The Working Dragon 32

David Lawrence

A library of practical subroutines and programs

The Working Dragon 32

A library of practical
subroutines and programs

David Lawrence

First published 1983 by:
Sunshine Books Ltd.,
Hobhouse Court,
19 Whitcomb Street,
London WC2 7HF

ISBN 0 946408 01 7

Cover designed by Three's Company.
Photograph of the Dragon 32 kindly supplied by Dragon Data Ltd.
Typeset and printed in England by Commercial Colour Press, London E7.

CONTENTS

Contents in detail

CHAPTER 5

High resolution text

5.1 Characters — this chapter looks at the problem of mixing text and high resolution graphics on the screen at the same time. The program enables you to build up any character capable of being fitted into an area on the screen of 32*32 pixels.
5.2 Dictionary — allows you to store up to 100 of your newly created characters at one time so as to provide material for high resolution programs which require text.

CHAPTER 6

Handy programs

6.1 Name and number — a general purpose tool to build up a dictionary of items and their related quantities, for example, in counting your calories.
6.2 Typist — not every program needs to be hundreds of lines long. This one helps you to learn to touch type.
6.3 Texted — a useful word-processing package.
6.4 Music — this introduces the Help function and makes use of the PLAY command to edit music.
6.5 Graph — this program enables the user to draw line graphs of a variety of data, specifying the units and the set-up of the axes.

CHAPTER 7

Fun and games

7.1 Tracker — an infuriating game in which you hunt an invisible quarry.
7.2 Headlong — almost impossible to win, this is a fast skilful game based on the Doodle program.
7.3 Quoits — how fine is your judgment?

Enter the Dragon

This book, and the series of which it forms a part, was undertaken to try and fill a huge gap. That gap was the absence of works aimed at fulfilling the new micro- owner's dream that his or her machine will not simply be a toy, nor even an educational introduction to the silicon age, but a powerful tool, taking over all kinds of tasks and opening up all kinds of possibilities. The majority of books consist either of trivia or assume too great a desire — perhaps even the capacity — to experiment.

I wanted to write a book based on a solid collection of programs in such areas as data storage, finance, graphics, music, household management and education. Discussion of programming techniques would arise out of the programs themselves rather than as part of a curriculum of 'things that should be learned'. I hope that you will find that the book that has emerged from that desire is a useful one, not only as a way of learning new programming techniques but also as a collection of programs in itself, offering a wide range of applications that might otherwise have been open only to those prepared to buy expensive commercial software or able to write substantial programs themselves early on in their programming experience.

In addition to the programs there are the parts of the programs — not as silly as it sounds, for all the programs in this book are written in 'modular' form. That is to say that they are made up of clearly identifiable functional units which, as you come to understand them, you will be able to lift out and employ for your own purposes.

Each module is fully commented upon where it covers new ground and instructions are given for the testing of programs at each stage of their entry.

In using this book you will find that, though there are sections where general issues are discussed, it is not a book to be read but to be used. The relevance of comments and advice will often only be apparent when you have taken the plunge and begun the task of entering what appear at first to be dauntingly long and complex programs. Here the modular approach will help to prevent programs becoming unredeemable tangles of errors, so do take the tests suggested seriously.

In the end, however, the success of failure of this book must be judged on whether it helps you to enjoy your Dragon. Whilst the structure of this book is closely modelled on my earlier book, *The Working Spectrum*, the programs have been revised extensively in the light of the Dragon's very different capabilities. I have enjoyed the writing of these programs — enjoyed the sensation that has come very often in the writing of this book that the Dragon has produced better programs for me than other micros I have worked with. It is an idiosyncratic machine, sometimes a downright irritating one but its capabilties go far beyond many others and, I suspect, far beyond the realisation of many of its owners.

Notes on style

1) Because multi-statement lines are difficult to debug, the early programs in this book, by and large, avoid their use. If you feel confident of your own ability there is no reason why you should not merge lines. This does mean that extra care should be taken in testing modules.
2) I have adopted no short cuts when it comes to spelling out Basic statements. You may well wish to omit LETs from statement lines or replace PRINT with ?.
3) The printer on which these programs were listed has replaced all occurrences of # with £, a fact of which you will need to be aware.

PROGRAM NOTES

There are two points to note with the listings:
1) The # symbol is always represented by £.
2) The ↑ symbol is always represented by ^.
3) Inverse characters appear in the program printouts in lower case.

CHAPTER 1
Storing and Searching

1.1 UNIFILE I

Sooner or later, most micro owners realize that their new digital friend really comes into its own when it is storing information, processing it and presenting it in ways that would be laborious in the extreme if done manually. They then begin the task of writing simple programs which will store their friends' names and addresses or catalogue their stamp albums. They may end up with half a dozen programs, each limited to one use but each working on much the same method.

In this opening chapter we jump in at the deep end and examine how a single program can be written to satisfy a wide variety of different filing tasks, without the need for constant rewriting every time a new application comes along.

The program is called Unifile and it is capable of flexibly storing up to 500 entries, as well as allowing the user to search through them for named items, to change entries and to delete them. Quite apart from the wide applications of the program, however, in building it up we shall learn a great deal about the Dragon's considerable abilities as a working member of the family.

MODULE 1.1.1

```
1 GOTO 3
2 CSAVE "UNIFILE":SOUND 1,1:STOP
3 REM
```

I think it's worth mentioning that all of my programs begin with these three lines (though obviously the program name changes). Typing GOTO2 is a great deal easier than spelling out the program name every time and a lot less prone to error. The result is that you will be more likely to save the program on tape regularly as you build it up and the result of that will be to save you a lot of frustration when you one day accidentally lose the last two hours hard work as a result of mishap or stupidity. Always enter these three lines first and then save the program regularly as you enter it.

MODULE 1.1.2

```
1000 REM****************************
1010 REM MENU
1020 REM****************************
```

```
1030 CLS:PRINT @ 9,STRING$(9,140)
1040 PRINT @ 41,CHR$(128)+"UNIFILE"+CHR$
(128)
1050 PRINT @ 73,STRING$(9,131)
1060 PRINT:PRINT "COMMANDS AVAILABLE:"
1070 PRINT: PRINT "      1)SET UP NEW FILE
"
1080 PRINT "      2)ENTER INFORMATION"
1090 PRINT "      3)SEARCH/DISPLAY/CHANGE"
1100 PRINT "      4)DATA FILES"
1110 PRINT "      5)STOP
1120 PRINT: INPUT "WHICH DO YOU REQUIRE:
"; Z
1130 CLS
1140 ON Z GOSUB 1500,2000,3500,6000,1170
1160 GOTO 1000
1170 PRINT @ 260,STRING$(22,140)
1180 PRINT @ 292,CHR$(128)+"FILING SYSTE
M CLOSED"+CHR$(128)
1190 PRINT @ 324,STRING$(22,131)
1200 END
```

As a rule of thumb, a utility program that does not commence with a fairly clear- cut menu of what the program does is a bad program. And if you don't agree with that statement now, you certainly will at some time when you have to return to a complex but useful program which has not been used for some weeks and find that you have to spend hours going through the listing trying to remind yourself of what it does and how.

In this module, which is common to many of the programs in this book, the user is asked to choose between five numbered functions. If a number outside the range 1–5 is input, it is ignored.

Commentary

Lines 1000–1020: All of the program modules in this book are labelled in this way. Normally the modules so headed are subroutines but even where they are not, they represent a clear-cut program function.

Lines 1030–1050: An uncomplicated way of dressing up the titles used in the process of the program. STRING$ simply prints a line of the same character, the line being as long as the first figure in the parentheses. The second figure is the ASCII code of one of the graphics characters referred to in Appendix A of the Dragon manual. These characters can be printed on the screen but not displayed in a program line since there is no key on the keyboard which will access them.

Line 1140: An economical and time-saving way of choosing between the different destinations. Without the ON. . . GOSUB we should be looking at a series of five IF. . THEN. . GOTOs. The destination chosen by this line will the Zth, based on the user's input.

Line 1160: Although lines 1000–1020 serve no useful purpose, lines which return the program to the beginning of this or any other module should always point to these first, decorative lines since you may, at some stage want to add another line before the present first, functional line at 1030, necessitating changes to any lines which took 1030 as the start of the module.

Lines 1070–1200: These lines are not strictly necessary but they neatly terminate the use of the program.

Testing Module 1.1.2

At this stage, all that can be tested is that the module presents a neatly ordered menu page and accepts an input. Inputs in the range 1–4 should result in an undefined line error report. Input of 5 should terminate the program. Any other input should be ignored.

