BASICALLY BASIC

Graham Hall, B.Sc.

Part 14

Files

BASIC has three methods of supplying data to a program:
1. The INPUT statement — the user interacts with the computer while the program is running. Each time the program is run new data is requested and is input from the terminal.
2. The READ, DATA and RESTORE statements — the READ statement directs the program to read from a list of values built into a data block by a DATA statement. In terms of program execution time, it is much more efficient to use READ and DATA statements than INPUT statements because the user does not have to interact with the program when it is run. The RESTORE statement enables the same data to be used more than once during the execution of the program.
3. The file statements — data can be accessed from or written to a uniquely named file which is separate from the main program. The computer system stores a file on peripheral devices such as disks or magnetic tapes. As the file is stored under its own name separate from the program which created it, different programs can use the file name to make that file's data available.

The use of the INPUT, READ, DATA and RESTORE statements has already been described. Now the BASIC statements used to create a file, place data into it, and make it available to a program are described. Since files are externally stored and program independent, their configuration and characteristics such as size and data access depend on the computer system and peripheral device that you use. For these reasons the description to follow will be general and serves only as an introductory explanation. For a specific and more complete description refer to your systems user guide.

Most personal computer systems use 'sequential files'. These files are usually terminal — format files in which the contents consist of a collection of ASCII characters stored in lines of various lengths exactly as they would appear on the terminal. A sequential file is one in which the data assigned to the file is arranged one item after another from the beginning of the file. To retrieve an item from the file all items preceding it must be retrieved first. Some systems also allow virtual array files and record files to be created but these will not be described here.

All versions of BASIC have statements to:
i) create a new file and to assign it a unique name,
ii) place data into a file (writing to a file),
iii) access data from a file (reading a file),
iv) close a file which has previously been opened by a program.
These will now be described.

Creating and Opening a File

The OPEN statement enables a new file, or an existing file, to be opened and associated with a file number which establishes a communication channel between the program and the file. Some versions of BASIC only allow one file at once to be used by a program so it is not required to associate a file number since this will be a system default.

Usually the OPEN statement performs several functions which include naming the file, designating the operations to be performed and opening a communication channel. An example of this would be:

10 OPEN "string" { FOR INPUT / FOR OUTPUT } AS FILE # expression

The OPEN "string" component of this statement either references a file which already exists, in which case the file name enclosed within quotation marks is used to locate the file or names a new file. The file name is a string of alpha-numeric characters enclosed within quotation marks. Usually it must be less than a certain maximum number of characters depending on the system being used. It could also be a string variable.

The FOR INPUT or FOR OUTPUT component of the open statement is optional — one or neither of these portions can be used. When the FOR INPUT option is used BASIC opens the file specified as the file name and allows the data it contains to be used by the program. An error message is returned and displayed on the terminal if BASIC tries to open a file which does not exist. The FOR OUTPUT option creates a new file and allows the program to write data to it. If neither is specified BASIC searches for a file with the name specified in "string". If the file is found it is opened; otherwise a new file assigned to that name is created.

The AS FILE # expression portion of the OPEN statement associates the file and the program with a common communication channel. This enables the file, which is stored on a peripheral device, to be associated with the current program which is in the computer's main memory area. The location associated with the file name is called a channel number and is specified in the expression part of the AS FILE # portion. This location can then be accessed by the program.

The following examples show how the OPEN statement is used:

```
20 OPEN "SUBJECT" FOR INPUT AS FILE # 1
```

This statement opens the file named SUBJECT. The FOR INPUT portion shows that the file already exists and that the data is to be read from the file. The AS FILE # portion establishes communication channel 1 as the link between the program in main memory and the file on a peripheral device.

```
20 OPEN "INFORM" AS FILE # 3
```

This statement causes BASIC to search for the file named INFORM. If the file exists it is opened and the program can access its data; if the file is not found a new file is created and assigned to the file name INFORM. The program can then write data to this file. The file is accessed by channel number 3.

```
10 OPEN "RESULTS" FOR OUTPUT AS FILE # 2
```

This statement creates a new file which is assigned to the file name RESULTS. The FOR OUTPUT portion of the statement notifies BASIC that this is a new file to which data can be written. The AS FILE # 2 portion of the OPEN statement establishes communication channel 2 as the link between the program in main memory and the file on a peripheral device.

Some versions of BASIC have different OPEN statements to open a file for reading and to open a file for writing. For example, to open a file to accept data the statement could be WOPEN (write open) but to open a file to retrieve data the statement could be ROPEN (read open). This depends on the system you are using and will be explained in the user's guide for your system.

Closing a File

All files opened by a program should be closed before the program terminates execution. Unless they are closed the file may become 'corrupt', that is some of the contents may be spuriously altered or destroyed. The CLOSE statement is used to close a file and dissociate it from a communication channel. After a file has been closed it cannot be accessed until it has been re-opened.

