Pocket Guide

Programming for the Apple

John Gray

Programming Pocket Guides

Pitman Programming Pocket Guides

Programming	John Shelley
BASIC	Roger Hunt
COBOL	Ray Welland
FORTRAN	Philip Ridler
Pascal	David Watt
FORTRAN 77	Clive Page
Programming for the BBC Micro	Neil Cryer and Pat Cryer
Assembly Language for the 6502	Bob Bright
Assembly Language for the Z80	Julian Ullmann
Programming for the Apple	John Gray

This series of pocket size reference guides provides you with reliable descriptions of the salient features of all the important languages and micros.

Programming for the Apple is intended for the reader who has a reasonable knowledge of BASIC and wishes to develop expertise in handling the operating procedures of the Apple][or Apple //e microcomputers.

The Publishers would welcome suggestions for further improvements to this series. Please write to Alfred Waller at the address below.

PITMAN PUBLISHING LTD
128 Long Acre, London WC2E 9AN

Associated companies
Pitman Publishing Pty Ltd, Melbourne
Pitman Publishing New Zealand Ltd, Wellington
Copp Clark Pitman, Toronto

Consultant Editor: David Hatter

First edition, 1984

Printed in Great Britain at The Pitman Press, Bath

ISBN 0 273 01991 0

Index

How to use this Pocket Guide

This guide is intended to help those of you who have gained access to an Apple][or an Apple //e, who have a reasonable idea what a microcomputer can do and have some knowledge of the BASIC programming language. The growing number of Apple machines in business, industry and education coupled with the appearance of Apple][machines in the secondhand market make this category of Apple user an increasingly large one. The main features of the Guide are the outline of Apple operating procedures (concentrating especially on use of disks) and the development of a number of 'Applesoft Libraries' made up of subroutines that can be appended to users' own programs. Their function is to have you learn about Applesoft by actually using its most commonly met commands and statements. Some advice is also proffered as to how best to take advantage of various possible 'developments' of your system such as use of an Applesoft compiler, getting rid of the garbage problem and the use of BitStik for graphics.

Apple][systems vary tremendously according to which particular bits of hardware are plugged into the Apple. This guide will assume that your system has 48K of random access memory and the Autostart ROM (read-only memory chip) which allows you to have turnkey 'booting up' from disks. You will need at least one disk drive and Applesoft BASIC either permanently present in the machine in a ROM or available on disk and loadable into a special RAM card mounted in extension Slot 0.

To get the best out of this guide, work through all the examples/demonstration programs and build up the Applesoft libraries. These libraries aren't complete but they should point you in the right direction. The intention has been to introduce Applesoft and DOS statements and commands in an applied context; so make notes on their use as they are introduced. Apologies in advance for all the things that *couldn't* be included for lack of space! The Apple is a great machine. Enjoy your exploring.

Operating the Apple

Installation

There just isn't enough space in this small guide to provide full installation details. We'll assume that you have had enough experience or help to make sure that

(*a*) You have your disk drive(s) connected to a disk controller card mounted in Slot 6.

(*b*) You have a TV monitor attached to the video output at the right rear of the Apple (or a TV receiver attached to a suitable RF modulator and colour card in Slot 7).

(*c*) Your printer (if you are using one) is attached to a suitable printer interface card mounted in Slot 1 and that you know how to switch the printer into ONLINE mode so it can accept output from the Apple.

If your Apple equipment doesn't appear to be working correctly, check that all the parts of the system are correctly connected to the mains power supply and switched on. If you still have a problem, check the things mentioned above — or get your supplier to check if you don't understand what it means! Above all don't fiddle with anything inside the Apple when the mains power is on.

All the demonstration programs and routines have been tested, so if you have trouble getting them to work, make sure that

(*d*) you have entered them exactly — no missing or incorrectly numbered lines, missing statements or incorrect values — please notice that in lots of the sample programs long program lines have been broken up, for clarity, into separate lines of the page. Don't press RETURN until the whole line has been entered.

(*e*) you have all the necessary library routines loaded into memory.

(*f*) you are correctly following the instruction to 'enter' as given on Page 4.

Getting started

To work through this section you will need the floppy disk labelled DOS 3.3 System Master that you should have received with your disk drive. Insert this disk into Drive 1. Now turn the mains power on. The disk drive begins to work and the red IN USE lamp on the front lights up. The system is being 'booted up' so that the Apple can communicate with its disk drives. If everything seems to be all right just sit back for a few seconds — if it isn't then check the points made on Page 1 and Page 2 concerning system components and installation. If your machine has a language card fitted you will see the message that INTEGER (or APPLESOFT) is being loaded. After some time the screen will show the prompt that indicates your machine has Applesoft BASIC and that it is available to you] and next to it there should be a flashing square. This flashing square is known as the 'cursor'. The cursor will be a solid square if your Apple is an **Apple][** and a flashing dotted square if you have an **Apple // e**. Whenever this flashing cursor is visible it indicates that the Apple is waiting for some kind of instruction or input from you. If you cannot see the] prompt it may be that there is a > prompt immediately to the left of the cursor instead. If there is, this indicates that your Apple is offering you a different kind of BASIC, Integer BASIC, which is standard on machines delivered in some parts of the world. You can try getting into Applesoft BASIC by typing FP and then pressing the RETURN key. If Appleroft is also available the] prompt will immediately appear, if it doesn't then you are stuck with Integer BASIC. There are many similarities between the Applesoft and Integer BASICs and much of this guide will be of interest to the Integer BASIC user. Nevertheless, there are important differences, so beware!

You don't always have to turn the mains power off in order to get your Apple started from scratch. If you want to reboot the machine when it is already switched on, type in PR#6 and press the RETURN key. To the Apple this is just like your turning its mains power off, then on again.

The keyboard

Make sure you can find all the alphabet characters, punctuation and numbers and the **SPACE** bar which lies across the bottom of the keyboard. Remember that if a key has two symbols on it you get the upper symbol by holding down the **SHIFT** key while pressing the key whose symbol you want. You should also locate **ESC** (the ESCAPE key) which you will often be prompted to use, although pressing it seems normally to have little effect! The 'control key', marked **CTRL**, is an important key which must always be used in conjunction with some other key. For example, if you are directed to press CTRL-C this means you must hold down the CTRL key while you press the C key. The Apple][provides you with two keys marked ← and → which we can call **LEFT ARROW** and **RIGHT ARROW**. These keys allow you to move forward or backward along a line on the screen. If you move backward when entering a program line, inputting data or entering a command, this has the effect of removing whatever characters you backspace over from the input line — the characters may not disappear from the screen but they will be ignored by the Apple. In this way you can type out any mistakes you make. All keys can be used in conjunction with the 'repeat key' which is labelled **REPT**. Holding REPT down will cause any other key you hold down to be printed as often as you wish — if you go too far just come back with the LEFT ARROW key. The most used key of all must be the **RETURN** key. *Please notice that throughout this guide whenever you are directed to ENTER something at the keyboard it is implied that you type in whatever the entry is and then press RETURN.* RETURN is the sign to the computer that you have typed in whatever was required and that it should now be acted upon. Dangerously close to the RETURN key lies the **RESET** button! The primary purpose of this button is to stop everything happening and it should be used with caution! Having removed the Apple's cover, it is possible to disable the RESET button by moving the position of a sliding switch on the Apple's keyboard encoder board. After moving the switch, pressing CTRL–RESET has the same effect RESET alone used to have.

ASCII Codes

It is worth looking a little closer at what the various keys mean to the Apple by entering and using the following short BASIC program. First enter NEW — type in NEW and press the RETURN key. This will clear any program present in the Apple's memory. Type in the lines of the program one by one, pressing RETURN at the end of each line.

```
10  REM *** ASCII CODES ***
20  GET KY$ : REM RECEIVES CHARACTER
30  PRINT "TO YOU ... "KY$; : REM SHOWS IT
40  PRINT SPC(5);: REM PRINTS 5 SPACES
50  PRINT "TO APPLE ... ";ASC(KY$): REM CONVERTS
    TO ASCII
60  GO TO 10
```

Enter RUN (don't forget to press RETURN) and then press any key. All the keys will have some effect apart from CTRL, SHIFT and REPT which don't do anything except when used in combination with another key. Every time you press a key the program feeds back a code number of the character you select. The code numbers it gives you are the standard ASCII codes which, although you may never realize it, are what the Apple always receives from the keyboard whenever you press a key! Try all the keys (don't forget the CTRL– and SHIFT– combinations) and then press RESET (or CTRL–RESET) to leave the program. To the Apple every character is a number — and machines like the Apple can generate 256 different characters. Enter NEW, type in the following program and RUN it.

```
10  REM *** CHARACTER STRINGS ***
20  FOR L = 0 to 255
30  PRINT "CODE ";L;" GIVES ";CHR$(L)
40  NEXT L
```

Press CTRL–S to stop the list at any time. Lots of codes produce invisible 'control characters' and one set of alphabetic characters gives lower case on a printer.

Program editing

As you work through the numerous program examples and library entries presented in this guide you will inevitably make errors. In any event there are numerous occasions on which you are required to amend lines that have already been entered. This means that you must know how to edit program lines — and unfortunately the Apple][‘s editing facilities are NOT the machine’s best feature! Work through the following guided example to see what’s involved.

Suppose the program line

```
30  HTAB H : GOSSUB 300
```

needs to be amended to

```
30  HTAB H : VTAB V : GOSUB 300
```

(*a*) LIST the part of the program containing the line to be edited. In this case, enter LIST 30.

(*b*) Press the ESCAPE key *once* to set the Apple into the edit mode. The four keys I, J, K, M will now cause the cursor to move respectively up, left, right and down across the screen until the edit mode is left as a result of some other key being pressed.

(*c*) Press I and the cursor will move up by one line — if a letter I is printed instead you mustn’t have pressed ESC once and you must use the LEFT ARROW to backspace and start all over again. Press I as many times as is necessary to take the cursor to the level of the line number of the line to be edited.

(*d*) Next press J as often as required to bring the cursor over the first digit of the line number.

(e) Press RIGHT ARROW repeatedly over the parts of the line that are correct. If you go too far just backspace with LEFT ARROW. Any characters you RIGHT ARROW over, including blank spaces, will be included in your edited line. If you want to stop blanks being included you must use ESC and K to skip over them.

```
30  HTAB H :
```

(f) You need to enter VTAB V : but there isn't any space! Press ESC once and I once to bring the cursor onto the line above. Press the space bar once to leave the edit mode and then type in VTAB V : Your line will now look odd, but don't worry, everything will be all right!

```
                VTAB V :
30  HTAB H : GOSSUB 300
```

(g) Now press ESC once and M once to bring the cursor back down again. Press J as often as needed to bring the cursor over the G of GOSSUB — the next correct character of the original line.

(h) Press RIGHT ARROW as often as needed to bring the cursor over the top of the second (and unwanted) letter S in GOSSUB. Press ESC once and K once to skip over the unwanted S.

(i) With the cursor now lying over the letter U press RIGHT ARROW several times to bring the cursor to the end of the line. Press return to enter your edited line and LIST the line to check your handiwork.

Program editing with the Apple is more tedious and prone to error than with some other micros. The best thing one can say is that you get used to it! Remember also that you can always change a faulty line by simply typing in from scratch the correct version — any existing line with the line number you use will be lost from memory.

BASIC and DOS

An Apple with disks offers you the use of two distinct facilities.

(*a*) A programming language to let you get the Apple to accept input, do calculations, draw pictures, print things . . . this is what **Applesoft BASIC** provides you with. You have already used Applesoft to run the short keyboard program presented earlier.

(*b*) A disk operating system to allow your Apple to communicate with the disk drive so that you can save/load programs you have written or bought and any of the data these programs might use . . . this is what the Apple **Disk Operating System (DOS)** provides. It was this program, Apple DOS, that was loaded from the disk in Drive 1 when you turned the power on — this loading of DOS is known as 'booting' the system.

Just to check that DOS is loaded and that everything is all right type in **CATALOG** and press the RETURN key. The disk light should come on for a few seconds and the screen should fill with a list of names — these are the names of the programs and files that are stored on the disk. At the bottom of the list you should see the] prompt and, next to it, the flashing cursor. If you can't see the Applesoft prompt but only the flashing cursor just press the SPACE bar across the bottom of the keyboard. This should cause the rest of the disk's contents to scroll up onto the screen and the cursor should now be visible. If it isn't then press SPACE again. CATALOG shows you 18 items from the disk's list of contents at one time and waits for you to press a key before going on to the next set.

Contents of disks

Entering **CATALOG** provides important information about what disks contain. The CATALOG of a disk might produce a list like this . . .

```
* A 002  HELLO
* I 018  ANIMALS
* B 020  FID
  B 034  PICTURE 6
  T 057  MAILSHOT : JUNE
  B 016  RUNTIME
* A 009  COPYA
```

The four columns of the CATALOG tell you

(*a*) *Whether the file is 'locked' or not* — if it is locked there is a * character in the first column.