MODULE 1.1.3

```
 4000 REM****************************
 4010 REM FUNCTIONAL SUBROUTINES
 4020 REM****************************
 4030 LINE INPUT Q$
 4040 LET Q$=Q$+"^"
 4050 RETURN
 4060 LET ST=INSTR(ST+1,B$(S),"^")
 4070 RETURN
```

When a function, whether complex or simple, needs to be carried out several times and in several places during the course of the execution of a program, it is worth considering either defining a user-defined function, which will be discussed later, or inserting a short subroutine which will do the job. These two short subroutines need to be entered early on since they are called up fairly frequently by other modules but the explanation of their functions will be left until they are actually used.

MODULE 1.1.4

```
 1500 REM****************************
 1510 REM ENTRY STRUCTURE
 1520 REM****************************
 1530 PCLEAR1:CLEAR 20000
 1540 PRINT @ 8,STRING$(16,128)
 1550 PRINT @ 40,CHR$(128)+"FILE STRUCTUR
E"+CHR$(128)
 1560 PRINT @ 72,STRING$(15,128)
 1570 PRINT:INPUT "HOW MANY ITEMS IN EACH
  ENTRY";X:CLS
 1580 DIM A$(X-1)
 1590 PRINT @ 9,"names";CHR$(128);"of";CH
R$(128);"items"
 1600 FOR I=0 TO X-1
 1610 PRINT "ITEM ";I+1;": ";
 1620 INPUT A$(I)
 1630 NEXT I
 1640 DIM B$(499)
 1650 LET B$(0)=CHR$(0)+"^"
 1660 B$(1)=CHR$(255)+"^"
 1670 N=2
 1680 GOTO 1000
```

The purpose of this module is to provide Unifile with its chameleon-like properties by allowing the user to specify the kind of file to be set up, and the names of the items that a typical entry will contain.

Commentary

Line 1530: PCLEAR 1 is an important instruction in programs such as this one, which require as much memory as possible for maximum usefulness. When you switch your Dragon on it automatically reserves roughly 6,000 memory locations for use with graphics. Since we shall not be using graphics in a data-handling program like this one we can reduce the amount of memory given over to the display. All we actually need is one 'page' or screenful of memory on which to display the program's printed output. This makes available an extra 4,500 memory locations — a considerable addition. The other command on this line sets aside 20,000 memory locations specifically for our filing use, otherwise we would quickly run out of space.

Lines 1570–1630: In this section the user is requested to input the number of items which a typical entry will contain, and then to give each item a name, such as 'name, address, etc.'. An array called A$ is set up, with as many elements as there will be items per entry. Note that although the user specifies X items, the array A$ is set up with apparently only X-1 elements in it. This is because in the version of the Basic language which the Dragon uses, all such arrays actually start with element number zero. You can ignore the zero element in your programming, which does make the numbering of things more sensible, but then again it wastes space. It's sad that a modern machine such as the Dragon has to be tied to such an outdated convention.

Line 1640: Our file will be held in the array B$, which has 499 + 1 elements. One limitation of this will be that the total number of characters in any one entry will not be able to exceed 255 and the program will have to be made to ensure that this does not happen in error.

Lines 1650–1660: The file will be arranged in alphabetical order of the first item in each entry. The method we shall use to insert new entries into the file in their correct position requires that there be some entries there already, to compare the new entry with. So rather than start with an empty file we insert a pair of dummy entries. The actual entries are the single characters CHR$(0) and CHR$(255). Neither of them actually mean anything, they are simply the A and Z of the Dragon's alphabet and any subsequent entry which begins with a normal text character will automatically be inserted between them.

Line 1670: The variable N records the number of entries in the file. Because of the eccentric numbering of the arrays, N will seldom if ever be used on its own, but usually as N-1 or N-2.

Line 1680: Subroutines end with the RETURN command, returning program execution to the line which called the subroutine up in the first place. The difference here is that during the course of this module we cleared the memory, and this led to the loss of the return address of the line which called the subroutine. Hence in this particular case we need to specify the return address.

Testing Module 1.1.4

You should now be able to RUN the program and call up the first function on the menu. You should be requested to specify the number of items in each entry and then to name the type of item. Having named the requisite number of items you should be returned to the menu. You may wish to check, in direct mode, that the names of the items are stored in the first X-1 places of the array A$.

MODULE 1.1.5

```
2000 REM***************************
2010 REM NORMAL INPUT
2020 REM***************************
2030 LET R$=""
2040 PRINT @ 10,STRING$(9,140)
2050 PRINT @ 42,CHR$(128)+"ENTRIES"+CHR$
(128)
2060 PRINT @ 74,STRING$(9,131)
2070 PRINT "COMMANDS AVAILABLE:"
2080 PRINT ">ENTER ITEM SPECIFIED":PRINT
 ">'ZZZ' TO RETURN TO MENU"
2090 PRINT STRING$(32,"*");
2100 PRINT "NUMBER OF ITEMS:";N-2;"/500"
2110 FOR I=0 TO X-1
2120 PRINT A$(I);":";
2130 GOSUB 4030
2150 IF LEN (R$)+LEN (Q$)>255 THEN PRINT
 "ENTRY TOO LONG.":FOR J=1 TO 5000:NEXT
J:RETURN
2160 IF Q$="ZZZ^" THEN RETURN
2170 LET R$=R$+Q$
2180 NEXT I
2190 CLS
2200 GOSUB 2500
2210 GOTO 2000
```

The purpose of this module is to accept the input of a new entry composed of the correct number of items specified in the last module and to present the new entry to the section of the program which will insert it into its correct place in the file.

Commentary

Line 2030: R$ is the string in which the entry will be built up before being placed in the main file.

Line 2130: Items for input are all accepted by the short subroutine (already entered) at 4030. All this does is to add the character ↑ to the end of each item. This symbol will later be used to redivide the entry into its constituent items. This means that the character ↑ is a reserved symbol as far as this program is concerned, and if you include it in an item then you are likely to get a nonsense result from that particular entry.

Line 2150: Having accepted an item, the length of the entry is checked to see that it will not exceed the maximum length imposed by the Dragon of 255 characters for a single string.

Line 2160: At any point in the process of inputting an entry, the user can type ZZZ as an item and the program will return to the menu.

Line 2170: Provided that the entry is not too long and the user has not input ZZZ, the item just input, with its added ↑ character, is added to R$ which is being built up into the completed entry.

Line 2200: The module which inserts the completed entry into the main file is now summoned up.

Testing Module 1.1.5

Though items cannot be inserted into the actual file, you should now be able to call up the first function on the menu, specify item names then call up this module and input items under your named headings. Putting a temporary line 2500 RETURN should enable you to go on doing it indefinitely. You should check that the module does not accept entries whose overall length is greater than 255 characters.

MODULE 1.1.6

```
2500 REM*************************
2510 REM PLACE DATA IN FILE
2520 REM*************************
2530 IF N<500 THEN GOTO 2560
2540 CLS:PRINT @ 14*32+10,"FILE NOW FULL
"
2550 FOR I=1 TO 1000:NEXT I:RETURN
2560 LET POWER=INT (LOG(N-1)/LOG(2))
2570 LET S=2^POWER
2580 LET T$=LEFT$(R$,INSTR(R$,"^")-1)
2590 FOR K=POWER-1 TO 0 STEP -1
2600 LET ST=1:GOSUB 4060:LET U$=LEFT$(B$
(S),ST-1)
2610 IF T$>U$ THEN LET S=S+2^K
2620 IF T$<U$ THEN LET S=S-2^K
2630 IF S>N-1 THEN LET S=N-1
2640 IF S<1 THEN LET S=1
2650 NEXT K
2660 LET ST=1:GOSUB 4060:LET U$=LEFT$(B$
(S),ST-1)
2670 IF T$<U$ THEN LET S=S-1
2680 FOR I=N+1 TO INT(S+2) STEP-1
2690 LET B$(I)=B$(I-1)
2700 NEXT I
2710 LET B$(S+1)=R$
2720 LET N=N+1
2730 RETURN
```

Probably the most complex module in the program. The purpose of this module is to determine the correct place for a new entry in the file. To understand its functions you must first of all understand the technique of the 'binary search' which is used to dramatically reduce the number of comparisons between entries that have to be made before the correct position is determined. Consider the following example:

We have established a file containing 2,000 entries (not in this program, but never mind) and there is a new entry which needs to be inserted at position 1731, though the program has yet to determine this. The program begins its search by looking at the first entry in the file and comparing it with the new entry. The new entry is found to be the greater of the two alphabetically and so the program proceeds to examine the next entry in the file and so on for 1, 731 comparisons until the correct place is found. This is a straightforward procedure and an easy one to program. Compare it with this:

The program begins by examining the entry in position 1024 in the file, since 1024 is the greatest power of 2 which is less than the total number of entries in the file. Entry 1024 is found to be alphabetically less than the new entry and so the program adds 1024/2 to the original 1024 and moves on to entry number 1536. The entry at 1536 is still less than the new entry and so 1024/4 is added to 1536, making 1792. Entry number 1792 is greater than the new entry and so 1024/8 is subtracted from 1792 giving 1664. The search proceeds at the following locations in the file and with the following additions or subtractions:
1644 (then add 64)
1728 (then add 32)
1760 (then subtract 16)
1744 (then subtract 8)
1736 (then subtract 4)
1732 (then subtract 2)
1730 (then add 1)
Final result 1731.