The general format of the CLOSE statement is:

line number CLOSE ([ ] expression list)

where expression list may be one file number or a list of opened file numbers separated by commas. The part of the CLOSE statement shown within square brackets is usually optional. If no expressions are specified all files opened by the program are closed.

The following examples illustrate the use of the CLOSE statement:

```
10 CLOSE # 1:REM CLOSE FILE ASSOCIATED WITH CHANNEL 1
20 X=3
30 CLOSE 2,X,3+2:REM CLOSE FILES 2,3&5
40 CLOSE:REM CLOSE ALL FILES
```

Writing to a File

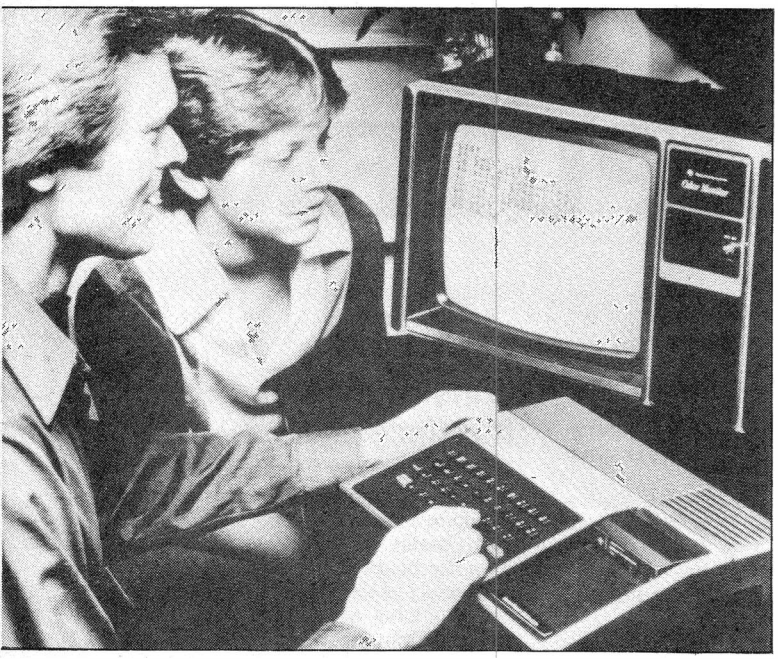
To write data to the terminal the BASIC PRINT statement is used. The PRINT statement can also be used to write data to a file. The general format is:

PRINT (#) channel number, list

where channel number can be the communication channel number associated with a file that has been opened with the OPEN statement or zero. If zero is specified the output is to the terminal. The # character preceding the channel number is usually optional. List can be any numeric or string expression or a numeric or string variable. Each item in the list must be separated with a comma or a semicolon. Also the first item in the list must be separated from the channel number by a comma.

The following short program opens a file called EXAMPLE for output and then writes the string "FIRST LINE OF FILE EXAMPLE" to the file when the program is executed.

```
10 OPEN "EXAMPLE" FOR OUTPUT AS FILE # 1
20 PRINT # 1, "FIRST LINE OF FILE EXAMPLE"
30 CLOSE # 1
40 END
```



The same result would be achieved by assigning the string to a string variable and then printing the string variable, i.e. line 20 could be substituted with the lines:

```
15 LET M$ = "FIRST LINE OF FILE EXAMPLE"
20 PRINT # 1, M$
```

The next section shows how this data can be retrieved from the file.

Reading from a File

To input data to a program from the terminal the BASIC INPUT statement is used. The INPUT statement can also be used to retrieve data from a file to use as input to the program. The general format is:

INPUT (#) channel number, list

where channel can be the communication channel number associated with a file previously opened using the OPEN statement, or zero. If zero is specified the input is from the terminal as for the program INPUT statement.

List can be a single string or numeric variable or a list of variables separated by commas. Also the first item in the list must be separated from the channel number by a comma. The '#' character preceding the channel number is usually optional.

The INPUT # statement line that retrieves data from a file must duplicate the format of the PRINT # statement that wrote the data. Also the type of variable used to store the retrieved data must correspond to the type of data item being retrieved. When the INPUT # statement is to request more than one data item, the data must have been written to the file separated by a string constant comma. This is because the INPUT # statement reads data in the file in the same manner as a program INPUT statement (where the data following a DATA statement is separated by commas). For example, the statement which writes the integers 1 and 2 and the string "THREE" to the file assigned to channel number 1 is:

```
10 PRINT # 1, 1, ",", 2, ",", "THREE"
```

The program line retrieving this data would be:

```
20 INPUT # 1, A, B, Z$
```

When this statement is executed the first data item retrieved from the file is assigned to the variable A, the next to variable B and finally the string is assigned to the string variable Z$. These variables can then be used in other BASIC statements to perform any desired operation on the data within the program.