(*b*) *What type of file each one is*

A Applesoft program files
I Integer BASIC program files
B Binary files
T Text files

(*c*) *The amount of space each file takes up on the disk* — measured in 'sectors' of 256 characters each. Disks initialized with DOS 3.3 have 560 such sectors but they aren't all normally available for you to store files on. Normally disks contain a copy of the DOS program which occupies 48 sectors. Later, you will find out how to create disks on which this space is available to you.

(*d*) *The name of the file* — with Apple DOS files can be called anything, as long as the name:

(1) isn't longer than 30 characters
(2) begins with a letter
(3) doesn't contain a comma

DOS commands

The Apple knows when and how to communicate with its disk drives from the DOS commands you send it. You have already used one, CATALOG, directly from the keyboard and you will meet many more as you progress through this guide. All DOS commands can be used from within programs if immediately preceded by CHR$(4) . . . CTRL–D . . . as part of a PRINT statement. Many Apple programmers assign the value CHR$(4) to the variable D$ early in their programs and then PRINT D$ to precede DOS commands in forms like PRINT D$;"LOAD HELLO,D1" Any DOS commands you use can specify the particular slot, drive and volume that you want to access, if you wish — none MUST be specified.

Slot For most Apple users the slot specification will rarely be used since most Apple systems have two floppy disk drives connected to a single disk controller card in Slot 6. However, if you do have more than two floppy disk drives, you will need to use the optional slot specification. You might enter RUN HELLO,S5,D1 or CATALOG,S6,D2 . . . the slot specification is used in just the same way from within a program. For example PRINT CHR$(4);"BLOAD FID,S6,D1"

Drive Most Apple users will frequently need to specify the drive to be accessed. When the Apple is booted up, Drive 1 is automatically selected as the drive that DOS commands will be directed towards. As soon as your programs need access to Drive 2 a DOS command containing a drive specification will have to be sent. Having booted up a data handling program from Drive 1, it might need access to some data in a file on Drive 2. The program will have to contain a line such as PRINT CHR$(4);"OPEN MAILSHOT,D2" It's important to note that once DOS has been directed to Drive 2 all subsequent DOS commands will also be directed towards it. As soon as Drive 1 is required again an appropriate drive specification will have to be made.

Volume One of the items of information first listed when you CATALOG a disk is its Volume Number. Disks can have Volume Numbers between 1 and 254. Disks are assigned their Volume Number when they are initialized. When you use the DOS command INIT by itself the new disk is automatically assigned a Volume Number of 254. It is only assigned a different number if you specify it with an INIT command like INIT WELCOME,V100. It can be useful to specify volume numbers for data disks so that particular sets of similarly structured data files can be distinguished from each other under program control.

Security and order In addition to CATALOG there are four more DOS commands which are commonly used directly from the keyboard to keep the contents of disks sorted out. Each can be used with any type of file.

LOCK: Prevents a file from being overwritten, deleted or renamed.

```
LOCK MAILSHOT,D2
PRINT CHR$(4);"LOCK HELLO,V254"
```

UNLOCK: Removes the 'locked' status of any file.

```
UNLOCK MAILSHOT,D2
PRINT CHR$(4);"UNLOCK HELLO,V254"
```

DELETE: Removes the specified file from the disk.

```
DELETE FID,D1
PRINT CHR$(4);"DELETE MAILSHOT,S6,V254,D1"
```

RENAME: Changes the name of any file on the disk.

```
RENAME FILE 1,FILE 2
RENAME FILE 1,FILE 2,S6,D2
PRINT CHR$(4);"RENAME FILE 1,FILE 2"
```

If the new name you give to the file might be the same as that of a file already on the disk, then DELETE the existing file first— or else you'll finish up with two files having the same name!

Initializing disks

Before you can do much with DOS you have to be able to prepare blank disks so that they can be used with the Apple to store programs and data files. This procedure is known as 'initializing' the disk. Without being initialized using the DOS command **INIT**, a disk cannot be 'read' by the Apple; so every disk you use must have this procedure carried out on it. Having booted up the Apple using the DOS 3.3 System Master disk you should replace this disk with an uninitialized disk in Drive 1. First enter NEW and then enter INIT HELLO . . . the red disk light will come on and the disk drive will work for a couple of minutes. When the light goes off you should enter the command CATALOG and you should be presented with

```
]CATALOG

DISK VOLUME 254

A 002 HELLO
```

The program listed on the disk as HELLO will run automatically whenever you boot up your Apple from this disk. As it stands this HELLO program doesn't seem very exciting and it doesn't seem to do much. However, you can replace the existing HELLO program by saving any other program under the name HELLO and it will become the program that runs automatically when you turn the Apple on. This '**turnkey operation**' allows you to make your software very user friendly indeed. When disks are initialized using the standard routine described above 12K of their storage space is used up creating a copy of the DOS program itself on the disk — this is in order that the disk can itself be successfully booted up from. Many of the disks you will use will never need this DOS image because you will never need to boot up your Apple system from them. This is especially the case for the data disks you will use with your word processor package and your database programs.

Such disks simply act as receptacles for the files that the program disks create on them. Here's a program to let you initialize your disks without DOS and so make another 12K available on each disk you initialize in this way. When you have entered it save the program on your workdisk under the name DOSFREE.

```
100  REM *** DOSFREE ***
105  HOME : VTAB2
110  FOR I = 1 TO 40 : PRINT "-"; : NEXT
120  INVERSE : PRINT "DOS -FREE DISKS" : NORMAL
130  FOR I = 1 TO 40 : PRINT "-"; : NEXT
140  PRINT : PRINT
150  PRINT "DEFAULTS SLOT = 6 DRIVE = 1 VOLUME = 254"
160  PRINT "PRESS RETURN TO ACCEPT DEFAULT" : PRINT
170  S = 6 : INPUT "SLOT ?    ";S$ : IF VAL(S$) > 0
     AND VAL(S$)<8 THEN S = VAL(S$)
180  D = 1 : INPUT "DRIVE ?    ";D$ : IF VAL(D$) > 0
     AND VAL(D$) < 3 THEN D = VAL(D$)
190  V = 254 : INPUT "VOLUME ? ";V$ : IF VAL(V$) > 0
     AND VAL(V$) < 255 THEN V = VAL(V$)
200  PRINT : PRINT
205  FLASH : PRINT "CAUTION" : NORMAL : PRINT
210  PRINT "ABOUT TO FORMAT THE DISK IN"
220  INVERSE
225  PRINT"SLOT = ";S;" DRIVE = ";D;" AND VOLUME = ";V
     : NORMAL
230  NORMAL : PRINT "PRESS ESCAPE TO ABANDON"
240  PRINT "OR THE RETURN KEY TO START"
250  PRINT
260  GET S$ : IF S$ = CHR$(27) THEN GO TO 100
270  IF S$ <> CHR$(13) THEN GO TO 260
275  PRINT
280  POKE 44723,0
290  POKE 44802,234 : POKE 44803,234 : POKE 44804,234
320  PRINT CHR$(4);"INIT HELLO,S";S;",D";D;",V";V
330  PRINT CHR$(4);"DELETE HELLO"
340  POKE 44723,12
350  POKE 44802,32 : POKE 44803,74 : POKE 44804,183
```

Copying disks

Floppy disks are a very reliable way of storing programs and data if they are used and stored properly. All the same, accidents of various kinds do happen and, if the contents of your disks are at all important to you, you must know how to make backup copies. Your System Master disk provides two ways of 'backing up': one allows you to back up a whole disk, the other allows you to back up particular files and programs.

(*a*) *Backing up a whole disk* Insert your System Master Disk into Drive 1 and enter RUN COPYA . . . a program will load from the disk. When the disk light goes off you will be prompted with

APPLESOFT DISKETTE DUPLICATION PROGRAM

ORIGINAL SLOT: DEFAULT = 6

Just press RETURN and respond appropriately to the series of prompts which follow. Your 'original' is the disk that you want a copy of. The duplicate can be any disk that doesn't contain something you want — it can be a brand new disk or a previously used one. It doesn't matter whether the disk has been initialized or not, whatever information is on the disk it will be removed when COPYA runs. If you have two disk drives it's best to get into the routine of having the 'original' disk in Drive 1 and the 'duplicate' in Drive 2. That way you are less likely to accidentally copy onto the original and lose valuable information. If you have just one disk drive you can still use COPYA. When prompted to enter the drive numbers for the original and the duplicate you should enter 1 for both. The Apple will copy the original in several chunks with you having to exchange the original and duplicate disks at each stage. It's when backing up disks that you'll most wish that you had two disk drives! As a security measure, you might want to place a 'write protect tab' over the notch on your original disk during copying so as to preclude the disaster that can result from mixing the disks up.

(b) *Backing up particular files* Insert your Systems Master Disk into Drive 1 and enter BRUN FID . . . **BRUN** causes a machine code program to load and run — a menu appears:

<1> COPY FILES
<2> CATALOG
<3> SPACE ON DISK
<4> UNLOCK FILES
<5> LOCK FILES
<6> DELETE FILES
<7> RESET SLOT & DRIVE
<8> VERIFY FILES
<9> QUIT

WHICH WOULD YOU LIKE ?

The FID program is a series of utilities that are there to help you keep your disks in order. Enter 1 to select the file copying routine. When prompted, enter 6 as the source slot and enter appropriate drive numbers for your 'original' and 'backup' disks. Enter 1 for both if you have just one disk drive. Next, you are asked for the FILENAME? . . . at this point you can either name a single file for copying, or, using the 'wildcard' symbol =, have a series of files copied at once. Suppose one of your disks contains the files FILE DEMO 1, FILE DEMO 2 and FILE DEMO 3. You might enter FILE DEMO 2 as the filename you wish to select for copying, the other files will then remain uncopied. However you could enter FILE= or F= or even =DEM= to have all three files copied onto a backup disk. The 'wildcard' character = will be taken by the Apple as equivalent to any character or set of characters that lie in a file name at its position. Copying all the files present on a disk will occur if = is entered by itself. Whenever you use the wildcard character in any specification of the filename(s) you want to copy you have to respond to the question DO YOU WANT PROMPTING? with either a Y or a N. If you answer Y then you will be presented with each filename and required either to confirm its transfer with Y or to cancel it with N.

Copy protection

COPYA or FID will allow you to keep backups of all the programs you write yourself, all your data files and many of the programs available commercially. However, to prevent illegal distribution of copies of the software they write, it is now common practice for Apple software authors to incorporate some kind of copy protection into their programs. Copy protection usually involves some kind of non-standard way of arranging the magnetic information on the disk. There are three ways of providing a security backup:

(*a*) Software producers will often provide an 'official' backup disk to the officially registered purchaser of the software.

(*b*) In addition to COPYA and FID there are many other copy programs available — one well-known example is called 'Locksmith'. Such programs are able to spot the tricks the software author has used to copy protect his programs and to overcome them. As soon as a copy program is released, new copy protection techniques are introduced by the software authors. If you buy such a copy program it may well copy some software that your ordinary copiers can't, but there's a good chance it won't copy everything.

(*c*) It's possible to buy a peripheral card to plug into your Apple that provides facilities for backing up copy-protected software. Once the card is installed in the Apple, the software to be backed up is loaded and run as normal. The copy card in effect holds a duplicate image of the contents of the Apple's memory and this duplicate image can be simply dumped onto a disk from the card. This stored chunk of memory contents can be loaded back into memory from the disk at any time; so although the 'backup' disk doesn't in fact hold exactly the same information as the original the effects are the same.

Using BASIC programs

Applesoft and Integer BASIC program files are the easiest to use of the file types. They can be brought into the Apple's memory and used with the LOAD or the RUN commands that DOS provides. Make sure the System Master disk is in Drive 1. Enter LOAD HELLO and the drive light will come on briefly and then stop. Nothing else happens and the flashing cursor comes back. The DOS command LOAD simply places a program in memory, the program is not executed. To execute the loaded program you next enter the BASIC command RUN. DOS in fact supplies a simpler way of doing the whole thing. Simply entering RUN followed by the name of a program file causes the program to be loaded *and* executed in one step. So entering RUN HELLO causes the System Master disk's introductory program to LOAD and RUN. If you have written a BASIC program it is of course important to be able to store it onto a disk. First let's write a program! Enter NEW to clear Apple's memory and then enter the following program lines

```
10  PRINT "HELLO!"
20  PRINT
30  PRINT "THIS IS A SAMPLE PROGRAM"
40  END
```

When you have entered the program, first of all RUN it to see what it does. Now replace the System Master disk in Drive 1 by your workdisk. Use the DOS command **SAVE** to store the program by entering SAVE SAMPLE ... when the disk drive stops whirring and the flashing cursor comes back CATALOG the disk. You should find an entry in the CATALOG like this

```
A 002 SAMPLE
```

To check that the program has been stored, enter NEW to clear the Apple's memory — if you now try to RUN the program nothing will happen. The program isn't there. Next enter RUN SAMPLE and the program should load back in from disk and run as before.

Using binary files

Binary files can be easily spotted on the CATALOG by the letter B in front of them. These files are straightforward copies of the contents of particular sections of the Apple's memory. The Apple's memory can be thought of as 65,536 (1K=1024 so 64K=65,536) pigeonholes each containing a value corresponding to one of the 256 characters the Apple can generate. As you do things with the Apple, the contents of many of these pigeonholes change. Whenever you type characters in at the keyboard a particular area of memory has new values poked into it. As you draw a high resolution graphics image with the graphics tablet or by using Applesoft's graphics statements a different area of the Apple's memory is altered. It's very convenient to be able to store a 'snapshot' of memory on disk for later retrieval.