The power of a binary search should be apparent.

Commentary

Lines2530–2550: A check is made to determine that there is in fact room in the file for another entry.

Line 2560: This line determines the largest power of two that is less than the total length of the file (including the zero element).

Line 2580: Using the powerful INSTR function, which scans a specified string for any combination of characters named, we find the first ↑ in the new entry and take everything to the left of that as the first item in the entry, T$.

Lines 2590–2650: This loop, using decreasing powers of 2 to add and subtract from the number of the search location, moves through the file carrying out the type of binary search described above.

Line 2600: This line calls up the one line subroutine at line 4060. The purpose of that subroutine is to search for the first occurrence of the character ↑ in whichever entry in B$ is pointed to by the search variable S. The search begins at character ST, which in this case is set at 1. The resulting figure is used in line 2600 to identify the first item of the particular entry in B$.

Lines 2630–2640: If the search moves beyond the confines of the file it is shunted back in.

Lines 2660–2670: At the end of the search the file entry next to which the new entry must be placed has been identified. These two lines are needed to see whether the new entry should go before or after the existing one.

Lines 2680–2710: The new entry is inserted into the file by the simple method of moving everything above its intended position up one place. Other versions of Unifile for other popular micros have avoided such a solution as being too slow, leading to complex methods of recording the correct position of each entry in the file. In the case of the Dragon, the speed of the machine meant that, even with a fairly full file the wait involved in shifting all the elements one place is such that it is not worth wasting the extra program space on more complicated solutions. Were the Dragon to be mass marketed with a 64K memory, permitting a practical file of more than 1,000 items easily, then you might want to look again at that decision. If you happen to like complexity for the sake of it, I would refer you to the method described in the same chapter of the previous book in this series, *The Working Spectrum*.

Testing Module 1.1.6

You should now be in a position, having set up your file, to enter some items and see them correctly inserted into the main file. Since you have not yet entered the module which displays the file, this can only be checked in direct mode by first entering one or two items and then printing out B$(1),B$(2) etc. If the module is misbehaving, work through the procedure (as you have entered it, not as it is in the book) with paper and a pencil, for the entries you have made. Such a technique is almost always the best way of debugging a complex module such as this. 'S starts as 2 and B$(2) would be . . . . .' may seem like a laborious method but it reduces the complexity to a manageable level.

MODULE 1.1.7

```
3500 REM***************************
3510 REM SEARCH
3520 REM***************************
3530 LET S1=1
3540 PRINT @ 11,STRING$(8,140)
3550 PRINT @ 43,CHR$(128)+"SEARCH"+CHR$(
128)
3560 PRINT @ 75,STRING$(8,131)
3570 PRINT ">INPUT SEARCH ITEM",">ENTER
FOR FIRST ITEM ON FILE"
3580 PRINT STRING$(32,137)
3590 INPUT "ENTER SEARCH COMMAND:";S$
3600 IF S$="" THEN GOTO 3650
3610 FOR S1=1 TO N-2
3620 IF INSTR (B$(S1),S$)<>0 THEN GOTO 3
660
3630 NEXT S1
3640 RETURN
3650 IF INT(S1)=N-1 THEN RETURN
3660 CLS
3670 PRINT "ENTRY ";S1;":-"
3680 LET ST=0
3690 FOR I=0 TO X-1
3700 LET TEMP=ST+1
3710 LET S=S1:GOSUB 4060:PRINT A$(I);":"
;MID$(B$(S1),TEMP,ST-TEMP)
3720 NEXT I
3730 LET S1=S1+1
3740 PRINT @ 10*32,"search:";S$
3750 PRINT "COMMANDS AVAILABLE:"
3760 PRINT ">'ENTER' FOR NEXT ITEM"
3770 PRINT ">'AAA' TO AMEND"
3780 PRINT ">'CCC' TO CONTINUE SEARCH"
3790 PRINT ">'ZZZ' TO QUIT FUNCTION";
3800 INPUT P$
3810 CLS
3820 IF P$="CCC" THEN GOTO 3620
3830 IF P$="" THEN GOTO 3650
3840 IF P$="AAA" THEN GOSUB 3010:GOTO 36
50
3850 IF P$="ZZZ" THEN RETURN
3860 LET S1=S1-1:GOTO 3650
```

Having placed your entries into the file, it would be nice to know that they can be retrieved for later examination. More than that, since they are stored in an electronic marvel, it would be nice to think that they could be retrieved at high speed and in clever ways. This is what this module sets out to achieve. Here again, we make use of the INSTR function on the Dragon, one of the most useful innovations to have surfaced in recent years.

Commentary

Line 3530: S1 is the variable that will be used to point to the entry to be displayed. Note that it starts at 1 since the first real (as opposed to dummy) entry is at B$(1).

Line 3570: To begin with the user is offered two choices:
1) to enter some characters which will then be searched for in the file.
2) to press ENTER, which will display the first item in the file.

Lines 3610–3640: If the user inputs anything other than ENTER, this simple loop will begin a fast search for it through the file. All that happens is that the INSTR function is applied to each item in the file in turn. If it produces a value other than zero, then the specified combination of characters is present in that file entry and it is displayed by a later part of the module. Having displayed an item which satisfies the search criteria, the search may be continued later if the user requires.

Line 3650: You may wonder why the INT function, which reduces a number like 1.16 to the integer 1, is applied to S1. The reason is that in a module you have yet to enter, SI is set equal to the search variable S from the binary search. That variable can sometimes pick up a totally invisible fraction as a result of inaccuracies in the LOG function. You cannot see anything irregular about the value of the variable, but it will never be found to be equal to an integer like N (the number of items in the file). As a safety check against this extremely infrequent occurrence, the INT function is used. It could just as well have been done at line 3240 in the next module, which sets SI = S + 1.

Lines 3660–3720: You should recognize what is going on here from previous lines you have entered. The subroutine at line 4060 is being used to find the ↑ symbols and thus identify the different items in the entry to be displayed. The difference here to previous uses is that, with the aid of the variable TEMP, the loop works all the way through the entry, rather than finding only the first item: TEMP records the first character of an item while ST is set by 4060 to the position of the ↑ at the end of the item — then TEMP is set to the position after ST, and so on.

Lines 3740–3850: Once an entry has been displayed, a new set of options is offered to the user. The search can be continued (for the same target characters), the next entry can be displayed, the search module can be quit, or the amend module can be called.

Line 3860: SI must be decremented since it has already moved on one place. This only happens if the user makes an incorrect input. The effect is simply to leave the same entry on display.

Testing Module 1.1.7

Having set up and entered some material on your file, you should now be able to search for any combination of characters within the file, or to move through the file entry by entry, pressing ENTER. If a combination of characters is not present in the file, the program should return to the menu. The same should happen if you move past the last entry. Tagging ↑ onto the front of the characters you specify for a search should result in a search only for items that begin with those characters.

MODULE 1.1.8

```
3000 REM*****************************
3010 REM CHANGE ENTRY
3020 REM*****************************
3030 LET S1=S1-1
3040 LET R$=""
3050 PRINT "ENTRY ";S1;":-"
3060 LET ST=0
3070 FOR I=0 TO X-1
3080 LET TEMP=ST+1
3090 LET S=S1:GOSUB 4060:PRINT A$(I);": "
;MID$(B$(S1),TEMP,ST-TEMP)
3100 PRINT @ 9*32+12,"amend"
3110 PRINT "COMMANDS AVAILABLE:"
3120 PRINT "'ENTER' LEAVES ITEM UNCHANGE
D"
3130 PRINT ">ENTER NEW ITEM"
3140 PRINT ">'DDD' DELETES WHOLE ENTRY"
3145 PRINT ">'ZZZ' TO QUIT FUNCTION"
3150 GOSUB 4030
3160 CLS
3170 IF Q$="ZZZ^" THEN RETURN
3180 IF Q$="↑" THEN LET Q$=MID$(B$(S1),T
EMP,ST-TEMP) + "↑":GOTO 3200
3190 IF Q$="DDD^" THEN GOSUB 4500:RETURN
3200 LET R$=R$+Q$
3210 NEXT I
3220 GOSUB 4530
3230 GOSUB 2500
3240 LET S1=S+1
3250 RETURN
```

This module permits the user to make changes to existing entries and to delete entries from the file.

