";Y$
180 IF Y$="N" OR Y$="n" THEN 250
190 INPUT "NAME    :";NX$
200 IF NX$="" THEN 210 ELSE N$=NX$
210 INPUT "ADDRESS :";AX$
220 IF AX$="" THEN 230 ELSE A$=AX$
230 INPUT "PHONE   :";PX$
240 IF PX$="" THEN 250 ELSE P$=PX$
250 PRINT #2,N$;",";A$;",";P$
260 PRINT : GOTO 120
270 INPUT "  ** ADD MORE DATA (Y/N) **";Y$
280 IF Y$="N" OR Y$="n" THEN 340
290 INPUT "NAME    :";N$
300 INPUT "ADDRESS :";A$
310 INPUT "PHONE   :";P$
320 PRINT #2,N$;",";A$;",";P$
330 GOTO 270
340 CLOSE
350 KILL "A:ADRSBOOK.DAT":'Erases old file
360 NAME "A:WORK.DAT" AS "A:ADRSBOOK.DAT":'Changes filename
370 END
```

```
run
NAME    :JOHN......
ADDRESS :LONDON....
PHONE   :01-000-0000
  <CORRECT DATA (Y/N)>? Y
NAME    :? BEN.....
ADDRESS :?
PHONE   :?


NAME    :SALLY.....
ADDRESS :NY.......
PHONE   :02-0000-0000
  <CORRECT DATA (Y/N)>? N

  <CORRECT DATA (Y/N)>? Y
NAME    :? TIMOTHY....
ADDRESS :? BERLIN.....
PHONE   :? 000-0000

  ** ADD MORE DATA (Y/N) **? N
Ok
```

4.3 Random Data File

The random data file allows data to be accessed anywhere on a disk. Although sequential data files may have different lengths, a fixed length must be set for a random data files.

Random data files are more useful than sequential data files when there are large quantities of data which must be frequently updated. They require less disk space for storage because data is recorded using a packed binary format, whereas sequential files are written as series of ASCII characters. In sequential data files, it is possible to read or write only the parts of data that need to be updated; there is no need to read or write data in sequence as is the case with sequential files. The unit of data handled in a single random access read or write operation is called a record. You can identify records in a random data file by assigning them record numbers.

4.3.1 Creating random data files

The steps required to create a random data file are as follows:

(1) Execute an OPEN statement in the "R" mode (random mode).

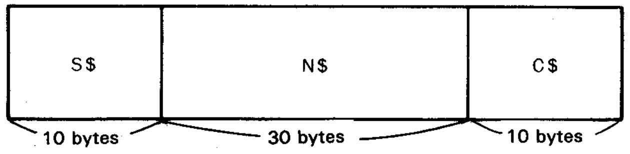
The OPEN statement assigns a file number to the file and defines the record length. If the record length is omitted, records of 128 bytes are assumed.

Example: `OPEN "R",#1,"A:STOCKLST.DAT",50`

The record length is set to 50 bytes in the above example.

(2) Specify the variables to be used for reading or writing data (buffer variables) and allocate space for them in the random file buffer using the FIELD statement.

Example: `FIELD #1,10 AS S$,30 AS N$,10 AS C$`



Note that the total equals the record length declared in the OPEN statement.

(3) Move the data to be written into the buffer variables using the LSET or RSET statement. Any numeric variables must be converted to character strings by using the MKI$, MKS$, or MKD$ function before executing the LSET or RSET statement.

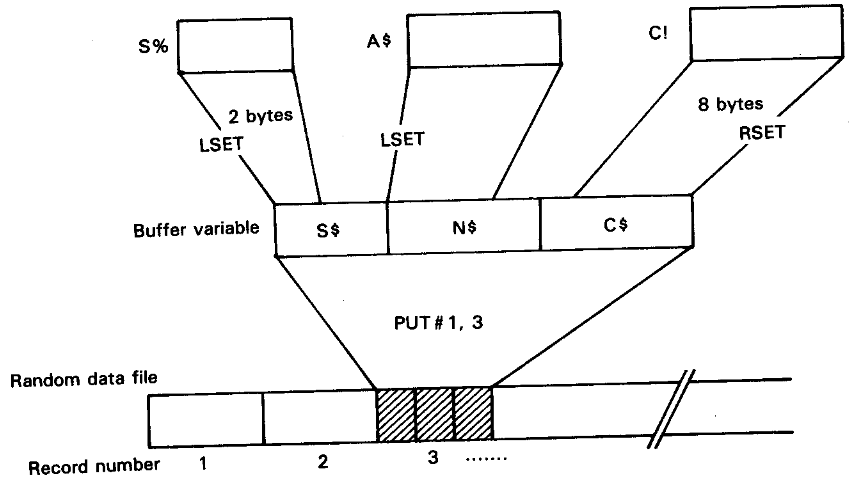
Example:

`LSET S$=MKI$(S%)`	Converts the contents of a integer variable into a character string and load it into buffer variable S$, left-justified.
`LSET N$=A$`	Loads the contents of a character variable into buffer variable N$, left-justified.
`RSET C$=MKS$(C!)`	Converts the contents of a single-precision variable into a character string and loads it into buffer variable C$, right-justified.

(4) Write records into the random data file using the PUT statement.

Example: `PUT #1,S%`

The record number indicating the position of a record in the file is placed in variable S%. When S% is specified as 3, the record is written into record position 3 of the file.