(a) *Saving and loading picture files* The Apple has two areas of memory that it uses for high resolution graphics images, Page 1 and Page 2. Any picture that you create on either of these pages can be easily saved as a disk file. First let's make a picture to experiment with. Enter and RUN the following program. As you will see, this program just produces a simple graph and leaves you able to see what you are entering at the bottom of the screen.

```
10  REM *** GRAPHER ***
20  TEXT : HOME : REM CLEARS TEXT SCREEN
30  HGR : REM HIRES PAGE 1
40  HCOLOR = 3 : REM CHOOSE WHITE
50  VTAB 22 : REM CURSOR DOWN SCREEN
60  HPLOT 0,0 TO 279,0 TO 279,159
    TO 0,159 TO 0,0 : REM DRAWS FRAME
70  HPLOT 0,55 TO 69,145 TO 139,25
    TO 209,115 TO 279,35: REM DRAWS GRAPH
```

To store any section of memory we have to know at which memory locations the section begins *and* how many locations there are in the section to be stored.

The information making up any Page 1 high resolution picture is located in the Apple's memory as an 8K block (8192 locations) starting at location 8192. To store the picture on the disk as GRAPH you should now enter the direct command BSAVE GRAPH,A8192,L8192 — of course you can use any other name you want for the file and save it onto a different drive if you wish. Enter BSAVE PICTURE,A8192,L8192,D2 to save it under the name PICTURE on Drive 2. A graphics image created in the Page 2 high resolution area of memory would be saved with BSAVE GRAPH,A16384,L8192 since Page 2 graphics start at location 16384. Once a picture is stored on the disk as a binary file it can be loaded back into memory at any time — either directly, or as part of a BASIC program. Let's do it! Enter NEW to clear Apple's memory. Enter TEXT to get out of the graphics mode. Now enter and RUN the following program:

```
100  REM *** PICTURE VIEWER ***
110  ONERR GO TO 900: REM IN CASE OF ERROR
120  HOME : VTAB2
130  INPUT "ENTER PICTURE NAME ";BF$
     : REM WHAT NAME DID YOU SAVE IT UNDER?
140  HGR : REM PAGE 1
150  HCOLOR = 3: REM WHITE
160  PRINT CHR$(4);"BLOAD ";BF$
     : REM DOS COMMAND TO LOAD PICTURE FILE
170  VTAB 22 : REM MOVE TEXT CURSOR DOWN
180  PRINT "PRESS RETURN TO WIPE SCREEN"
190  GET S$ : IF S$ <> CHR$(13) THEN GO TO 190
200  TEXT: REM LEAVE GRAPHICS
210  PRINT "ANOTHER PICTURE ? Y/N ";
220  GET S$ : IF S$ = "Y" THEN PRINT : GO TO 120
230  HOME : END
900  REM *** SIMPLE ERROR TRAP ***
910  TEXT : HOME : VTAB 12 : HTAB12
920  EC = PEEK(222) : IF EC <> 6 THEN
     PRINT "UNKNOWN ERROR"
     : POKE 216,0 : END
930  FLASH : PRINT "NO SUCH PICTURE" : NORMAL
940  FOR DL = 1 TO 3500: NEXT : CALL -3288 : GO TO 100
```

You may have bought some software that displays high resolution graphics screens when it runs. If you have, CATALOG the disk containing the software. If you see a binary file which is 034 sectors in size the chances are that it is a picture! Note the file's name and use PICTURE VIEWER to have a look at it. If you want a quick way to look at a picture you know is on the disk you can also get to it by using direct commands. If the file were called PICTURE then you would enter **HGR** and then next enter **BLOAD PICTURE** — this will successfully reload and display any picture that was originally saved from high resolution Page 1. If nothing seems to happen even though the disk drive whirrs away for quite some time it could be that the picture was originally saved from Page 2 and it has automatically loaded back to the location in memory where it came from — you can't see it because you are looking at Page 1! When the disk stops working enter BLOAD PICTURE,A8192 ... although the file 'knows' it should load at location 16384 which is the beginning of Page 2, the command you have entered forces it to load at location 8192 which is the beginning of Page 1.

(*b*) *Saving and loading a text screen* Sometimes it's useful to store the contents of an ordinary text display onto disk. For example, you can store a series of 'help pages' that your program users can call up from the disk if they get into trouble. These pages shouldn't often be needed and it would be wasteful to use up memory having them as a permanent part of the program. Saving a text display is similar to saving pictures. Press CTRL–RESET to make sure you're in the text mode and type in any series of characters and lines that you fancy. Enter BSAVE TEXT,A1024,L1024 to save a snapshot of the text display on the screen now. The disk drive will operate for a few seconds and the flashing cursor will return. Enter HOME to clear the screen and prove that you really did take a snapshot of the screen by loading it back into memory with BLOAD TEXT ... the following program lets you create on disk a number of 'help pages' that you can retrieve at will.

When you RUN this program you will be prompted with the title for each page. Use the space bar, RETURN key and LEFT ARROW to move around the screen to the point where you want to write text. Press ESCAPE to save a page and CTRL–Q to leave the program.

```
100  REM *** HELP PAGER ***
110  P = 1 : REM SETS FIRST PAGE NUMBER
120  HOME : VTAB2
130  FOR I = 1 TO 40 : PRINT "–"; : NEXT
140  INVERSE : PRINT "HELP PAGE ";P : NORMAL
150  FOR I = 1 TO 40 : PRINT "–"; : NEXT
160  IF PEEK(37) > 22 THEN POKE 37,3
170  GET S$ : IF S$ = CHR$(17) THEN HOME: END
     : REM CHECKS FOR CTRL–Q
180  IF S$ = CHR$(27) THEN GO TO 200
190  PRINT S$; : GO TO 160
200  PRINT
210  PRINT CHR$(4);"BSAVE HELP PAGE ";P;",A1024,L1024"
220  P = P + 1
230  GO TO 120
```

A program wanting to retrieve any of the help pages you have saved would use lines such as

```
200  INPUT "WHICH HELP PAGE ? ";P
210  PRINT CHR$(4);"BLOAD HELP PAGE ";P
```

Using stored text screens in this way can occasionally cause problems with disk access in other parts of a program. This is because the block of memory that contains the screen information also has, hidden away inside it, a number of important values used by DOS. If you do have this problem then solutions such as those published in *Windfall* magazine's August 1983 issue may be of interest.

(c) *Machine code programs* We have seen how we can BSAVE and BLOAD sections of memory that represent 'snapshots' of the Apple's screen display. Many of the binary files listed when you CATALOG your disks will be machine code programs or subroutines that have to be BRUN or BLOADed and CALLed from a BASIC program. Don't expect pictures if you use these!

Using text files

(a) *Opening files* All data file handling using Apple DOS involves certain fundamental steps. Before data can be stored in a file, or retrieved from it, the file has to be 'opened'. We use the DOS command **OPEN** to create a new file or to use one that has been created previously. The program line PRINT CHR$(4);"OPEN NAMES" opens a disk file called NAMES so that the file can be used in some way. Notice how the DOS command is used as part of a PRINT statement which begins with a **CTRL—D** character, **CHR$(4)**. Once opened, the file remains open until we send the DOS command to **CLOSE** it. Under normal conditions DOS will let you keep up to 16 files open at any one time. An OPEN command can be followed by any number of **READ** or **WRITE** commands.

(b) *Writing to files* WRITE creates the conditions that enable you to store data on disk — to actually store the data it must be sent to the disk as part of a **PRINT** statement following the issue of OPEN and WRITE commands. The following program shows how string constants, numbers, string variables and numeric variables can be PRINTed onto the disk.

```
100  REM *** WRITE SIMPLE FILE ***
110  D$ = CHR$(4) : REM CTRL-D
120  A$ = "CAT" : B$ = "DOG" : C$ = " HORSE"
130  A = 1 : B = 2 : C = 3
140  PRINT D$;"OPEN ANIMALS"
150  PRINT D$;"WRITE ANIMALS"
160  PRINT "ANIMALS"
170  PRINT 3
180  PRINT A$ : PRINT A
190  PRINT B$ : PRINT B
200  PRINT C$ : PRINT C
210  PRINT D$;"CLOSE ANIMALS"
```

Notice that in Line 210 the DOS command CLOSE tidies up the file handling operation by closing the ANIMALS file just opened. When it is desired to close a single file of several that have been opened the filename should be included in the DOS command, as in this case. An alternative PRINT D$;"CLOSE" would have functioned just as well in this program — and, at the same time, closed down any other files accidentally left open. Avoiding leaving files unclosed is important because as soon as the Apple tries to open one too many files the error message NO BUFFERS AVAILABLE will be received and your program is likely to crash.

(c) *See what's going on* Apple DOS uses **MON** and **NOMON** to provide a way of 'monitoring' what communications are going on between the computer and the disks. Add the following lines to the program you have just entered and then RUN the program

```
135  PRINT D$;"MON C,I,O"
205  PRINT D$;"NOMON C,I,O"
```

Line 135 turns screen monitoring of DOS communications on and Line 205 turns it off. C, I and O refer to 'control', 'input' and 'output' respectively and any combination of these can be specified. Add these lines into any of the file handling programs you use in order to see what is being passed to and from the disk.

(d) *Reading the files* It is most important to recognize that files can only be read if they have been written to . . . and that the data comes out of the file in the pattern it went in! This means that you, the programmer, need to make sure that somehow or other your file reading programs reflect the structure of the data storage that your file writing programs have established. Enter NEW to clear memory and then enter and RUN the next program. Notice carefully how the file ANIMALS is opened in Line 120 and then Line 130 indicates that the file is soon to be 'read' from. The **INPUT** statement is the means by which data is actually recovered from the disk.

```
100  REM *** READS SIMPLE FILE ***
110  D$ = CHR$(4) : REM CTRL-D
120  PRINT D$;"OPEN ANIMALS"
130  PRINT D$;"READ ANIMALS"
140  INPUT FI$
150  INPUT F
160  FOR I = 1 TO F
170  INPUT AN$(I) : INPUT AN(I)
180  NEXT I
190  PRINT D$;"CLOSE"
200  HOME : VTAB2
210  PRINT "IN THE ";FI$;" FILE"
220  PRINT "THERE ARE ";F;" ANIMALS"
230  PRINT : PRINT
240  FOR I = 1 TO F : PRINT AN(I),AN$(I) : NEXT
```

Notice the following important points:

(1) The quantity, type and order of data items retrieved from the file ANIMALS in Lines 140–180 are exactly the same as those stored in the file by Lines 160–200 of the previous program.

(2) The variable names assigned to the data values read from the file do NOT have to be the same as those used to store them.

(3) Line 150 inputs a numeric value indicating how many sets of data are stored in the rest of the data file. Having been INPUT (in this case as the variable F), this value is then used to control how many times the computer passes around the data input loop contained in Lines 160–180. Keeping within the file itself the information needed to control any program reading from the file is a very useful technique.

(4) All the data required must be INPUT from the file before any PRINT statements are used to send output to the screen or printer. PRINT statements sent before the file has been read and closed can cause problems.

(e) *Sequential and random* In the previous simple examples our file has consisted of a list of strings and/or numbers that are sent to the disk by WRITE/PRINT statement combinations and retrieved from the disk by READ/INPUT statements. In reading such a file it is necessary to start at the beginning and work through towards the end until you find what you want. Files in which the data has to be read 'in sequence' like this are called *sequential files*.

For a file of three animals this is fine, but suppose your file consists of hundreds of names and addresses and your retrieval program needs to find just one of them in order to read or amend it! In such a case it is an advantage to be able to tell the computer exactly where the required 'record' is in the file and for the computer to direct disk reading operations straight to the appropriate part of the file without having to read all the data before it in the file. This involves specifying how long each record is and which position in the file the record is. *Random access files* use records of fixed length and allow direct access to a particular record.