Commentary

Line 3150: Note the continuing use of the short subroutine at 4030 to accept inputs.

Line 3180: Whether or not any changes are actually made, a new entry is being built up in a string called R$, which will be presented to the insertion module. If ENTER is pressed in response to any item, the item is lifted out of the original entry and placed into the new entry.

Line 3190: An input of DDD results in deletion of the entry by a module yet to be entered.

Line 3200: Entry of any combination of characters other than the prompts specified in the menu for this module is taken as the input of an item to replace the item currently on display.

Line 3220: The current entry is deleted since the new entry may not fit alphabetically into the same position.

Testing Module 1.1.8

The module cannot be properly tested until the next short module has been entered.

MODULE 1.1.9

```
4500 REM**************************
4510 REM TELESCOPE FILE
4520 REM**************************
4530 FOR I=S1 TO N-1
4540 LET B$(I)=B$(I+1)
4550 NEXT I
4560 LET N=N-1
4570 RETURN
```

This module simply shifts the whole file, from the position above the item to be deleted, down one place, thus overwriting it.

Commentary

Line 4560: Note that there is now a duplicated item at the end of the file — the dummy entry. This is not erased since once N has been reduced by 1, the program will no longer be aware of the position of the second entry.

Testing Module 1.1.9

You should now be able to amend existing entries and also to delete them from the file.

MODULE 1.1.10

```
6000 REM**************************
6010 REM DATA FILES
6020 REM**************************
6030 MOTOR ON:AUDIO ON:CLS:INPUT "POSITI
ON TAPE THEN PRESS enter  (MOTOR IS ON):
";Q$
6040 MOTOR OFF:INPUT "PLACE RECORDER INT
O CORRECT MODETHEN PRESS enter:";Q$
6050 PRINT:PRINT "FUNCTIONS AVAILABLE:",
"1)SAVE DATA",,"2)LOAD DATA":INPUT "WHIC
H DO YOU REQUIRE:";Q:ON Q GOTO 6070,6190
6060 RETURN
6070 MOTOR ON:FOR I=1 TO 10000:NEXT
6080 OPEN "O",£-1,"UNIFILE"
6090 PRINT £-1,X
6100 FOR I=0 TO X-1
6110 PRINT £-1,A$(I)
6120 NEXT I
6130 PRINT £-1,N
6140 FOR I=1 TO N-2
6150 PRINT £-1,B$(I)
6160 NEXT I
6170 CLOSE £-1
6180 RETURN
6190 PCLEAR1:CLEAR 20000:DIM B$(499)
6200 OPEN "I",£-1,"UNIFILE"
6210 INPUT £-1,X
6220 DIM A$(X)
6230 FOR I=0 TO X-1
6240 INPUT £-1,A$(I)
6250 NEXT I
6260 INPUT £-1,N
6270 FOR I=1 TO N-2
6280 INPUT £-1,B$(I)
6290 NEXT I
6300 LET B$(0)=CHR$(0)+"^"
6310 LET B$(N-1)=CHR$(255)+"^"
6320 GOTO 1000
```

This module, or one very much like it, will be shared with the vast majority of programs in this book. Its purpose is to allow the user to save the contents of the file and the associated variables, on tape. The advantage of this is that if changes have to be made to the program, or when the machine is switched off, the data need not be lost. Many modern micros automatically save the data associated with a program when that program is saved (or CSAVEd), unfortunately the Dragon does not. To make up for this the Dragon provides a flexible set of file handling commands which allow data to be stored on tape and retrieved with ease. This module is designed to make the process of finding the right place on the tape and saving or loading as painless as possible.

Commentary

Line 6030: This command is included for those who have purchased the special lead to connect the Dragon to their cassette recorder and can therefore place their recorder under the control of the Dragon. If you don't have such a lead it will serve as a reminder of the procedure to follow in loading or saving. What the line does is to switch on the cassette recorder's motor, and the Dragon's AUDIO function, thus allowing you to position the tape in the recorder to take account of the contents.

Lines 6040–6050: Having positioned the tape, the cassette is put into record or playback mode, according to whether you want to load or save. The correct function is then chosen from the menu.

Lines 6070–6180: This section prints a 'header' of several seconds before actually recording the data. This ensures that when you read the data back you are not going to find it immediately preceded by the garbled contents of some previous recording, leading to an error message. The rest of the section opens an output file, that is, opens communication with the cassette recorder and then 'prints' into that file (i.e. onto tape) the contents of A$ and B$.

Lines 6190–6320: This section is slightly more complex since, before loading a set of data from tape, the memory must first be cleared in the same way that it would be in setting up a file in the first place. In addition, once having cleared the memory, arrays must be set up to receive the data. In the case of the array A$, this cannot be done until we have first read from the tape the number of item names it is to contain. Lastly the dummy entries must again be loaded into first and last place. They cannot be saved on tape and reloaded since they are not recognised as real characters by the Dragon's operating system. Since the memory has been cleared we must return with a GOTO, as in Module 4.

Testing Module 1.1.10

Having entered some data and satisfied yourself that the rest of the program functions are working satisfactorily, call up this module from the menu and save the file on tape. Stop the program then restart with RUN, which will wipe out all the data. Call up this module again and reload your file— it should beas if you had never lost the data. If this module functions correctly then the program is ready for use.

Summary

You have now completed the input of a substantial and complex program which I hope you find useful in a variety of applications. Along with that process you have also learned a number of techniques which will stand you in good stead whenever you decide to embark on ambitious programs to store and process non- numeric data.

More importantly, however, if you have taken the trouble to understand what you have been entering, you will have gained confidence that substantial and complex programs are not as awesome as they are often made out to be. Using a modular approach which breaks down the program into a series of manageable tasks, applications like this one can be tackled by anyone who is prepared to invest a little time (and a little hair).

Going further

1) The program is deliberately written without much use of multi-statement lines. Once it is working well and you understand what is going on, try to reduce the number of lines by combining them where appropriate. Substantial memory savings are to be had in this way.
2) The program makes no provision for output to a printer if you possess one.
3) One fascinating challenge would be to see if you could make the program handle numeric data as well as strings. This would involve setting up a numeric array with 500 elements, at least, and making provision to input values to it and perhaps some search commands appropriate to the stored values, like 'search for all items larger than specified number'. There are quite a large range of applications where the ability to store one or more numeric items in an entry would be an advantage.

1.2 UNIFILE II

After entering Unifile I and debugging it, the last thing that you may want to face is variations on it, so feel free to skip this part of the chapter forthe present if you'd like to move on to freshpastures. At some stage, however, you will want to come back to this program to solve some problems that Unifile itself is not designed to cope with. Unifile is fine for those files which have a regular structure, and many do. Equally, there are a large number of applications where you simply do not know in advance how many items there are going to be in an entry. You may, for instance, want to catalogue your books. You could set up the original Unifile program to request author and title but if you happen to have several books by the same author, that is going to be a great waste of space.

Unifile II is designed to cope with such less structured files. It is more flexible than Unifile I in that you can go on adding items to an entry as long as you like (provided that the total length does not exceed 255 characters). The price to be paid is that the program is more complicated to use since there are none of the easy prompts as to which item to put in next. If you want to label items you have to specify the labels to be attached to them. The program is also more flexible in that it will conduct a 'multiple search' — that is, it will search for entries in the file which satisfy a number of search criteria at the same time like all entries containing 'red' and 'London' and 'male' if you were looking for the entry of a red-headed man who lives in London.

Because the program is so similar to Unifile I we shall begin by entering the only new module it requires and the rest of the modules will only be commented upon insofar as they differ significantly from the original. Note that the modules are in the same place within the program but that there is not an exact correspondence between line numbers. You can use Unifile I as a basis for entering this one but you will need to adapt GOTOs and GOSUBs here and there.