The following programs demonstrate how data can be written to and retrieved from a file using BASIC file statements.

```
10 REM FILE EUROTEMPS TO BE WRITTEN TO
20 OPEN "EUROTEMPS" FOR OUTPUT AS FILE # 1
30 READ P$, C, F
40 IF P$ = "" THEN GOTO 130
50 PRINT # 1, P$; ","; C; ","; F
60 GOTO 30
70 DATA "LONDON", 18.7, 65.7
80 DATA "AMSTERDAM", 21.0, 69.8
90 DATA "EDINBURGH", 17.8, 64.0
100 DATA "PARIS", 22.8, 73.0
110 DATA "MUNICH", 22.8, 73.0
120 DATA "", 0, 0
130 PRINT # 1, P$; ","; C; ","; F
140 CLOSE # 1
150 END
```

RUN

The program consists of the following lines:

Line 10 — The REM statement serves only as a comment. The characters after REM are ignored.

Line 20 — The OPEN FOR OUTPUT statement creates a new file and assigns it the name "EUROTEMPS". The AS FILE # 1 portion establishes communication channel number 1 as the link between the program in main memory and the file on a peripheral device.

Lines 30, 70 to 120 — The READ statement on line 50 is associated with the DATA statements on lines 70 to 120. When it is executed the READ statement assigns data from the DATA statements to the variables P$, C and F. Each data line is a string followed by two numeric constants.

Lines 40, 120 — The end of the data is signalled by a null string followed by two zeros. The IF THEN statement tests the string assigned to P$ to see if it is null. If the condition is true the program control is directed to the CLOSE statement on line 140 which closes the file. Some versions of BASIC have file statements which test whether the end of the file has been reached. If this facility was available on your system it would not be necessary to set up a dummy data item to signify the end of file.

Line 50 — The PRINT # statement writes the variables P$, C and F to the file associated with channel 1. The data is written to the file separated by string constant commas. This is to enable the data to be retrieved using the INPUT # statement.

Line 60 — The GOTO statement repeats lines 30, 40 and 50 to write all the data contained in the DATA statements to the file.

Line 130 — The PRINT # statement writes the null string and zeros to the file.

Line 140 — The CLOSE statement closes the file EUROTEMPS and disassociates it from communication channel 1.

Line 150 — The END statement signifies program completion.

To write the data to the file EUROTEMPS the program must be executed by typing RUN. After program execution is complete the data contained in the file EUROTEMPS can be made available to any BASIC program.

A program to retrieve the data contained in the file EUROTEMPS is:

```
10 REM DATA TO BE RETRIEVED FROM FILE EUROTEMPS AND
   DISPLAYED ON THE TERMINAL
20 OPEN "EUROTEMPS" FOR INPUT AS FILE # 1
30 INPUT # 1, C$, C, F
40 IF C$ = "" THEN GOTO 70
50 PRINT C$, C, F
60 GOTO 30
70 CLOSE # 1
80 END
```

RUN

```
LONDON          18.7    65.7
AMSTERDAM       21.0    69.8
EDINBURGH       17.8    64.0
PARIS           22.8    73.0
MUNICH          22.8    73.0
```

The program consists of the following lines:

Line 10 — The REM statement serves only as a comment.

Line 20 — The OPEN statement causes BASIC to locate the file EUROTEMPS on the peripheral storge device. The FOR INPUT portion of the statement shows that the file already exists and that the data it contains is to be retrieved. AS FILE # 1 associates the file with communication channel number 1.

Line 30 — The INPUT # statement reads the data items from the file and stores them in variables. This duplicates the format of the PRINT # statement on line 50 of the program that created the file. That is, the variables match in type and number, and a comma separates the data items in the file. If this was not the case, BASIC would display an error message when the program is executed.

Line 40 — The IF THEN statement tests for the end of the file. The last data line written to the file consisted of a null string followed by two zeros. When a null string is retrieved from the file the condition is satisfied and the program is directed to the CLOSE statement on line 70.

Line 50 — The PRINT statement outputs the contents of the file as stored in the variables C$, C and F to the terminal. The variables are separated by commas so each argument is output to its own field. The output from the program is shown after the RUN command.

Line 60 — The GOTO statement repeats lines 30, 40 and 50 until all of the data are read.

Line 70 — The CLOSE statement closes the file EUROTEMPS and disassociates it from communication channel number 1.

Line 80 — The END statement signifies program completion.

BASIC programs are also stored in files. The BASIC commands to store and retrieve programs in files from a peripheral device (the SAVE and LOAD, or OLD commands) have been described previously. There are also BASIC commands to delete files which are no longer required.