Using random access files allows for much faster data storage and retrieval if files are of medium to large size. The following program doesn't create a large file and therefore only demonstrates the techniques of random access file use, not their advantages. The program demonstrates how you can use random access files to get to a particular record quickly. However, there is a disadvantage! Line 115 of the program instructs the Apple to make the records in the file 8 characters long ... PRINT D$;"OPEN RANDOM ANIMALS,L8" Lines 120 and 270 instruct the Apple to write or read the record specified by the numerical value specified at the end of the DOS command.

```
100  REM *** RANDOM ANIMALS ***
110  D$ = CHR$(4) : REM CTRL-D
115  PRINT D$;"OPEN RANDOM ANIMALS,L8"
120  PRINT D$;"WRITE RANDOM ANIMALS,R0"
125  READ N : REM NOT A DOS COMMAND
     – 'READS' FROM DATA LINE 900
130  PRINT N
135  FOR R = 1 TO N
140  PRINT D$;"WRITE RANDOM ANIMALS,R";R
145  READ AN$ : REM NOT A DOS COMMAND
     – 'READS' FROM DATA LINES 910–950
150  PRINT AN$
155  NEXT R
160  PRINT D$;"CLOSE"
200  HOME : VTAB2
205  PRINT "THERE ARE ";N;" RECORDS"
210  PRINT "ENTER RECORD NUMBER "
220  PRINT "OR PRESS RETURN TO QUIT"
230  PRINT
240  INPUT "READ WHICH RECORD ? ";R$ : R = VAL (R$)
245  IF R$ = "" THEN END: REM CHECKS FOR RETURN
250  IF R<1 OR R>N THEN FLASH : PRINT "NO SUCH RECORD"
     : NORMAL : GOTO 240
260  PRINT D$;"OPEN RANDOM ANIMALS,L8"
270  PRINT D$;"READ RANDOM ANIMALS,R";R
280  INPUT AN$
290  PRINT D$;"CLOSE"
300  HOME : VTAB2
305  INVERSE : PRINT "RECORD ";R : NORMAL : PRINT
310  PRINT "THE ANIMAL IS ";AN$
320  FOR DL = 1 TO 2000 : NEXT : GO TO 200
900  DATA 5
910  DATA "CAT"
920  DATA "DOG"
930  DATA "EAGLE"
940  DATA "HORSE"
950  DATA "COW"
```

How efficiently was the disk's storage space used? The longest name stored, EAGLE, took up only five of the eight character spaces allocated to it in Line 115 — the other spaces are just wasted.

Perhaps we should have made the record length six characters? If we had, then it wouldn't be possible to fit ELEPHANT or MONGOOSE into a record. Even as it is, our file can't accept RHINOCEROS without its last few characters spilling over into the following record. This is a fundamental problem associated with the use of random access files — the record length has to be large enough to accommodate the longest intended entry. Wherever records are shorter than this maximum length the difference between them will just be wasted space. Lack of economy in the disk storage of data is the price paid for improved speed of access.

(f) File structure In a sequential file, various data items are stored one after the other separated by RETURN characters, which are inserted when the file is created, to indicate the end of each item. Within each record of a random access file there may be a number of subdivisions corresponding to different items of data. Each such subdivision is referred to as a field. So, for example, a random access **file** called MAILSHOT may be composed of a number of records, each of which contains a name and address. Each **record** would probably consist of **fields** each holding an item such as a person's name, street name, town name, county or postcode. Within the record DOS separates the fields by a RETURN character to indicate the end of each item. This means in effect that each record within a random access file acts like a mini-sequential file. Although the record itself has a maximum length, the relative lengths of the fields within it are not fixed — the only constraint upon what goes into a particular record is that the combined length of the constituent fields (including the RETURN characters that DOS inserts between them) does not exceed the length allocated to the record. Now add

```
915 DATA "SYLVESTER"
925 DATA "ROVER"
935 DATA "GOLDIE"
945 DATA "RED RUM"
955 DATA "DAISY"
```

Some lines need amending so as to read as follows

```
115  PRINT D$;"OPEN RANDOM ANIMALS,L16"
145  READ AN$,N$ : REM NOT A DOS COMMAND
     - READS DATA LINES
150  PRINT AN$ : PRINT N$
260  PRINT D$;"READ RANDOM ANIMALS, L 16"
280  INPUT AN$ : INPUT N$
300  PRINT
305  PRINT R
310  FOR I = 1 TO LEN(AN$): INVERSE
     : PRINT MID$ (AN$,I,1);: NORMAL
     : PRINT CHR$(32);:NEXT : INVERSE
     : PRINT ",";:NORMAL:PRINT CHR$(32);
311  FOR I = 1 TO LEN(N$): INVERSE
     : PRINT MID$ (N$,I,1):NORMAL
     : PRINT CHR$(32);:NEXT
312  FOR I = 1 TO 16 - ( LEN(AN$)+LEN(N$)+1 )
     : INVERSE : PRINT CHR$(32);: NORMAL
     : PRINT CHR$(32);: NEXT : PRINT
330  GO TO 210
```

Now RUN the program. Your amendments to Line 145 and the DATA lines cause the records in RANDOM ANIMALS to contain fields of varying length according to the animal and the name given to it. The retrieval and display part of the program illustrates the internal structure of any record you choose to examine. Why don't you alter some of the DATA lines to deliberately make the combined length of an animal and its name greater than the 16 characters that the new Line 115 allows for? If you now RUN the program again and examine the record concerned (and the one immediately after it) you will be able to observe the effects that writing too long a record will have on the integrity of the data you store.

Applesoft libraries

Applesoft BASIC is a development of the Microsoft BASIC used on a number of microcomputers. This section provides an opportunity to see Applesoft BASIC being used in a variety of applications and it is hoped that, by being exposed to a wide range of Applesoft statements and commands, you will best get to grips with the language. Libraries of routines will be developed covering text formatting, graphics and file handling. These sections of the guide are based around 'hands-on' tutorial sessions in which, by following the material presented, you will be able to build up libraries of Applesoft routines. These routines may be appended to your own programs and called upon by them. When you are familiar with the Applesoft and DOS statements and commands they use, you can dispose of them and so save memory and processing time. Having booted up the system from your work disk in Drive 1 enter NEW and then type in the following program which you should *NOT* save on your disk.

```
10  REM *** EXECUTIVE ***
15  HOME : INPUT "FILENAME ? ";FI$
20  PRINT CHR$(4);"OPEN ";FI$
25  PRINT CHR$(4);"DELETE ";FI$
30  PRINT CHR$(4);"OPEN ";FI$
35  PRINT CHR$(4);"WRITE ";FI$
40  POKE 33,30
45  LIST 10,70 : REM LINES TO BECOME AN EXEC FILE
50  PRINT CHR$(4);"CLOSE"
55  POKE 33,40
60  PRINT "EXECUTIVE REMOVED"
65  PRINT "ENTER 'EXEC EXECUTIVE' TO RECOVER"
70  DEL 10,70
```

When you have finished typing the program in enter RUN and you will be prompted by the question FILENAME? ... you should enter EXECUTIVE in response. The program stores itself as a text file on disk and can be recovered using the DOS command **EXEC**.

Format library

The Apple's screen is normally one of 24 lines, each of 40 character columns. As characters are sent from the computer to the TV screen the display is built up line by line from the very top left. When the screen is full the whole text display scrolls upwards one line to make more room for more text to be added at the bottom. There are very few applications in which uncontrolled scrolling output of this kind will produce acceptable results. A far more attractive and communicative effect will be achieved if **paged format** is implemented in your programs. The idea is to present a 'page' of the program with information and prompts for input carefully positioned on the screen for maximum visual impact and clarity. When the user has interacted with this page the screen is cleared and the next page is presented. A number of statements unique to Apple BASIC can be involved in the provision of effective paged formatting. Boot up the Apple from the workdisk you have prepared and enter NEW. Enter EXEC EXECUTIVE and for a few seconds the disk drive will operate. When it stops enter LIST and see what you have 'executed'. Now enter the following program lines carefully:

```
1000  HOME : RETURN : REM CLEAR SCREEN
1010  INVERSE : RETURN : REM REVERSE VIDEO
1020  NORMAL : RETURN : REM NORMAL VIDEO
1030  FLASH : RETURN : REM FLASHING VIDEO
1100  PRINT ST$ : RETURN : REM PRINT STRING & LINEFEED
1110  PRINT ST$;: RETURN : REM STRING WITHOUT
      LINEFEED
1200  FOR UL = 1 TO 40 : PRINT "-";: NEXT UL : RETURN :
      REM UNDERLINE
```

Amend Line 45 in the EXECUTIVE lines to read

```
45  LIST 1000,1999
```

and then enter RUN. When prompted for FILENAME? enter FORMAT LIBRARY and all the program lines in the range mentioned in Line 45 will be filed away on the disk.

To append library routines to a program, type in EXEC followed by the library name and press RETURN. The whole library will load from disk and you should then delete any of the library's routines that you don't want by entering their line numbers and pressing RETURN.

(a) *Clearing the screen* Wherever the cursor is positioned on the screen, when you enter **HOME** it disappears, the screen clears and the cursor is 'homed' to the top left of the screen. Try it now, enter HOME and see what happens. Remembering that we still have a number of program lines (numbered 1000 to 1200) sitting in the Apple's memory, let's enter some more as follows:

```
100  GOSUB 1000
110  GOSUB 1200
120  GOSUB 1010
130  ST$ = "FORMAT LIBRARY DEMO"
140  GOSUB 1100
150  GOSUB 1020
160  GOSUB 1200
999  END
```

If you don't recognize the BASIC statement **GOSUB** suffice it to say that GOSUB is short for 'go to the subroutine' and that each line number mentioned after GOSUB in this case is the line number of a section of the library you entered earlier. So, Line 100 says 'go to the subroutine at Line 1000 . . .' and the subroutine at Line 1000 happens to be the one that clears the screen. Now enter RUN to see what our demo program does. As you can see, we have a way of presenting a page title at the top of the screen.

A number of other screen clearing statements are available by issuing a **CALL** to built-in subroutines in the Apple's memory. **CALL −958** clears the screen everywhere to the right and below where the cursor is. **CALL −868** clears the rest of just the line that the cursor is on. Both these CALL statements can be used extensively in formatting screen displays.

(b) Positioning the cursor Suppose we hadn't wanted everything to start printing at the top left? Carefully enter the lines listed below.

```
1300  HTAB H : H = 0 : RETURN
1310  HTAB (PEEK(36) + H) : H = 0 : RETURN
1320  VTAB V : V = 0 : RETURN
1330  VTAB (PEEK(37) + V) : V = 0 : RETURN
```

HTAB and **VTAB**, followed by an appropriate value, allow you to put the cursor anywhere on the screen. The Applesoft statement **PEEK** lets you look into any of the memory locations in the Apple. It so happens that memory locations 36 and 37 contain the computer's record of the present horizontal and vertical position of the cursor—you'll see shortly how we use this information in Lines 1310 and 1330. Enter two more lines into your program:

```
105  V = 2 : GOSUB 1320
165  V = 4 : GOSUB 1330
```

and RUN the program again. What has changed? You can see that the whole title frame has moved down one line from the top and the cursor ends up four lines further down that it did! Can you see how the lines we most recently added caused this effect? Program Line 105 sets the value of V as 2 and then goes off to the subroutine at Line 1320. This subroutine uses the VTAB statement to put the cursor on screen line V . . . in this case on Line 2. Program Line 165 sets the value of V as 4 and then goes off to the subroutine at Line 1330. This subroutine adds the value of V onto whatever value is already present in memory location 37 and then puts the cursor on the screen line that results . . . so this routine puts the cursor V lines away from wherever it was before! Check for yourself what effect the values for V in these lines have by changing program Lines 105 and 165 to read

```
105  V = 10 : GOSUB 1320
165  V = -5 : GOSUB 1330
```

RUN the program again and see what happens. Experiment a little by putting in values of your own for V — but remember that there are only 24 lines on the screen. If your instructions result in the Apple trying to put the cursor above Line 1 or below Line 24 you'll be in trouble! So far we have been controlling the vertical position of the cursor. The subroutines at 1300 and 1310 are used to control the horizontal position of the cursor in just the same way. You simply set a value of H before going to the subroutine, instead of setting V. The text screen has forty horizontal positions. Experiment a little with it. When you have finished experimenting with this horizontal cursor control carefully enter the following program line

```
1900  GOSUB 1000 : V = 2 : GOSUB 1320 : GOSUB 1200
      : GOSUB 1010 : GOSUB 1100 : GOSUB 1020 : GOSUB 1200
      : V = 2 : GOSUB 1330 : RETURN
```

This multiple statement line is a subroutine made up of other subroutines. You will be able to use it repeatedly to provide a title frame for your 'pages'. Since it is in fact a condensed version of several of the program lines you previously entered you should now get rid of those lines. To do this enter DEL 100,999 ... if you now enter LIST the program lines in memory will scroll up on the screen. These lines represent the present state of development of your text formatting library. Recover your important EXECUTIVE file by entering EXEC EXECUTIVE. Amend Line 45 so that it reads LIST 1000,1999 and then enter RUN ... store the library on disk under FORMAT LIBRARY.

Some other interesting statements related to the cursor are CALL −998, CALL −922, CALL −1008 and CALL −1036. These move the cursor one place up, down, left and right respectively. Can you see how they might be used in place of VTAB and HTAB in the library?