MODULE 1.2.I

```
5000 REM ***************************
5010 REM ITEM TYPES
5020 REM***************************
5030 FOR I=0 TO 49 STEP 7
5040 CLS:FOR J=I TO I+6
5050 PRINT J+1;" >";A$(J)
5060 NEXT J
5070 PRINT:PRINT "COMMANDS:"
5080 PRINT ">'ZZZ' QUIT"
5090 PRINT ">'III'=ITEM"
5100 PRINT ">'NNN'=NEXT PAGE"
5110 INPUT Q$:IF Q$="ZZZ" THEN RETURN
5120 IF Q$="NNN" THEN NEXT I:RETURN
5130 IF Q$="III" THEN INPUT "NUMBER:";TY
PE:INPUT "TYPE NAME:";Q$:LET A$(TYPE-1)=
Q$:CLS:GOTO 5040
5140 GOTO 5110
```

Since there will be no regular structure to the files this program will handle, there will be no regular set of headings for the items. This module allows headings to be stored which may be attached to items if the user specifies a heading when an item is input.

Commentary

Lines 5030–5040: The effect of these two loops is to display all the items stored in A$ in batches of 7.

Lines 5070 – 5130: The user has the option of moving on to the next batch of item headings, quitting the module, or inputting III. In the latter case, the user is asked to specify a position in the dictionary and a heading to be placed in that position. There is no provision for deletion, which can in any case be accomplished by simply inputting an empty string to the desired position. The use of these item types will be discussed later.

Testing Module 1.2.1

Full testing of this module cannot be undertaken yet, but a temporary test can be made by dimensioning an array A$(49) in direct mode and then entering GOTO 5000. You should be a)le to enter item types, to delete them and to alter them.

MODULE 1.2.2

```
1000 REM****************************
1010 REM MENU
1020 REM****************************
1030 CLS:PRINT @ 9,STRING$(9,140)
1040 PRINT @ 41,CHR$(128)+"UNIFILE"+CHR$
(128)
1050 PRINT @ 73,STRING$(9,131)
1060 PRINT:PRINT "COMMANDS AVAILABLE:"
1070 PRINT: PRINT "     1)SET UP NEW FILE
"
1080 PRINT "     2)ENTER INFORMATION"
1090 PRINT "     3)SEARCH/DISPLAY/CHANGE"
1100 PRINT "     4)DATA FILES"
1110 PRINT "     5)NEW ITEM NAMES"
1120 PRINT "     6)STOP
1130 PRINT: INPUT "WHICH DO YOU REQUIRE:
"; Z
1140 CLS
1150 ON Z GOSUB 1500,2000,3500,6000,5000
,1180
1160 CLS
1170 GOTO 1000
1180 PRINT @ 260,STRING$(22,140)
1190 PRINT @ 292,CHR$(128)+"FILING SYSTE
M CLOSED"+CHR$(128)
1200 PRINT @ 324,STRING$(22,131)
1210 END
```

For comments and testing see Unifile 1.1.1, with the exception that you should be able to call up the item names function, which will report a BAD SUBSCRIPT error.

MODULE 1.2.3

```
4000 REM****************************
4010 REM FUNCTIONAL SUBROUTINES
4020 REM****************************
4030 LINE INPUT Q$
4040 LET Q$=Q$+"^"
4050 RETURN
4060 LET ST=INSTR(ST+1,B$(S),"^")
4070 RETURN
```

As Unifile I.1.3.

MODULE 1.2.4

```
1500 REM***************************
1510 REM ENTRY STRUCTURE
1520 REM***************************
1530 PCLEAR1:CLEAR 20000
1540 DIM M$(19)
1550 DIM A$(49)
1560 DIM B$(499)
1570 LET B$(0)=CHR$(0)+"^"
1580 B$(1)=CHR$ (255)+"^"
1590 N=2
1600 GOTO 1000
```

Shorter than the equivalent in Unifile, since there is no need to make provision for the input of item headings here.

Commentary

Line 1540: The array M$ will be used to store the individual items to be searched for when a multiple search is specified. Accordingly, up to 20 items can be included in the search.

Line 1550: A$ will be used to store the item headings input by Module 1.2.1. There can be 50 such headings.

Testing Module 1.2.4

Calling up menu function 5 after calling this module should not result in an error as before.

MODULE 1.2.5

```
2000 REM***************************
2010 REM NORMAL INPUT
2020 REM***************************
2030 LET R$=""
2040 PRINT @ 10,STRING$(9,140)
2050 PRINT @ 42,CHR$(128)+"ENTRIES"+CHR$
(128)
2060 PRINT @ 74,STRING$(9,131)
2070 PRINT "COMMANDS AVAILABLE:"
2080 PRINT ">ENTER ITEM SPECIFIED ('*' E
NDS)";:PRINT ">'ZZZ' TO RETURN TO MENU"
2090 PRINT STRING$(32,"*");
2100 PRINT "NUMBER OF ITEMS:";N-2;"/500"
2110 GOSUB 4030
2120 IF Q$="ZZZ^" THEN RETURN
2130 IF LEN (R$)+LEN (Q$)>255 THEN PRINT
 "ENTRY TOO LONG.":FOR J=1 TO 5000:NEXT
J:RETURN
2140 IF LEFT$(Q$,1)="£" THEN PRINT A$(VA
L(MID$(Q$,2,2))-1);":";
2150 IF LEFT$(Q$,1)="£" THEN PRINT MID$(
Q$,4,LEN(Q$)-4) ELSE PRINT "UNTYPED:";LE
FT$(Q$,LEN(Q$)-1)
2170 LET R$=R$+Q$
2180 IF LEFT$(Q$,1)<>"*" THEN GOTO 2110
2190 CLS
2200 GOSUB 2500
2210 GOTO 2000
```

Function as in Unifile Module 1.1.5.

Commentary

Lines 2140–2150: These two lines illustrate one difference between this program and its predecessor. If the first character of an input consists of the symbol # then the next two characters are taken to be a two digit number between I and 50. The item input is reprinted with the item heading found at that position in A$ preceding it. If no item heading is specified, the item is reprinted with the heading Untyped:.

Line 2180: If the item input was a single asterisk, this is added to the entry but the entry is regarded as finished and control is passed to the insertion module.

Testing Module 1.2.5

You should be able to input item headings and to see them displayed if you tag #NN onto the front of an item you are inputting (where NN is the number of a heading you have entered).

MODULE 1.2.6

```
2500 REM************************
2510 REM PLACE DATA IN FILE
2520 REM************************
2530 IF N<500 THEN GOTO2560
2540 CLS:PRINT @ 14*32+10,"FILE NOW FULL
"
2550 FOR I=1 TO 5000:NEXT I:RETURN
2560 LET POWER=INT (LOG(N-1)/LOG(2))
2570 LET S=2^POWER
2580 LET T$=LEFT$(R$,INSTR(R$,"^")-1)
2590 FOR K=POWER-1 TO 0 STEP -1
2600 LET ST=1:GOSUB 4060:LET U$=LEFT$(B$
(S),ST-1)
2610 IF T$>U$ THEN LET S=S+2^K
2620 IF T$<U$ THEN LET S=S-2^K
2630 IF S>N-1 THEN LET S=N-1
2640 IF S<1 THEN LET S=1
2650 NEXT K
2660 LET ST=1:GOSUB 4060:LET U$=LEFT$(B$
(S),ST-1)
2670 IF T$<U$ THEN LET S=S-1
2680 FOR I=N+1 TO INT(S+2) STEP-1
2690 LET B$(I)=B$(I-1)
2700 NEXT I
2710 LET B$(S+1)=R$
2720 LET N=N+1
2730 RETURN
```

Identical to Unifile Module 1.1.6.