```
5 'RANDOM FILE1
100 OPEN "R",#1,"A:STOCKLST.DAT",40
110 FIELD #1,2 AS S$,30 AS N$,4 AS C$
120 PRINT
130 INPUT "STOCK NO.  (0 : End) ";S%
140 IF S%=0 THEN 220
150 INPUT "ITEM NAME ";A$
160 INPUT "PRICE     ";C!
170 LSET S$=MKI$(S%)
180 LSET N$=A$
190 C$=MKS$(C!)
200 PUT #1,S%
210 GOTO 120
220 CLOSE
230 END
```

The above program writes data entered from the keyboard into a random access file. Line 110 specifies buffer variables S$, N$, and C$, and their sizes. Lines 130, 150, and 160 allow data to be entered from the keyboard for storage in the random access file. In this example, STOCK NO. is also used as the record number. Lines 170, 180, and 190 load the input data into the buffer variables, and line 200 writes the record into the file.

NOTE:
Once a buffer variable is specified in a FIELD statement, do not use that variable on the left-hand side of an INPUT or LET statement. If you do, the specification for the variable in the FIELD statement will be canceled; the variable will be treated as an ordinary variable.

4.3.2 Reading random data files

The following steps are required to retrieve data from a random data file:

①Execute an OPEN statement in the "R" mode (random mode).

Example: `OPEN "R",#1,"A:STOCKLST.DAT",50`

②Specify the buffer variables and their sizes using the FIELD statement.

Example: `FIELD #1,10 AS S$,30 AS N$,10 AS C$`

③Read records using the GET statement.

Example: `GET #1,S%`

④Since numeric values are converted into binary format character strings when they are placed in the random data files, they must be converted back into the original format before reuse by the program. This is done using the CV1, CVS, or CVD functions.

Example: `PRINT CVI(S$),N$,CVS(C$)`

The following sample program reads the random file created in section 5.3.1. The number of records to be read is entered from the keyboard at line 130. Line 150 reads the record specified by the record number, and lines 160 to 180 display the contents of the record.

```
5 'RANDOM FILE2
100 OPEN "R",#1,"A:STOCKLST.DAT",40
110 FIELD #1,2 AS S$,30 AS N$,4 AS C$
120 PRINT
130 INPUT "STOCK NO. (0 : End) ";S%
140 IF S%=0 THEN 200
150 GET #1,S%
160 PRINT USING "####";CVI(S$)
170 PRINT N$
180 PRINT USING "####.##";CVS(C$)
190 GOTO 120
200 CLOSE
210 END
```

While a file is open, data can be read or written until the file is closed. The LOC function is useful to control the flow of program execution according to the total number of records which have been written to the file. With random files, the LOC function returns the record number last written to or read from the file.

The following statement ends program execution if the record number is greater than 50:

Example:

```
IF LOC(1)>50 THEN END
```