(c) *Text windows* The Apple text screen can be reduced in size temporarily so that, for example, titles and input prompts may remain unaltered on the screen while other parts of the text display change. Add the following program lines to the FORMAT LIBRARY.

```
1400  POKE 32,LM : LM = 0 : GOSUB 1450 : RETURN
1410  POKE 33,WD : WD = 40 - PEEK (32)
      : GOSUB 1450 : RETURN
1420  POKE 34,TM : TM = 0 : GOSUB 1450 : RETURN
1430  POKE 35, ABS(PEEK(34)-24) - BM
      : BM = ABS(PEEK(34)-24)
      : GOSUB 1450 : RETURN
1440  POKE 32,0 : POKE 33,40 : POKE 34,0 : POKE 35,24
      : GOSUB 1450 : RETURN
1450  PRINT CHR$(1):RETURN
```

You will remember that the statement PEEK allows us to look into memory locations to see what's there . . . the program below uses **POKE** to put new values into locations 32 to 35 in the Apple's memory which contain information about the present settings for left margin, screen width, top margin and bottom margin respectively.

```
100  ST$ = "TEXT WINDOW DEMO" : GOSUB 1900
120  LM = 10 : GOSUB 1400
130  WD = 20 : GOSUB 1410
140  TM = 5 : V = TM : GOSUB 1320 : GOSUB 1420
150  BM = 10 : GOSUB 1430
200  FOR I = 1 TO 500 : PRINT I;" ";: NEXT I
210  GOSUB 1440 : V = 8 : GOSUB 1320
220  TM = 4 : GOSUB 1420 : GOSUB 1000 : GOSUB 1010
230  PRINT "DID IT WORK?"
240  V = 3 : GOSUB 1330 : GOSUB 1010
250  PRINT "CHANGE THE VALUES"
260  PRINT "IN LINES 120-150"
270  PRINT "TO VARY TEXT WINDOW"
280  GOSUB 1020 : V = 3 : GOSUB 1330
290  LIST120,150 : GOSUB 1440
999  END
```

By altering the values in Lines 120–150 you can set up a text window within which all output to the screen will be confined. Remember that the whole screen is still only 24 lines deep by 40 columns wide; so if you enter nonsense values for LM, WD, TM and/or BM you'll get a nonsense text window! When you have finished experimenting you can get rid of the non-library lines of this program by entering DEL 100,999.

(*d*) *Tidy data input* To get a neat layout for our page format we also need to have ways of controlling the position of prompts for input within whatever text window is defined. Add the following lines to your FORMAT LIBRARY and store it away on disk using EXECUTIVE.

```
1500  GOSUB 1300 : GOSUB 1320 : GOSUB 1010 : GOSUB 1110
      : GOSUB 1020 : PRINT SPC(BL - LEN(ST$));: RETURN
      : REM SETS POSITION
1505  FOR J=1 TO RL : PRINT "-";: NEXT : H = 0 - RL
      : GOSUB 1310 : RETURN
1510  INPUT IS$ : I$=LEFT$(I$,RL) : RETURN
1520  GOSUB 1500 : GOSUB 1505 : GOSUB 1510 : RETURN
```

Enter the following program lines to demonstrate how we might use these new extensions to the library:

```
100  ST$ = "PAGED INPUT DEMO"
105  GOSUB 1900
110  V = 8 : H = 5 : ST$ = "YOUR SURNAME" : BL = 15
115  RL=20 : GOSUB 1520 : I$(1) = I$
120  V = 10 : H = 5 : ST$ = "YOUR FORENAME" : BL = 15
125  RL=15 : GOSUB 1520 : I$(2) = I$
130  V = 12 : H = 1 : ST$ = "AGE" : BL = 5
135  RL = 3 : GOSUB 1520 : I$(3) = I$
140  V = 12 : H = 10 : ST$ = "SEX M/F" : BL = 10
145  RL = 1 : GOSUB 1520 : I$(4) = I$
150  V = 12 : H = 22 : ST$ = "JOB" : BL = 5
155  RL=15 : GOSUB 1520 : I$(5) = I$
999  END
```

Now RUN the program and observe how the program lines control the positioning of the input prompts on the screen. It is important to remember that you can vary the main control section of your program (lines below 1000 in all our examples) in whatever way you wish. For instance the effects achieved by the program you have presently in memory could be replicated by changing the control module to use information in **READ** and **DATA** lines to format the screen.

```
100  ST$ = "PAGED INPUT DEMO"
105  GOSUB 1900
110  RESTORE : READ N : REM 'RESTORES' DATA & READS N
115  DIM I$(N) : REM SETS ARRAY SIZE FOR N INPUTS
120  FOR I = 1 TO N
130  READ V , H , ST$ , BL , RL
135  REM VTAB , HTAB , PROMPT , BOX , RESPONSE
140  GOSUB 1520
150  I$(I) = I$
160  NEXT I
900  DATA 5 : REM N = 5 FIVE INPUT BOXES
910  DATA 8 , 5 , "YOUR SURNAME" , 15, 20
920  DATA 10, 5 , "YOUR FORENAME" , 15, 15
930  DATA 12, 1, "AGE", 5, 3
940  DATA 12, 10, "SEX M/F", 10, 1
950  DATA 12, 22, "JOB", 5, 15
999  END
```

While at first sight this version appears more complicated than the first version, it has considerable advantage over it. Extra input boxes can be very easily catered for by changing the value for N in Line 900 and adding DATA line(s) numbered 910 and above. Any alterations to the data input format are very straightforward to implement by editing the particular values in these DATA statements. Notice the Applesoft statement **DIM** is used to reserve space for the array I$(*), according to the value of N stored in Line 900. Before leaving this section make sure your entire FORMAT LIBRARY is stored on disk using EXECUTIVE.

Graphics library

Applesoft high resolution graphics are easy to use from within BASIC programs and this section provides some simple routines to demonstrate how to do it. Enter NEW to clear memory and enter the following lines

```
3000  HGR : RETURN : REM CLEARS & ACCESSES PAGE 1
3010  HGR2 : RETURN : REM CLEARS & ACCESSES PAGE 2
3020  POKE -16304,0 : POKE -16297,0 : POKE -16300,0
      : REM ACCESSES PAGE 1
3030  POKE -16304,0 : POKE -16297,0 : POKE -16299,0
      : REM ACCESSES PAGE 2
3040  HOME : TEXT : RETURN
      : REM CLEARS TEXT SCREEN AND RETURNS TO IT
3050  HCOLOR = CO : RETURN
      : REM SETS THE COLOUR FOR PLOTTING- 0 TO 7
```

The Apple can store two high resolution pictures at any one time — Page 1 and Page 2. The lines above allow you to control access to these picture pages. Lines 3000 and 3010 put graphics pages on your screen and clear them of whatever they have on them at the time. Any picture previously there will disappear before your eyes and your program can start drawing! Lines 3020 and 3030 also get into your graphics pages but this time any picture you have on these pages will *not* be cleared. You can keep a picture in memory and gradually work on it as your program acquires more and more data. Graphics produced on these graphics pages consist of dots. The picture you see is made up of lines and patches which are themselves made up of dots. Every dot is a spot of light produced at a particular horizontal (X) and vertical (Y) position and there are 53760 such positions on the Apple's screen. The library routines use the fundamental high resolution statements **HGR**, **HCOLOR** and **HPLOT** to perform commonly required operations. They require the control module of any program using them to pass the X,Y co-ordinate information needed to perform a particular task.

Add the following lines to your library and then — *having amended Line 45 to read LIST 3000,3999* — use your EXECUTIVE program to save them as a disk file named GRAPHICS LIBRARY.

```
3100  HPLOT XP,YP : RETURN : REM PLOTS A POINT
3105  REM PLOTS A BLOB - POINTS ALL AROUND A POINT
3106  IF YP > 190 OR YP < 1 OR XP > 278 OR XP < 1
      THEN GO TO 3109
3107  FOR PL = YP-1 TO YP+1 : HPLOT XP-1,PL
      : HPLOT XP,PL : HPLOT XP+1,PL : NEXT
3108  FOR PL = XP-1 TO XP+1 : HPLOT PL,YP-1
      : HPLOT PL,YP : HPLOT PL,YP+1 : NEXT
3109  RETURN
3110  HPLOT XL,YL TO XH,YH : RETURN : REM PLOTS A LINE
3120  HPLOT XL,YL TO XL,YH TO XH,YH TO XH,YL TO XL,YL
      : RETURN : REM DRAWS A FRAME
3130  FOR PL = YL TO YH : HPLOT XL,PL TO XH,PL : NEXT
      : RETURN : REM FILLS A 'BOX'
3200  REM DRAWS A KIND OF CIRCLE
3201  FOR PL = (0-R) TO R
3202  XD = ((R*R)-(PL*PL))^0.5
3203  XD = XP + XD : YD = YP + PL
3204  HPLOT XP,YP TO XD,YD
3205  XD = (((R*R)-(PL*PL))^0.5)*(-1)
3206  XD = XP + XD : YD = YP + PL
3207  HPLOT XP,YP TO XD,YD
3208  NEXT PL
3209  RETURN
```

In the routine beginning at Line 3105 the XP,YP co-ordinate pair defined is checked to see if it lies within the range of the screen before going on to plot a 'blob' at the point specified. The Apple's high resolution screen lets you plot points 0 to 279 across the screen and points 0 to 191 down it. In Page 2 graphics all these points can be seen but no text can be printed on the screen. In Page 1 four lines are left at the bottom of the screen which can be used to display text but points with Y values of more than 159 will not be visible.

Add the following program control module to produce a program which takes advantage of Page 1's four text lines to guide you through a demonstration program using some of the library routines.

```
100  GOSUB 3000 : CO = 3 : GOSUB 3050
105  ST$ = "GOSUB 3000 PUTS YOU IN PAGE 1 HIRES"
     : GOSUB 900
110  XL = 50 : YL = 150 : XH = 130 : YH = 30 : GOSUB 3120
115  ST$ = "GOSUB 3120 DRAWS A FRAME" : GOSUB 900
120  XP = 155 : YP = 120 : GOSUB 3100
125  ST$ = "GOSUB 3100 PLOTS A POINT" : GOSUB 900
130  XP = 200 : YP = 50 : GOSUB 3105
135  ST$ = "GOSUB 3105 PLOTS A 'BLOB'" : GOSUB 900
140  XL = 150 : YL = 10 : XH = 250 : YH = 40 : GOSUB 3130
145  ST$ = "GOSUB 3130 FILLS A BOX" : GOSUB 900
150  XP = 100 : YP = 130 : R = 20 : GOSUB 3200
155  ST$ = "GOSUB 3200 FILLS A KIND OF CIRCLE"
     : GOSUB 900 : REM RADIUS = R
160  XL = 110 : YL = 10 : XH = 25 : YH = 115 : GOSUB 3110
165  ST$ = "GOSUB 3110 PLOTS A LINE" : GOSUB 900
210  ST$ = "PROGRAM ENDED" : GOSUB 900
220  HOME : TEXT : END
900  VTAB 21 : CALL -958 : PRINT ST$ : RETURN
910  REM CURSOR TO LAST FOUR LINES,CALL-958 CLEARS
     SCREEN
920  PRINT "PRESS ANY KEY TO CONTINUE "; : GET S$
     : RETURN : REM WAITS TILL YOU ARE READY
```

Now RUN the program and observe its effects. Examine the program lines and observe how, according to the routine being used, values of XL,XH,YL,YH,XP or YP have to be passed from the control module of the program to the library routines. Play about with these values by amending the program lines and see what effects you can produce — notice that XL and YL must normally be lower than XH and YH to get the effect you want — notice that higher values for Y plot lower down on the graphics screen! When you have finished playing with this enter DEL 100,999 to get rid of the existing program control module.

Now let's get down to something useful!

```
3300  REM ROUTINE TO CONSTRUCT SCALED AXES
      FOR GRAPHING DATA
3301  REM NEEDS (XL(X LOW),YL(Y LOW),XH(X HIGH),
      YH(Y HIGH),
      XD(X DIVISIONS),YD(Y DIVISIONS) PASSING TO IT
3302  YE = (INT(150/YD) * YD)
      : YI = 155 - YE * (ABS(YL)/(ABS(YL)+YH))
      : IF YL = > 0 THEN YI = 155
3303  XE = INT (240/XD)*XD : HPLOT 20,YI TO 20 + XE,YI
3304  XI = XE * (ABS(XL)/(ABS(XL)+XH)) : XE = INT(240/XD)
      : IF XL = > 0 THEN XI = 0
3305  HPLOT 20+XI,155-YE TO 20+XI,155 : YE = INT(150/YD)
3306  FOR PL = 0 TO XD : XP = 20 + (PL*XE)
      : HPLOT XP,YI-2 TO XP,YI+3 : NEXT PL
3308  FOR PL = 0 TO YD : YP = (155 - (YE*YD)) + (PL*YE)
      : HPLOT 17 + XI,YP TO 22 + XI,YP : NEXT PL
3310  RETURN
3320  XP = ((XP-XL) / (XH-XL) * (XE*XD)) + 20 : RETURN
3330  YP = 155 - (((YP-YL) / (YH -YL)) * YE * YD)
      : RETURN
```

This routine requires you to send it values for the maximum and minimum X and Y values that are to be encountered and (XL,XH,YL,YH) and the number of scale divisions required along each axis (XD and YD). The routine then automatically adjusts the scaling to take best advantage of the screen space available. The routine is quite complicated because the Apple has quite an odd way of numbering its points in the Y dimension — whereas one would normally expect the X,Y point 0,0 to be at the *bottom* left of the display it is in fact at the *top* left on the Apple! Unlike the other routines in this library this routine expects to be passed *values YH that are higher than those of YL*. This routine is worth keeping in your library because there are lots of situations in which being able to graph data in this way can be useful.