MODULE 1.2.7

```
3500 REM************************
3510 REM SEARCH
3520 REM************************
3530 LET S1=1
3540 IF N=2 THEN RETURN
3550 PRINT @ 11,STRING$(8,140)
3560 PRINT @ 43,CHR$(128)+"SEARCH"+CHR$(
128)
3570 PRINT @ 75,STRING$(8,131)
3580 PRINT ">INPUT SEARCH ITEM",">ENTER
FOR FIRST ITEM ON FILE"
3590 PRINT ">'MMM' FOR MULTIPLE SEARCH"
3600 PRINT STRING$(32,137)
3610 INPUT "ENTER SEARCH COMMAND:";S$
```

```
3620 IF S$="" THEN GOTO 3740
3630 IF S$="MMM" THEN GOTO 3680
3640 FOR S1=1 TO N-2
3650 IF INSTR (B$(S1),S$)<>0 THEN GOTO 3
740
3660 NEXT S1
3670 RETURN
3680 CLS
3690 PRINT:INPUT "NUMBER OF SEARCH ITEMS
:";SEARCH:FOR K=0 TO SEARCH-1:PRINT "SEA
RCH ITEM ";K+1;"    ";:INPUT Q$:LET M$(K)=
Q$:NEXT K
3700 FOR S1=1 TO N-2
3710 FOR K=0 TO SEARCH-1:IF INSTR(B$(S1)
,M$(K))<>0 THEN NEXT K:GOTO 3730
3720 NEXT S1:RETURN
3730 IF INT(S1)=N-1 THEN RETURN
3740 CLS
3750 PRINT "ENTRY ";S1;":-"
3760 LET ST=0
3770 LET TEMP=ST+1
3780 LET S=S1:GOSUB 4060
3790 IF MID$(B$(S1),TEMP,1)="*" THEN GOT
O 3830
3800 IF MID$(B$(S1),TEMP,1)="£" THEN PRI
NT A$(VAL(MID$(B$(S1),TEMP+1,2))-1);":";
:LET C1=3
3810 PRINT MID$(B$(S1),TEMP+C1,ST-TEMP-C
1):LET C1=0
3820 GOTO 3770
3830 LET S1=S1+1
3840 PRINT @ 10*32,"search: ";S$
3850 PRINT "COMMANDS AVAILABLE:"
3860 PRINT ">'ENTER' FOR NEXT ITEM"
3870 PRINT ">'AAA' TO AMEND"
3880 PRINT ">'CCC' TO CONTINUE SEARCH"
3890 PRINT ">'ZZZ' TO QUIT FUNCTION";
3900 INPUT P$
3910 CLS
3920 IF P$="CCC" AND S$="MMM" THEN GOTO
3710
3930 IF P$="CCC" THEN GOTO 3650
3940 IF P$="" THEN GOTO 3730
3950 IF P$="AAA" THEN GOSUB 3010:GOTO 37
30
3960 IF P$="ZZZ" THEN RETURN
3970 LET S1=S1-1:GOTO 3730
```

The same function as the module in Unifile I except that provision is made for multiple searches.

Commentary

Lines 3680–3720: If a multiple search is specified by the input of MMM then the user is asked to specify how many items are to be searched for (up to 20) and then to input them one by one. The file is then scanned for the first of the items specified. If it is found in an entry, then the same entry is scanned for the next search item until it has been scanned for all the search items. If all are present in an entry then the item will be displayed. If no entries are found to contain all the specified search items, the program returns to the menu.

Line 3790: Since there is no regular number of items in an entry the program goes on printing items until it comes across an item which consists of *.

Line 3800: Item headings from A$ are printed where appropriate.

Line 3920: To continue a multiple search we have to jump back into the middle of the loop in order not to reset S1 to 1.

Testing Module 1.2.7

As for the equivalent module in Unifile except that you should be able to specify a multiple search too.

MODULE 1.2.8

```
3000 REM**************************
3010 REM CHANGE ENTRY
3020 REM**************************
3030 LET S1=S1-1
3040 LET R$=""
3050 LET ST=0
3060 CLS:PRINT "ENTRY ";S1;":-"
3070 LET TEMP=ST+1
3080 LET S=S1:GOSUB 4060
3090 IF MID$(B$(S1),TEMP,ST-TEMP)="*" TH
EN LET R$=R$+"*^":GOSUB 4530:GOSUB 2510:
RETURN
3100 IF MID$(B$(S1),TEMP,1)="£" THEN PRI
NT A$(VAL(MID$(B$(S1),TEMP+1,2))-1);":";
:LET C1=3
3110 PRINT MID$(B$(S1),TEMP+C1,ST-TEMP-C
1):LET C1=0
3120 PRINT @ 7*32+12,"amend"
3130 PRINT "COMMANDS AVAILABLE:"
3140 PRINT ">'ENTER' LEAVES ITEM UNCHANG
ED"
3150 PRINT ">NEW ITEM ENDING WITH '>'"
3160 PRINT ">ENTER REPLACEMENT ITEM"
3170 PRINT ">'ZZZ' QUITS WITHOUT CHANGES
"
3180 PRINT ">'DDD' DELETES WHOLE ENTRY"
3190 PRINT ">'RRR' REMOVES THIS ITEM"
3200 GOSUB 4030
3210 CLS
3220 IF Q$="ZZZ^" THEN RETURN
3230 IF Q$="RRR^" THEN GOTO 3060
3240 IF RIGHT$(Q$,2)<>">^" AND Q$<>"^" T
HEN GOTO 3250
3244 IF LEN(R$)+ST-TEMP+1>255 THEN CLS:P
RINT "ENTRY NOW TOO LONG.":FOR I=1 TO 50
00:NEXT:RETURN
3248 LET R$=R$+MID$(B$(S1),TEMP,ST-TEMP+
1)
3250 CLS
3260 IF Q$="^" THEN GOTO 3060
3270 IF Q$="DDD^" THEN GOSUB 4500:RETURN
3280 IF RIGHT$(Q$,2)=">^" THEN LET Q$=LE
FT$(Q$,LEN(Q$)-2):LET Q$=Q$+"^"
3282 IF LEN(R$)+LEN(Q$)>255 THEN PRINT C
LS:PRINT "ENTRY NOW TOO LONG.": FOR I=1
TO 5000:NEXT:RETURN
3290 LET R$=R$+Q$
3300 GOTO 3060
```

A slightly more complex module than the equivalent one in Unifile I, since provision must be made for the insertion and deletion of items, not simply of entries.

Commentary

Lines 3230: Input of RRR does not add the existing item to the new R$ being built up, thus effectively deleting it from the entry.

Line 3240: In the case of an input ending in > the item currently on display is added to the new R$ thus insertions are always after the item currently displayed.

Line 3280: Insertion of items input ending in > .

Testing Module 1.2.8

You should be able to delete entries, to amend items, to delete items and to insert items.

MODULE 1.2.9

```
4500 REM**************************
4510 REM TELESCOPE FILE
4520 REM**************************
4530 FOR I=S1 TO N-1
4540 LET B$(I)=B$(I+1)
4550 NEXT I
4560 LET N=N-1
4570 RETURN
```

Identical to Unifile 1.1.9.

MODULE 1.2.10

```
6000 REM**************************
6010 REM DATA FILES
6020 REM**************************
6030 MOTOR ON:AUDIO ON:INPUT "POSITION T
APE THEN PRESS enter  (MOTOR IS ON):";Q$
6040 MOTOR OFF:INPUT "PLACE RECORDER INT
O CORRECT MODETHEN PRESS enter:";Q$
6050 PRINT:PRINT "FUNCTIONS AVAILABLE:",
"1)SAVE DATA",,"2)LOAD DATA":INPUT "WHIC
H DO YOU REQUIRE:";Q:ON Q GOTO 6070,6150
6060 RETURN
6070 MOTOR ON:FOR I=1 TO 10000:NEXT
6080 OPEN "O",£-1,"DBASE"
6090 FOR I=0 TO 49:PRINT £-1,A$(I):NEXT
6100 PRINT £-1,N
6110 FOR I=1 TO N-2
6120 PRINT £-1,B$(I)
6130 NEXT I
6140 CLOSE £-1:RETURN
6150 PCLEAR 1:CLEAR 20000:DIM A$(49),B$(
499),M$(19)
6160 OPEN "I",£-1,"DBASE"
6170 FOR I=0 TO 49:INPUT £-1,A$(I):NEXT
6180 INPUT £-1,N
6190 FOR I=1 TO N-2
6200 INPUT £-1,B$(I)
6210 NEXT I
6220 CLOSE£-1
6230 LET B$(0)=CHR$(0)+"^"
6240 LET B$(N-1)=CHR$(255)+"^"
6250 GOTO 1000
```

Almost identical to the equivalent module in Unifile I.

Testing Module 1.2.10

If this module is functioning correctly you deserve some kind of medal and the program is ready for use.

Summary

The comments made in relation to Unifile apply equally to this even more substantial program.

In addition, I hope that entering this program has given you some insights into the strengths of modular programming — as writing it did for me. This whole program, when it was first written, took less than a morning for the simple reason that the structure was already laid down in the Unifile I program and its modular form made it clear where changes would have to be made. Provided that a desperate need for space does not lead you to compress everything as much as possible, you will save time and many tears over your life as a programmer by setting out your programs in clearly labelled functional units. Not only does this make the programs readable, it increases the likelihood that you will be able to call the same routine from different parts of the program, eases replacement of functions that you feel you can improve upon later and, not least, makes it a great deal easier to lift whole sections of program out for use in other contexts.

Going further

1) One sophistication that is present on professional database programs is the ability to order a search for entries containing, say, four out of eight specified search items. It should not be difficult to include such a provision in the present program.
2) The developments suggested for Unifile I would be equally applicable to this program.

CHAPTER 2
Managing your money

Where microcomputers are set to work in the home, it is most often in handling family finance, and for that reason we turn our attention in this chapter to three financial programs. Their interest, however is not limited to those who wish to use their Dragon to supervise their finances, for in discussing the programs we shall be examining problems common to all programs which handle fairly large bodies of numeric data.