The following control module represents one way of using the routine.

```
100  REM *** DATA PLOTTER ***
105  GOSUB 3040 : ST$ = "DATA PLOTTER" : GOSUB 1900
110  V = 6 : GOSUB 1320 : INPUT "MAXIMUM X VALUE ? ";XH
115  INPUT "MINIMUM X VALUE ? ";XL
120  INPUT "X-AXIS DIVISIONS ? ";XD
130  V = 10 : GOSUB 1320 : INPUT "MAXIMUM Y VALUE ?";YH
135  INPUT "MINIMUM Y VALUE ? ";YL
140  INPUT "Y-AXIS DIVISIONS ? "; YD
145  IF YH > YL AND XH > XL AND XD > 0 AND YD > 0
     THEN GO TO 155
150  ST$ = "CAUTION" :GOSUB 1030 :GOSUB 1100
     :GOSUB 1020
     : ST$ = "SCALE ERROR ... ANY KEY TO RESTART"
     : GOSUB 1110 : GET S$ : GO TO 100
160  TM = 20 : GOSUB1420 : LM = 4 : GOSUB1400
170  GOSUB 3000 : CO = 3 : GOSUB3050 : GOSUB3300
200  GOSUB 1000 : INVERSE : PRINT "X";
201  NORMAL : PRINT " ";XL;" TO ";XH;SPC(4);
202  INVERSE : PRINT"Y"; : NORMAL
     : PRINT SPC(2);YL" TO ";YH : PRINT
205  INPUT "X-VALUE ? ";X$ : XP = VAL(X$)
206  IF X$ = "" THEN PRINT "PROGRAM ENDED"
     : PRINT"TYPE IN <RUN> TO START AGAIN" : END
207  GOSUB 3320
210  INPUT "Y-VALUE ? ";Y$ : YP = VAL(Y$)
211  IF Y$ = "" THEN GO TO 200
212  GOSUB 3330
220  GOSUB 3105
230  GO TO 200
```

Having entered this control module you will also need to EXEC into memory your FORMAT LIBRARY since a number of its routines are also called from the program. Try entering different ranges for the scales of the X and Y axes — you will be prompted on the screen to enter these values and told if they don't make any sense. To leave the program press RETURN when being asked for an X value.

File library

Enter NEW to clear memory and then enter the following lines as the first part of our file-handling library. We will build this library up gradually, just as earlier we built up a library of screen formatting routines. Pay particular attention to entering the spaces within quotation marks where they are present and to entering any commas, semicolons and quotation marks in exactly the right places.

```
2000  PRINT CHR$(4);"OPEN ";FI$;",D";D : RETURN
      :REM OPENS DATA FILE FI$ ON DRIVE D
2010  PRINT CHR$(4);"READ ";FI$ : RETURN
      :REM INDICATES THAT FILE IS TO BE READ FROM
2015  PRINT CHR$(4);"WRITE ";FI$ : RETURN
      :REM INDICATES THAT FILE IS TO BE WRITTEN TO
2020  PRINT CHR$(4);"OPEN ";FI$;",L";FL;",D";D : RETURN
      :REM OPENS RANDOM ACCESS FILE FI$ WITH
      RECORDS FL CHARACTERS LONG ON DRIVE D
2030  PRINT CHR$(4);"READ ";FI$;",R";R : RETURN
      :REM INDICATES THAT RECORD R IN THE FILE IS TO
      BE READ FROM
2035  PRINT CHR$(4);"WRITE ";FI$;",R";R : RETURN
      :REM RECORD R IN THE FILE IS TO BE WRITTEN
2040  PRINT CHR$(4);"CLOSE" : RETURN
      :REM CLOSES ALL OPEN FILES
2045  PRINT CHR$(4);"CLOSE ";FI$ : RETURN
      :REM CLOSES FILE FI$
2090  PRINT CHR$(4);"MON C,I,O" : RETURN
      :REM TURNS ON "MONITORING" OF DISK
      COMMUNICATIONS
2095  PRINT CHR$(4);"NOMON C,I,O" : RETURN
      :REM TURNS "MONITORING" OFF
```

Enter EXEC EXECUTIVE and, when the file is loaded, amend Line 45 to read LIST 2000,2999 ... now you are about to work on the FILE LIBRARY all its lines will be in this range. Now RUN the program and store the file as FILE LIBRARY.

(a) *Simple record keeping* The program below creates random access file records containing one field.

```
100  REM*** RECORD MAKER ***
110  INPUT "ENTER A FILENAME "; FI$
120  INPUT "ENTER THE DRIVE NUMBER ";D
130  INPUT "MAXIMUM RECORD SIZE ? ";FL
140  PRINT : PRINT
200  INPUT "ENTER RECORD NUMBER ";R
205  PRINT : IF R < 1 THEN 200
210  PRINT "ENTER AN APPROPRIATE RECORD"
215  PRINT "(NB. DO NOT INCLUDE ANY COMMAS!)"
220  PRINT "OR PRESS RETURN TO QUIT PROGRAM" : PRINT
225  INPUT RC$ : IF LEN(RC$) > FL THEN FLASH
     : PRINT "TOO LONG" : NORMAL : PRINT : GO TO 210
230  IF RC$ = "" THEN PRINT
     : PRINT "FILING FINISHED" : END
235  GOSUB 2090 : REM LET'S SEE WHAT HAPPENS
240  GOSUB 2020 : GOSUB 2035
245  PRINT RC$
250  GOSUB 2040
255  REM ABOVE LINES OPEN FILE, STORE RECORD, CLOSE
     FILE
260  GOSUB 2000 : GOSUB 2015 : PRINT FL:GOSUB 2040
265  REM ABOVE LINE STORES RECORD LENGTH
     AT BEGINNING OF FILE
270  GOSUB2095 : REM TURNS OFF MONITOR
275  PRINT : PRINT
280  GO TO 200
```

If you run this program now you can see the mechanism of a typical simple file creation routine in operation. Notice that Line 2090 of the file library uses the DOS command MON to let you 'monitor' what goes in and out of your data files. To create records with multiple fields, Line 225 should be replaced by as many INPUTs as necessary and Line 245 by a matching set of PRINT statements. To read from the files created, a program using GOSUBs 2010 and 2030 (instead of 2015 and 2035) would be needed.

(b) A simple database A data file may be created from single field records in which each record is made up of several data items joined together into one long string. Such an approach has the advantage of simplifying the file writing operations involved and providing a general purpose mechanism for creating files of different structures. DATA statements can be used as the means of providing information about the name of the file, its structure and its location. Enter, for a mailing list,

```
900 DATA "MAILSHOT" : REM FILENAME
905 DATA 7 : REM NUMBER OF FIELDS IN RECORD
906 DATA 1 : REM DRIVE NUMBER FOR STORAGE
```

Accompanying DATA statements can contain details of the internal structure of each record, the lengths of each of the items making it up and information of use in paging the sequence used to input the data. Make sure that your values of position and length combine sensibly as in the example below — 'box' is the total length of the prompt and the spaces between it and the cursor.

```
910 REM VTAB,HTAB,PROMPT,BOX,RESPONSE
911 DATA 9,1,"TITLE",11,5
912 DATA 11,1,"SURNAME",11,20
913 DATA 13,1,"FORENAME",11,20
914 DATA 15,1,"ADDRESS 1",11,25
915 DATA 17,1,"ADDRESS 2",11,25
916 DATA 19,1,"ADDRESS 3",11,25
917 DATA 21,1,"POSTCODE",11,10
```

As long as the combined length of the field responses doesn't exceed 255 characters, a file of any structure can be built up by changing the number and content of the DATA lines at Line 910 and upwards. Use EXECUTIVE to store any set of structure-defining DATA lines you create on disk as a file. Amend Line 45 of EXECUTIVE to read LIST 900,999 and when prompted for FILENAME? respond with the database file's name, prefixed by **DATA**. For the example above we would enter DATA.MAILSHOT as the file's name.

Additional lines will be needed in the library in order to make sense of the DATA lines in the programs. Enter the following lines and then use EXECUTIVE to store the whole library (Lines 2000–2999) on disk again.

```
2100  RESTORE : READ FI$,F,D : RETURN
2110  READ V,H,ST$,BL,RL : RETURN
2200  GOSUB 2110 : GOSUB 1520 : RETURN
```

(c) Creating the database To create a program which will allow you to build up a database of your chosen structure, combine the control module below with the FILE LIBRARY and with an appropriate set of DATA statements as illustrated in the previous section. The control module listed below uses routines from the FORMAT LIBRARY and so they should be EXECed back into memory from disk..

```
100  REM *** SIMPLE DATABASE CREATOR ***
110  ST$ = "DATABASE CREATOR" : GOSUB 1900
120  PRINT "NEW FILE ? PRESS Y OR N ";: GET S$
     : PRINT : IF S$ <> "Y" THEN 200
130  GOSUB 2100:GOSUB 2000:GOSUB 2015:PRINT 0
     :GOSUB 2040
200  GOSUB 2100:GOSUB 2000:GOSUB 2010:INPUT R
     :GOSUB 2040
210  ST$ = FI$ : GOSUB 1900
220  R = R + 1 : ST$ = "RECORD " + STR$(R)
     : V = 0 : GOSUB 1330
230  GOSUB 1010 : GOSUB 1100 : GOSUB 1020
240  FL = F : RC$ = "" : FOR L1 = 1 TO F
250  GOSUB 2200 : FL = FL + RL : RC$ = RC$ + I$
260  IF LEN(I$) < RL THEN FOR L2 = 1 TO (RL - LEN(I$))
     : RC$ = RC$ + CHR$(32) : NEXT L2
270  NEXT L1
300  TM = 23 : GOSUB 1420 : GOSUB 2090
310  GOSUB 2020 : GOSUB 2035 : PRINT RC$ : GOSUB 2040
320  GOSUB 2000 : GOSUB 2015 : PRINT R : GOSUB 2040
330  TM = 0 : GOSUB 1420 : GOSUB 2095
340  ST$ = "ENTER ANOTHER RECORD ? Y/N "
350  GOSUB 1900 : V = 2 : GOSUB 1330
360  GET S$ : PRINT : IF S$ = "Y" THEN GO TO 200
370  PRINT : PRINT "ENTER <RUN> TO RESTART" : END
```

(d) *Using the database* To create a program using any database that you have established from use of the previous program, combine the control module below with the FILE LIBRARY, the FORMAT LIBRARY and the DATA lines appropriate to the file by EXECing them back into memory from disk.

```
100  REM *** DATABASE READER ***
110  GOSUB 2100:GOSUB 2000:GOSUB 2010:INPUT N
     :GOSUB 2040
120  ST$= FI$ + "  " + STR$(N) + " RECORDS":GOSUB 1900
130  ST$="WHICH RECORD":V=0:GOSUB1330
140  GOSUB 1010:GOSUB 1110:GOSUB1020
150  INPUT R : IF R <1 OR R > N THEN ST$="NO SUCH
     RECORD":GOSUB 1030:GOSUB 1100:GOSUB 1020:GO TO 150
160  FL=F : GOSUB 2100 : FOR L1=1 TO F : GOSUB 2110
     :FL=FL + RL : NEXT L1
200  GOSUB 2020:GOSUB 2030:INPUT RC$:GOSUB 2040
210  ST$=FI$ + " RECORD "+ STR$(R):GOSUB1900
     :GOSUB 2100
220  P = 1 : FOR L1 = 1 TO F:GOSUB 2110
230  FL$=MID$(RC$,P,RL): REM TAKES RC$ TO BITS
240  GOSUB 1500 : PRINT FL$ : P = P + RL : NEXT L1
300  PRINT : GOSUB 1200
310  ST$="READ ANOTHER RECORD ? Y/N "
320  GOSUB 1100:GOSUB 1200:V=-1:GOSUB 1330
     :H=30:GOSUB 1300
330  GET S$:PRINT:IF S$="Y" THEN GO TO 120
340  PRINT : PRINT "ENTER <RUN> TO RESTART" : END
```

Notice that Line 200 gets a record from the file and information in the DATA statements is used in Line 230 to correctly disassemble the record — **MID$** is used to take a section out of the middle of the record's single field. The section is RL characters long and starts at character position P in the field. Bit by bit the field is disassembled and Line 240 displays each section as it is obtained. Lines 220–240 could easily be adapted so as to select only certain fields for display, to direct output to a printer, to sort alphabetically the records in the file on the basis of the entries in one or more sections of the records. A simple amendment to the database creation program will allow you to change existing records — amend Line 220 to INPUT a value for R instead of simply incrementing the R counter.

Peripheral slots

The Apple can communicate with lots of external devices through interface cards located in its peripheral slots — the slots are numbered 0 to 7. Conventionally Slot 6 is used for the disk drive controller card and Slot 7 for a colour card. Slot 0 is always used for a language card. Peripheral cards are available that will let your Apple do tasks as diverse as digitizing TV pictures, creating electronic music and converting the Apple into a digital storage oscilloscope ... most of these tasks involve the use of custom designed interface cards. Lots of devices can be connected to interface cards using a standard data communications format. For example, many printers and modems can be interfaced with the Apple via an RS232 standard serial interface card. Probably the most widely used peripheral, however, is the low cost dot-matrix printer which accepts output of parallel format data from the Apple through a parallel interface card — the *de facto* standard for such communications is the Centronics standard. Your parallel interface is best placed in Slot 1 because that is where lots of commercially produced software expects to find it.

To send output to the printer in Slot 1 instead of the TV screen the command **PR#1** must be issued — **PR#0** will return output to the screen. Programs using a peripheral device in Slot 2 would use PR#2 to send data to it. If the Apple tries to send output to a slot with no card the system will 'hang' and you will have to RESET the Apple. The commands can be used directly from the keyboard to get program listings and the like but in programs that are using disks they should be sent as DOS commands in the form PRINT CHR$(4);"PR#1". Having entered NEW to clear memory enter the following lines.

```
4000  PRINT CHR$(4);"PR#";SL : RETURN
      : REM SENDS OUTPUT TO SLOT SL
4005  PRINT CHR$(4);"PR#0" :RETURN
      : REM RETURNS OUTPUT TO SCREEN
```

Having opened up communication with a particular slot, things can begin to get complicated! Unlike the TV screen the printer will certainly be able to print in more than 40 columns and the desired column width may have to be communicated to the printer. Many printers using the Centronics standard can accept this information in statements like **CTRL—I80N** which, when sent in a program in the form PRINT CHR$(9);"80N", would be interpreted as a command to set the print width at 80 columns. Unfortunately one printer very commonly used with the Apple, the Epson MX–80 series, doesn't conform to this standard and a POKE statement of the form POKE 1656 + SL,W must be issued ... so, where the printer card is in Slot 1 and you require a 70-column width the statement POKE 1657,70 would have to be sent to the printer.

To add to the complexity, such printers as the Epson often provide a variety of additional facilities such as expanded print, proportional print and underlining. To make use of these features your programs must send the particular sequence of **control characters** that your make of printer is programmed (through its own internal circuitry) to recognize as the signs to act in a certain way. These codes usually consist of one or more of the 'invisible' ASCII codes that we came across earlier. For example, to put your Epson MX–80 into the mode where it doubles the size of the characters it prints, you must send CHR$(14) in front of the string you want to print ... PRINT CHR$(14); "FRED" prints a big version of FRED. To do exactly the same kind of thing on the widely used Centronics 730 series of printers you would have to PRINT CHR$(27);CHR$(14);"FRED". The idea of using 'libraries' of routines appended to your program really comes into its own when we are using printers. The control module of your program can be set up to call a subroutine that gives underlining, or expanded print or whatever. The actual contents of the PRINT LIBRARY will vary according to which printer is to be used.

The following lines might form the basis of an EPSON LIBRARY.

```
4100  PRINT CHR$(14);:RETURN:REM EXPANDED PRINT
4110  PRINT CHR$(20);:RETURN:REM CANCELS EXPANDED
4120  PRINT CHR$(15);:RETURN:REM CONDENSED PRINT
4130  PRINT CHR$(18);:RETURN:REM CANCELS CONDENSED
4140  PRINT CHR$(27);CHR$(69);:RETURN:REM EMPHASIZED
4150  PRINT CHR$(27);CHR$(70);:RETURN
      :REM CANCELS EMPHASIZED
```

In addition to the complexity brought about by the existence of different conventions that are used for controlling printer mode, the particular characteristics of the printer interface card being used may vary. For example, some of the control characters the Epson MX–80 needs may be recognized and acted upon by the printer card itself and never get to the printer. Such a library can only really be developed by reference to the manual accompanying a particular printer and by reference to the characteristics of the interface card being used, but the concept of print statement libraries like this is becoming widely used by software developers. The arrival of the new generation of printers like the Epson FX–80 makes their use very desirable. Such printers offer a huge range of facilities including powerful formatting capability, graphics printing, user definable character sets, multiple typefaces, underlining, superscripts and subscripts all of which have to be initiated by sequences of control characters. Inclusion of these characters time after time with a program would be wasteful of memory and very prone to error. When buying your printer and printer interface equipment make a special point of finding out just what facilities are available — and whether they can be easily accessed from your Applesoft programs!

Many peripheral devices send data *to* the Apple rather than receive from it. To open the Apple up to these devices commands of the form **IN#1** must be used — **IN#0** returns 'input' to the Apple's keyboard.

Software

A very large proportion of the typical Apple user's software needs can probably be catered for by access to a relatively small number o. software 'packages'.

(a) *Spreadsheets* These allow you to use the calculating power of the computer to manipulate the large amounts of numerical data involved in financial modelling and projection. The user's involvement in instructing the computer is limited to the use of a range of standard functions that allow him to enter data into various parts of the sheet and then set up the mathematical relationships that project values into other parts of the sheet from the data entered. The best known spreadsheet package for the Apple is **Visicalc** but a rival package, **Multiplan**, offers some more advanced features such as variable column widths and sorting of entries alphabetically or numerically.

(b) *Database packages* These provide computer-based filing and data retrieval facilities. According to the nature of the data to be stored, the package is used first to create custom-designed data entry 'forms' on the screen that provide robust and user-friendly procedures to allow for the input and checking of data. A variety of different file structures can be created to suit the particular data being handled. Once entered, any individual record can be viewed on the screen and amended as details within it become outdated. The whole file can be periodically rearranged alphabetically, numerically or according to a data and time contained within key fields of the record. A variety of standard 'report' formats can be defined which will give printout of tabular data or labels or index cards using selected fields and records from the data file. There are many well regarded database packages available including **Visifile** (the companion to Visicalc) and an excellent new package, **Quick File II**, is available to Apple //e users. This package provides very speedy data processing and considerable flexibility of input and report configuration.

(c) *Graphics packages* Much of the data contained within spreadsheet files and databases can be used more effectively in graphical rather than numerical form. Packages are available that can accept data from such sources (as well as directly through the keyboard) and plot from it line graphs, bar charts, pie diagrams and scattergrams. **A fundamental limitation of the Apple with normal use of Applesoft is its inability to mix text and graphics.** Graphics packages are usually designed to overcome this limitation and provide you with ways of scaling and labelling any diagrams you produce. **Apple Plot** and **Apple Business Graphics** are good examples of this type of software.

(d) *Word processing packages* In their simplest form, these allow you to use the computer as a typewriter to enter and format letters, reports and other documents for subsequent output on a printer. The principal advantages of using the micro as a word processor are the ease of error correction at the time of entry, the ability to store the document on disk for later retrieval or amendment as required and the opportunity to use different typefaces, justification and emphasized printing when the document is printed out. A wide variety of word processing packages are available for the Apple, but all those designed to run on Apple][have to come to terms with the machine's 40-column screen presentation of upper-case characters only. Word processing really needs an 80-column screen and the use of upper- and lower-case characters; so all word processing systems for the Apple][involve some compromise. The cheaper ones mainly compromise the user in that they don't give an entirely satisfactory screen presentation — although the quality of the document when printed out on paper can be as good as the printer allows it to be. Dearer systems often involve the use of an 80-column card in order to make the screen appearance of the document during editing more acceptable. The variety of hardware and software combinations that this can involve is considerable.

Many word processing packages of quality are available (see *Windfall* April 1983) including, for those who have the Z80 softcard and CP/M, the very sophisticated **WordStar** package. It's a significant fact that the most important of the enhanced features included in the Apple //e are those that provide for much improved word processing capability. **Applewriter** // is the word processing package developed to take advantage of the Apple //e's improved facilities. Among a range of facilities this package offers the mailmerging capability which allows the personalizing of a standard letter with names and addresses of a mailing list file.

The DIF format Lots of software packages provide facilities for the passing of data from one package to another — for example the passing of numerical data entered in Visicalc into Visifile for sorting and incorporation in reports. A standard format for data storage, the Data Interchange Format (DIF), is often made use of for this purpose. Apple's Business Graphics package can for example accept data in the DIF format. It's worth checking that your software packages are in some way compatible with each other if data you use in one is at all likely to be used in any of the others.

Extensions

(*a*) *Improved garbage collection* Apple programs using lots of string variable data will sometimes stop, unannounced and apparently do nothing for minutes, hours even, as a result of the 'garbage collection' routine built into the Apple's operating system. Most books say all you can do is include a line like X = **FRE(0)** in the parts of your program where most string handling is going on. This forces garbage collection to take place more often — you don't save any time overall, but you do know what is going on! The line below checks how much string memory is left and gives you a warning before starting the garbage collection routine.

```
IF PEEK(112)-PEEK(110) < 4 THEN FLASH
:PRINT "COLLECTING GARBAGE" : NORMAL : X = FRE(0)
```

However, a recent article presented a method of drastically reducing the time spent on garbage collection in programs handling lots of string data. This garbage collection routine is written by Cornelis Bongers and was published in *MICRO*, No 51, August 1982 ('Straight garbage collection for the Apple'). The routine is also available in the Ampersoft package published by MICRO SPARC, INC (The *NIBBLE* magazine), P.O. Box 639, Lincoln, MA 01773 USA. Ampersoft contains, apart from the garbage collection routine, a sophisticated PRINT USING routine, a matrix handling routine (Inverse, Transpose, Identity, Multiply, etc.), a fast sort routine and a routine to store quickly and recall numeric arrays to/from disk. Ampersoft requires a 16K RAM card. Ampersoft and DOS reside in the RAM card, so leaving 46K of free memory for the user. Type the progam printed below, and the DATA lines on the next page, carefully into your Apple—*you will only have to do it once!* SAVE it all on disk and then RUN it.

```
10  REM *** NOGARBAGE MAKER ***
20  READ FI$,LOC,LE
30  FOR L = LOC TO (LOC + LE - 1)
40  READ V : POKE L,V : NEXT L : PRINT
80  PRINT CHR$(4);"BSAVE ";FI$;",A";LOC;",L";LE
90  END
```

A binary file called NOGARBAGE will be created on your disk — this is the file that you need. Make the very first line of any program using it

```
1  PRINT CHR$(4);"BRUN NOGARBAGE":CLEAR
```

Include **&FRE(n)** in any part of your program where much string handling is going on e.g., in a line of the form

```
10  &FRE(4) : REM HERE N=4, AND N*256 IS THE AMOUNT OF
    RAM BELOW WHICH GC WILL BE FORCED
```

This forces a garbage collection routine which is much faster than the normal one — and saves you lots of time!

```
900 DATA "NOGARBAGE",36864,608,1
901 DATA 169,19,141,246,3,169,144,141,247,3,169,0
902 DATA 133,115,169,144,133,116,96,201,189,208,53,32
903 DATA 177,0,32,217,247,24,160,1,162,254,181,111
904 DATA 149,64,181,157,149,68,200,177,155,117,157,149
905 DATA 62,232,208,238,160,0,32,44,254,24,202,181
906 DATA 67,233,0,149,110,232,240,247,32,183,0,201
907 DATA 44,240,204,96,201,214,240,3,76,201,222,74
908 DATA 133,23,32,177,0,32,187,222,201,201,208,8
909 DATA 32,33,146,102,23,32,177,0,41,15,240,11
910 DATA 24,101,110,166,109,228,111,229,112,144,7,165
911 DATA 23,48,3,32,164,144,32,177,0,201,44,208
912 DATA 28,32,177,0,32,227,223,133,133,132,134,32
913 DATA 106,221,32,232,226,165,23,16,3,32,78,232
914 DATA 165,18,32,99,218,76,184,222,133,161,32,110
915 DATA 145,32,128,145,176,59,228,175,229,176,144,245
916 DATA 166,6,240,241,224,3,144,20,177,157,9,128
917 DATA 145,157,136,48,228,177,157,200,145,252,136,185
918 DATA 252,0,176,240,160,0,132,161,177,157,145,252
919 DATA 169,255,200,202,240,2,177,157,145,252,200,169
920 DATA 255,145,252,48,192,32,33,146,76,242,144,32
921 DATA 46,146,62,160,2,41,127,145,157,136,177
922 DATA 157,133,9,136,177,157,133,8,177,8,133,6
923 DATA 200,177,8,136,145,157,160,2,177,8,136,145
924 DATA 157,165,254,229,6,133,254,176,2,198,255,145
925 DATA 8,165,255,200,145,8,164,6,136,177,157,145
926 DATA 254,152,208,248,240,189,165,161,208,8,32,110
927 DATA 145,32,128,145,144,9,165,254,133,111,165,255
928 DATA 133,112,96,162,1,177,252,201,255,208,234,136
929 DATA 177,252,48,5,232,32,20,146,36,136,177,252
930 DATA 32,20,146,138,145,252,200,165,254,145,252,200
931 DATA 165,255,145,252,208,203,169,7,133,7,56,165
932 DATA 105,233,7,133,8,165,106,233,0,133,9,96
933 DATA 24,165,7,170,101,8,133,8,144,2,230,9
934 DATA 224,7,240,54,166,9,160,0,228,160,208,25
935 DATA 197,159,208,21,240,62,160,0,177,8,48,220
936 DATA 200,177,8,16,215,136,165,8,105,2,144,1
937 DATA 232,133,252,134,253,177,252,133,6,200,177,252
938 DATA 133,157,170,200,177,252,133,158,24,96,166,9
939 DATA 228,108,208,210,197,107,208,206,70,7,133,159
940 DATA 134,160,165,159,160,160,0,228,110,208,4
941 DATA 197,109,240,225,133,8,134,9,177,8,170,200
942 DATA 177,8,72,200,177,8,101,159,133,159,200,177
943 DATA 8,101,160,133,160,104,16,214,138,48,211,200
944 DATA 177,8,10,105,5,101,8,133,8,144,129,230
945 DATA 9,76,144,145,164,254,208,2,198,255,198,254
946 DATA 160,0,145,254,96,162,1,181,115,149,254,149
947 DATA 157,202,240,247,160,0,165,158,197,112,144,43
948 DATA 198,158,136,177,157,48,5,152,208,248,240,238
949 DATA 72,166,158,152,24,101,157,144,1,232,228,112
950 DATA 144,16,208,4,197,111,144,10,233,2,133,157
951 DATA 176,1,202,134,158,56,104,96
```