2.1 BANKER

This program is a neat tool which allows you to keep your financial records in much the same form as a bank statement. It deals with recurring payments, both regular and irregular and inserts them into each monthly statement on the day on which they occur.

The program is a relatively simple one compared to what has gone before but it is worth pointing out that it is not as uncomplicated as it looks at first sight, since in this program, for the first time, we make use of a considerable proportion of multi-statement lines. Without them the program would appear considerably longer.

One of the points to watch out for in entering this, or any other program which uses multi-statement lines is the behaviour of IF statements. Such statements are capable of creating havoc if used improperly in multi-statement lines, creating program bugs which are extremely difficult to trace. Equally, multi-statement lines can be used to increase the effectiveness of IF statements by virtue of the fact that if the condition specified by an IF statement is not met, the program does not simply skip over that part of the line directly tied to the IF statement, it skips over the whole of the rest of the line.

In other words, any statements after the IF statement will only be executed if the IF statement is true. This is so different from the behaviour of single- statement lines that it is easy to be caught out by it.

The advantage of all of this is that it provides a form of automatic and elegant GOTO that you do not even have to specify. If you have a series of 10 operations that will be performed together provided, say, that C = 1 at some point, then with single statement lines you would have to place an IF statement at the beginning of the section to specify a jump past the 10 operations if C is not equal to 1. It works but at the same time it feels messy and it is difficult to read a program in which there are a lot of such jumps.

With multi-statement lines, however, you could begin a single line with IFC = I and follow it with the 10 operations. Not only would this work and save memory, it would actually make the program more readable, since it would be immediately clear that the 10 operations form a logical unit.

MODULE 2.1.1

```
7000 REM*************************
7010 REM FORMAT TITLES
7020 REM*************************
7030 LET P2=14-INT(LEN(F$)/2)
7040 PRINT @ 32*P1+P2,STRING$(LEN(F$)+2,
150)
7050 PRINT @ 32*(P1+1)+P2,CHR$(150)+F$+C
HR$(150)
7060 PRINT @ 32*(P1+2)+P2,STRING$(LEN(F$
)+2,150)
7070 RETURN
```

In entering the last two programs, you may have found the lines needed to decorate the program titles a little tedious to enter. This module is the answer. It creates the same kind of decorative box around a word or phrase but once entered it can be applied to a variety of different titles at different points during the execution of the program. All that needs to be specified before the module is called up is the line on which the title is to be printed and the title or phrase itself. Clearly the graphics characters specified in lines 7040–7060 can be altered to taste.

MODULE 2.1.2

```
1000 REM*************************
1010 REM MENU
1020 REM ***********************
1030 CLS:LET F$="BANKER":LET P1=1:GOSUB
7000
1040 PRINT:PRINT "COMMANDS AVAILABLE:"
1050 PRINT:PRINT"   1)NEW PAYMENTS"
1060 PRINT "   2)EXAMINE/DELETE PAYMENTS"
1070 PRINT "   3)PRINT STATEMENT"
1080 PRINT "   4)DATA FILES"
1090 PRINT "   5)INITIALISE"
1100 PRINT "   6)STOP"
1110 PRINT:INPUT "WHICH DO YOU REQUIRE:"
;Z:CLS
1120 IF Z<>5 AND IN=0 THEN CLS:PRINT @7*
32,"PROGRAM NOT INITIALISED YET.":FOR I=
1 TO 5000:NEXT:GOTO 1000
1130 IF PAYMENTS=0 AND (Z=2 OR Z=3) THEN
 CLS:PRINT:PRINT "SORRY, NO DATA YET.":F
OR I=1 TO 5000:NEXT I:GOTO 1000
1140 ON Z GOSUB 3000,4000,5000,6000,2000
,1160
1150 GOTO 1000
1160 CLS:LET F$="BANKER":LET P1=1:GOSUB
7000
1170 LET F$="CLOSED FOR BUSINESS":LET P1
=7:GOSUB 7000:STOP
```

A standard menu module but its execution should provide ample proof of the effectiveness of the previous module in brightening up the presentation of the program.

MODULE 2.1.3

```
2000 REM***************************
2010 REM VARIABLES
2020 REM***************************
2030 PCLEAR 1:CLEAR 10000:LET FLAG=0
2040 LET IN=1
2050 DIM A$(99,1),A(99,1):LET A(0,1)=999
2060 DIM MONTH$(12):RESTORE:FOR I=0 TO 1
1:READ MONTH$(I):NEXT I
2070 DATA "JANUARY","FEBRUARY","MARCH","
APRIL","MAY","JUNE","JULY","AUGUST","SEP
TEMBER","OCTOBER","NOVEMBER","DECEMBER"
2080 IF FLAG=0 THEN GOTO 1000 ELSE GOTO
6140
```

This module initialises the program variables.

Commentary

Line 2030: The variable FLAG is used in an interesting way here — to make up for the lack of GOSUB...RETURN when the memory has been cleared. Normally this module is called from the main menu but on some occasions it is called from data file module, before loading data from tape. In that case, the data file module clears the memory first and then calls this module from line 2040, first having set FLAG to 1. When program execution reaches line 2080, FLAG can be easily used to determine whether the program is to return to the main menu or to the data file module.

Line 2040: This program, like many others, is capable of accepting a few entries without the variables having been properly initialised. After a few entries the program stops — a frustrating experience. The variable IN (initialised) is therefore used in the main menu to determine whether this module has yet been called. If IN is 1 at line 1120 then all the program functions are available, otherwise you can only call this module.

Line 2060: DATA statements are not only useful for storing complex facts and figures. Here READ, DATA and RESTORE are used to simplify the placing of the names of the months of the year into an array for later use.

Testing Module 2.1.3

This module can only be properly tested when other modules begin to call upon the variables it has set up but you might like to check that the months of the year are stored in MONTH$.

MODULE 2.1.4

```
3000 REM***************************
3010 REM ENTER NEW ITEMS
3020 REM***************************
3030 CLS:LET F$="NEW ITEMS":LET P1=0:GOS
UB 7000
3040 PRINT "  1)CREDIT",,"  2)DEBIT":INP
UT "WHICH DO YOU REQUIRE:";CD:LET CD=CD-
1
3050 PRINT @ 7*32,"";:INPUT "NAME OF PAY
MENT:";Q$
```

```
3060 PRINT @ 9*32,"";:INPUT "AMOUNT:";Q
3070 PRINT @ 11*32,"";:INPUT "MONTHS (E.
G. 01040710):";R$
3080 PRINT @11*32+24,"";:FOR I=1 TO LEN(
R$) STEP 2:PRINT MID$(R$,I,2);"/";:NEXT
I
3090 PRINT @ 13*32,"";:INPUT "DAY OF PAY
MENT:";S
3100 PRINT @ 15*32,"";:INPUT "ARE THESE
CORRECT (Y/N):";T$:IF T$="N" THEN CLS:GO
SUB 3000
3110 LET PAYMENTS=PAYMENTS+1:FOR J=PAYME
NTS-1 TO 0 STEP -1
3120 IF S<A(J,1) THEN FOR K=0 TO 1:LET A
$(J+1,K)=A$(J,K):LET A(J+1,K)=A(J,K):NEX
T K:NEXT J
3130 LET J=J+1
3140 LET A$(J,1)=STRING$(12,"0"):FOR I=1
 TO LEN(R$) STEP 2:MID$(A$(J,1),VAL(MID$
(R$,I,2)),1)="1":NEXT I
3150 LET A$(J,0)=Q$:LET A(J,0)=Q
3160 LET A(J,1)=S
3170 IF CD=1 THEN LET A(J,0)=A(J,0)*-1
3180 RETURN
```

The purpose of this module is to accept new payment items, whether credit or debit, and place them into their correct place in the file of payments.

Commentary

Line 3040: The only distinction made between credit items and debit items is the setting of the variable CD to 0 or 1.

Line 3070–3080: The user is required to input months in which the particular payment will be made in a continuous string of numbers, two digits per month. This is simple and fast but, since it can sometimes lead to mistakes on input, the program reprints the months apparently entered with a separator between each so that the user can check the input.

Lines 3110–3130: The file of payments is kept in order of the day of the month on which payment is made. In this section the program scans the file from the highest numbered day downwards. If the day of the current entry is lower than the day of the item in the file then the whole file from that point is moved up one place and the next item down the file is examined. In this way, by the time the program finds the correct position in the file for the new entry, a place has already been made for it.