(*b*) *TASC compiler* Programs written in BASIC may seem fast to the newcomer to computers, especially when the task being performed is relatively straightforward. However, as soon as a BASIC program involves lots of complex calculations or lots of sorting for example, its relative lack of speed soon becomes apparent. For those of you who can program adequately in BASIC, who need more speed than BASIC can provide but who don't have the time or inclination to become machine code programmers, a BASIC compiler, such as the Microsoft TASC compiler, can provide an acceptable compromise solution. The TASC compiler is itself a program. It's a program which takes another program, your Applesoft BASIC program, examines it in detail and produces from it a block of machine code which can be used to do the same things that your BASIC program does. The original Applesoft program is known as the **source file** and the block of machine code produced from it is called the **object file**. When the object file is used no 'interpretation' is used and so the program runs much faster. One extension to Applesoft provided by the package is the capability to define COMMON variables which will always be stored in the same location in memory — at the time of compilation TASC provides the user with precise details of where such variables are stored. Saving the relevant portion of memory onto disk as a binary file provides a simple and very speedy way of storing even very large arrays of numeric data on disk and it is just as easily read back in again when needed. Very lengthy file-reading operations can be reduced to just a few seconds of disk access using this technique. There are some limitations attached to the use of the compiler. One particularly convenient feature of Applesoft, the dynamic dimensioning of arrays, is *NOT* available — TASC programs cannot contain sequences which accept input of a variable and *then* use its value to set up string or numeric arrays to suit. All TASC arrays have to be explicitly dimensioned. On the other hand, true integer arithmetic and a considerably more effective CHAIN function for linking programs are valuable extensions to Applesoft. Working effectively with TASC is an acquired skill, but the TASC compiler is a very worthwhile piece of software.

(c) *BitStik graphics* The BitStik is a sophisticated joystick-like device which can be used to produce and manipulate complex high resolution graphics without involving the user in any programming. It is interfaced with the Apple via the games controller socket. Two disk drives, 64K of RAM, a colour TV or monitor and colour card are needed to support the BitStik. After booting up from the BitStik software, two disks are now inserted into the drives. DRIVE 1: A Library Disk containing pre-prepared graphical units that can be inserted into the graphics screen being worked on. Such Library Disks can be prepared by the user using BitStik software commands or, for a number of specialist applications such as electronic circuit drafting, bought commercially. DRIVE 2: A Work Disk on which the graphical designs that BitStik produces can be saved. A drawing area is presented on the screen with a range of options such as DRAW, PAINT, TRACE and many others listed down the right-hand side of the screen. A further set of parameters appropriate to the particular option being used are presented along the bottom of the screen. The software is already configured to give hard copy of graphics onto the Epson MX–80FT2 dot matrix printer and software for sending BitStik originated drawings onto an X–Y plotter is available from BitStik's manufacturer for a range of well-known plotters including Hewlett-Packard and Digiplot. The image produced by plotted output is far superior to that visible on the screen or from a graphics printer, with lines in any direction as straight and curves as smooth as intended by the designer. A built-in feature of the BitStik software is the ability to communicate with a Graphics Tablet connected to the Apple via Slot 5. Selection of the TRACE option from the screen menu allows the Graphics Tablet to take control of the software. Drawings on paper placed on the Tablet can be "traced" onto the BitStik screen for subsequent amendment. BitStik and its associated software are sophisticated CAD tools well worth your attention if the production and use of graphics figure at all in your area of work. BitStik is manufactured by Robocom Ltd., CIL Building, Goodwin Street, London N4 3HQ.

(*d*) *Winchester disks* Floppy disks used with the Apple normally contain a maximum of 140K characters. If this is insufficient for your data handling needs, a Winchester 'hard disk' can provide large (3–50 megabyte) on-line storage which is accessed very fast. Costs are becoming very reasonable. A number of hard disk systems do away with floppy disks altogether and rely on a variety of other devices for backup storage. Most hard disk systems will be 'transparent' to the user in that standard Apple DOS commands can be used to communicate with them.

(*e*) *Print buffer* Whenever your Apple is directed to output to a printer the machine becomes fully occupied performing that task. The printer is a very slow device by comparison with the computer and, if the documents to be printed are long, you can lose the use of the computer for considerable periods. Print buffers, which contain lots of memory chips, are situated in-line between the Apple and the printer. The Apple can send whole documents into the buffer and get back to other jobs while the contents of the buffer are then gradually released at the speed the printer can accept them. Buffers may be built in as part of printer cards, as peripheral cards mounted in the Apple or as self-contained devices with their own power supply that are mounted outside the Apple.

(*f*) *RAM cards* As an 8-bit processor, the 6502 microprocessor that drives the Apple can only address 64K memory locations. However, it is possible to have 'banks' of RAM using the same memory space as each other, with the Apple switching from one bank to another to find what it needs. The extra memory can be used to hold much bigger data arrays than the standard memory configuration can cater for — this is particularly useful for work with spreadsheet programs such as Visicalc. RAM cards can even provide 'virtual disk' capability, holding the contents of whole disks with programs' DOS commands being directed there, instead of to the real disk.

(g) *Z–80 card* This card provides your Apple with the Z–80 microprocessor that is essential in order to use the CP/M operating system that, as an international standard, has been the basis for the development of a very large amount of quality software on other microcomputers. A number of manufacturers now supply these cards, but the Z–80 Softcard from Microsoft comes complete with the CP/M that is needed to use it — Microsoft M–BASIC and G–BASIC are also provided. Once the Z–80 card is installed (usually in Slot 4), it can be accessed either by booting up from a CP/M formatted disk or by issuing a PR# command.

(h) *Multifunction card* Mountain Computer's CPS Card can help solve problems of equipment compatibility if that is likely to be a problem. On one peripheral card three separate functions are provided — a two way serial interface (for use with printers, modems and other devices), a Centronics standard parallel output interface and a real-time clock and calendar. The characteristics of these three interfaces are user definable via a program that is supplied with the card. Once defined, the interface characteristics are stored in an area of RAM on the card itself. This RAM is kept energized, along with the real-time clock, by a small battery also mounted on the card. As with all powerful tools, the CPS card takes some getting used to, but it offers real flexibility of use — and the real-time clock can be very useful in providing for the automatic dating of reports and listings produced by your software.

(i) *Languages* Languages other than Applesoft BASIC can of course be loaded into the RAM that the Apple][language card (or built-in RAM on the Apple //e) provides. COBOL, FORTRAN, FORTH, LISP are available and UCSD Pascal is widely used by Apple software writers for its particular advantages of structure and speed of execution. Apple's implementation of LOGO is widely regarded as one of the best and, for educators, Apple SuperPILOT provides an ideal environment for the development of CAL software.

The Apple //e

If your Apple was produced after the beginning of 1983 then it will be an Apple //e rather than an Apple][. A significant revision of the Apple's construction, the keyboard and certain aspects of its facilities was made in order to maintain the Apple]['s marketability in the face of competition from more recently introduced personal computers. The Apple //e's main features, its Applesoft BASIC and its operation are essentially the same as those of the Apple][and the biggest part of all Apple][software will run on the Apple //e without modification — another example of the philosophy of '**upward compatibility**' that has allowed Apple continually to improve its product but at the same time avoid existing Apple users being left with wasted investment in software. This section summarizes the important differences between the Apple][and the Apple //e and points out the extra facilities available to the //e user. While most of the following section relates to all versions, certain specific detail relates to the UK version of the //e only.

New features With the //e you can forget about the Language Card. The //e comes equipped with 64K of RAM on the main board so that, with Applesoft BASIC in ROM, Integer BASIC or Pascal will be automatically loaded into 16K of the RAM when you boot the system up from a disk which holds one of these other languages. A much accelerated loading procedure incorporated into the boot routine loads the alternative language in just a few seconds. At last you can have upper- and lower-case text on your Apple without any special chips, character generators or peripheral cards. The shift keys and a capitals lock key are used to control the case of text being produced. All keys on the keyboard have an auto-repeat facility when held down. As well as LEFT– and RIGHT–ARROW keys there are now **UP—ARROW** and **DOWN—ARROW** keys as well. These are used to make movement around the screen easier during editing.

The RETURN key is still present, and as important as ever, but it is now marked symbolically ←┘ rather than with the word itself. Some completely new keys are introduced

Del the DELETE key
→| the TAB key
 the OPEN–APPLE key
 the CLOSED–APPLE key

These keys are all programmable function keys in addition to having certain intrinsic functions assigned to them by the Apple //e hardware. OPEN– and CLOSED–APPLE are, among other things, used to substitute for the buttons on the now-defunct Apple][games controllers, so that the //e can be used with the many games programs designed for the][that used them. They are also used to (*a*) *Initiate a 'cold boot'* — this is the equivalent of turning off the mains power to initialize the Apple's memory and then booting up from the disk in Drive 1. To perform this 'cold boot' on the //e hold down the CTRL–key and the OPEN–APPLE key and then press and release the RESET button. (*b*) *Perform a memory test* If you hold down the CTRL– key, the OPEN–APPLE key and the CLOSED–APPLE key all at once, and then press and release the RESET button, the //e performs a test of its RAM chips and warns the user if any faults are discovered. A small switch under the keyboard projection at the front of the machine provides for changing from £ to # according to the application being used — the switch is possible because two character sets (US and local) are included in the //e's character generator ROM. A miniature DB9 connector (wired in parallel with the existing games controller socket) is mounted on the rear of the casing. This provides for considerably easier connection of joysticks, dongles, BitStik, etc. The //e is supplied with much less support documentation than the][used to be. Applesoft BASIC manuals and technical guides are now items for separate purchase only — their standard remains high. An excellent introductory disk, 'Apple presents Apple', provides a 'hands-on' tour around the machine's facilities, new and old.

//e expansion slots Inside the //e there are still eight slots for expansion cards as in the Apple][, but there are some changes in their function. In the Apple][Slot 0 was used only to house the Language Card or additional RAM card. Since the //e comes with 64K RAM already there the need for a Language Card is done away with and Slot 0 isn't there any more! Slots 1 to 7 are still present at the back of the main board and can be used with the whole range of peripheral cards designed for the Apple][. The //e has one additional slot which is specifically intended for an 80-column card. In the UK version this 'auxiliary slot' is situated physically in-line with the ordinary Slot 3 and is wired in parallel with it. Inserting an 80-column card into the auxiliary slot prevents you from using Slot 3. Two different 80-column cards are available from Apple for use in the auxiliary slot:

(*a*) A standard card providing for 40- or 80-column display on 24 lines and for straightforward switching from one display mode to the other.

(*b*) An 'extended' card which also provides an additional 64K of RAM which can be used via bank-switching between blocks of memory.

Whichever card is being used it is activated by a **PR#3** command which can be issued directly or from within a program. When the 80-column mode is activated it is possible to switch to and fro between the 40- and 80-column display format according to the needs of your program. Delivering warnings and displaying titles might for example be best done in 40-column mode, while spreadsheet type displays would benefit from using the 80-column mode. Pressing CTRL–Q directly from the keyboard forces a 40-column display and CTRL–R forces the 80-column display. From within programs PRINT CHR$(17) forces 40 columns and PRINT CHR$(18) forces 80 columns.

//e TV connections The video output from the //e is a composite video signal with colour information encoded to the PAL standard used in the UK and certain other parts of the world. This means that a colour card is no longer required to provide output from the Apple which will give a display on the colour monitors available locally. The quality of the colour display is much better than that available from any of the colour cards that were previously used to convert the Apple][′s American video output standard to one that the UK monitors could use.

The //e still doesn't provide a signal that can be fed directly into the aerial socket of an ordinary TV receiver, but Apple Computer (UK) do provide a relatively inexpensive solution to this problem. The solution is in the form of a card mounted in Slot 7 for support only, with two leads and plugs connected to terminals on the main board. A lead from the card goes off through a firm mounting on the backplate of the //e to the aerial socket of a TV set.

In addition to sending the Apple's video information as the radio frequency signal your TV needs to make a picture, the card also sends the sound output normally directed to the Apple's internal speaker! This means that the volume control on your TV can be used to amplify music, synthesized voice or any other sound output from the Apple. This feature should be of particular value if your Apple is ever used for demonstrations to groups of people.

Pitman

ISBN 0 273 01991 0