Line 3140: The month(s) in which the payment will actually be made are recorded in a string of 12 characters. The string begins as 12 0s and then the positions corresponding to the desired months are set to 1. If you wish to minimise the amount of space taken by the program it is perfectly feasible to use only two bytes for this, setting individual bits to represent the desired months — the trade off being that more time is required to translate this kind of representation.

Line 3170: If the item is meant to be a debit (i.e. a payment out), then the amount of the item is multiplied by −1.

Testing Module 2.1.4

You should now be able to input payment items together with their associated months and days of payment. Though you cannot yet display them, you can stop the program and check that the relevant sections of the arrays A and A$ contain the payment details in order of day of payment. At A (0,PAYMENTS) should be 999, a dummy entry to ensure correct insertion of the first item.

MODULE 2.1.5

```
5000 REM***************************
5010 REM COMPILE STATEMENT
5020 REM***************************
5030 LET SUM=0
5040 LET F$="STATEMENT":LET P1=1:GOSUB 7
000
5050 INPUT "NUMBER OF MONTH FOR STATEMEN
T:";Q
5060 FOR Q1=1 TO Q-1:FOR I=0 TO PAYMENTS
-1:IF MID$(A$(I,1),Q1,1)<>"1" THEN GOTO
5080
5070 LET SUM=SUM+A(I,0)
5080 NEXT I:NEXT Q1
5090 CLS:LET F$=MONTH$(Q-1):LET P1=0:GOS
UB 7000
5100 PRINT "BALANCE C/F";:PRINT TAB(24)"
";:IF SUM<0 THEN PRINT CHR$(191); ELSE P
RINT CHR$(159);
5110 PRINT USING "££££.££";SUM;
5120 FOR I=0 TO PAYMENTS-1
5130 IF MID$(A$(I,1),Q,1)<>"1" THEN GOTO
 5210
5140 PRINT USING "£££";A(I,1);:IF A(I,0)
<0 THEN PRINT CHR$(191);ELSE PRINT CHR$(
159);
5150 PRINT A$(I,0);
5160 PRINT TAB(16);:IF A(I,0)<0 THEN PRI
NT CHR$(191); ELSE PRINT CHR$(159);
5170 PRINT USING "££££.££";A(I,0);
5180 LET SUM=SUM+A(I,0):IF SUM<0 THEN PR
INT CHR$(191); ELSE PRINT CHR$(159);
5190 PRINT USING "££££.££";SUM;
5200 IF INKEY$="" THEN GOTO 5200
5210 NEXT I: PRINT:INPUT "enter TO CONTI
NUE";X
5220 RETURN
```

This module prints out a statement for any particular month specified. A balance is carried forward from previous months in the same calendar year.

Commentary

Lines 5060–5080: These two loops scan each entry in the file for any payments occurring in the months prior to the month specified for the statement. Any such payments are added to the variable SUM. It may comfort you to know that when this program was first entered the variable used was named TOTAL and the line gave a syntax error report. After a long time spent cursing the Dragon I realized that variable names which corresponded to the first two letters of Basic words confused the poor beast.

Lines 5100–5110: Unfortunately the Dragon cannot print text in different colours, or debit items could be printed in red. To make up for this a yellow square is used to delineate columns for credit items and a red square for debit items, this particular item being the balance carried forward.

Lines 5120–5210: This loop prints the details of day of payment, name of payment, amount of payment and the running total of the account, provided that the relevant character in A$(1) is I rather than 0. The account makes use of the PRINT USING command to ensure that pence are correctly printed (otherwise £10.70 would be rendered 10.7). The presentation of the account is in columns and the squares making up the column are red or yellow according to whether a debit or credit item is being printed. The coloured squares are obtained by reference to the chart on page 138 of the Dragon manual.
Note that items are printed one by one and to obtain the next item it is necessary to depress ENTER: the purpose of this is to prevent items being printed and scrolled off the screen before they can be examined if there is more than one screenful of data.

Testing Module 2.1.5

You should now be able to enter some data and obtain a statement for any month. In the case of months for which there are no payments registered the balance carried forward will be printed. After the last item of the month's account, pressing ENTER again will return to the menu.

MODULE 2.1.6

```
4000 REM****************************
4010 REM DELETE PAYMENTS
4020 REM****************************
4030 FOR I=0 TO PAYMENTS-1:CLS
4040 PRINT:PRINT "PAYMENT:";A$(I,0)
4050 PRINT:PRINT "AMOUNT:";A(I,0)
4060 PRINT:PRINT "MONTHS:";
4070 FOR J=1 TO 12: IF MID$(A$(I,1),J,1)
="1" THEN PRINT STR$(J);:PRINT "/";
4080 NEXT J
4090 PRINT:PRINT:PRINT "DAY OF PAYMENT:"
;A(I,1)
4100 PRINT:PRINT "COMMANDS:",,"   1)'DDD
' DELETE",,"   2)'ZZZ' QUIT",,"   3)'ENTE
R' FOR NEXT ITEM"
4110 INPUT Q$:IF Q$="DDD" THEN FOR J=I T
O PAYMENTS-1:FOR K=0 TO 1:LET A$(J,K)=A$
(J+1,K):LET A(J,K)=A(J+1,K):NEXT K:NEXT
J:LET PAYMENTS=PAYMENTS-1:RETURN
4120 IF Q$="" THEN NEXT I
4130 RETURN
```

This module allows individual payments to be examined, together with their associated data and, if necessary, to be deleted from the file.

Commentary

Line 4110: An example of the usefulness of IF statements when properly combined with multi-statement lines. This whole section of eight consecutive statements is only executed if the user inputs DDD.

Testing Module 2.1.6

You should now be able to delete items from the file.

MODULE 2.1.7

```
6000 REM****************************
6010 REM DATA FILES
6020 REM****************************
6030 MOTOR ON:AUDIO ON:PRINT:INPUT "POSI
TION TAPE THEN PRESS enter  (MOTOR IS ON
)";Q$:MOTOR OFF
6040 PRINT:INPUT "PLACE RECORDER IN CORR
ECT MODE  THEN PRESS enter";Q$
6050 PRINT:PRINT "FUNCTIONS AVAILABLE:",
"1)SAVE DATA",,"2)LOAD DATA":INPUT "WHIC
H DO YOU REQUIRE:";Q:ON Q GOTO 6070,6130
6060 RETURN
6070 MOTOR ON:FOR I=1 TO 10000:NEXT I
6080 OPEN"O",£-1,"BANKER"
6090 PRINT£-1,PAYMENTS
6100 FOR I=0 TO PAYMENTS:PRINT£-1,A$(I,0
),A$(I,1),A(I,0),A(I,1):NEXT I
6110 CLOSE£-1
6120 RETURN
6130 PCLEAR 1:CLEAR 10000:LET FLAG=1:GOT
O 2040
6140 OPEN "I",£-1,"BANKER"
6150 INPUT£-1,PAYMENTS
6160 FOR I=0 TO PAYMENTS:INPUT£-1,A$(I,0
),A$(I,1),A(I,0),A(I,1)
6170 NEXT I
6180 CLOSE£-1
6190 GOTO 1000
```

A standard data-file module.

Commentary

Line 6130: The use of the variable FLAG has already been discussed but it is worth pointing out that the only reason that the initialisation module is actually called is the loading of the month names into MONTH$. Other than this it is often more economical simply to dimension the arrays in the data-file module itself.

Testing Module 2.1.7

You should now be able to input data, store it on tape and recall it for subsequent use. If this module functions properly the program is ready for use.

Summary

This straightforward program raises some interesting questions about the degree of sophistication required to make a program useful. Inputting the months during which a particular payment is to be made is, in some ways, rather crude compared to specifying whether the payment is to be made quarterly or annually or whatever and letting the program insert the payment in the relevant months. Such an added facility would be easily possible but it would add considerably to the length of the program and it would reduce the degree of flexibility inherent in simply typing in the months. When designing a program you will need to be constantly aware of this tension between what is worth doing automatically and what it is better to allow the user to do — the answer may well vary from user to user but complexity simply for the sake of it can be costly in terms of memory and can actually reduce the usefulness of a program.

Going Further

1) The deletion module is extremely crude in that it only allows the user to page through the items one by one. What about adding a facility which would allow the user to specify a jump backwards or forwards in the file, thus making it easier to access items towards the end of the file.
2) The program is set up on the basis that the financial year begins in January. It would be more useful if the user could specify when the financial year should begin and the program would then scan the entries from that time to produce the balance carried forward in the printed account. This would allow the user to keep accounts for the last three months, say, even in January and February.

2.2 ACCOUNTANT

This program won't actually cook the books for you, but it will make them very much easier to keep and present them in a neat format whenever you wish, with provision for single items, main headings and sub-headings in the printing of the actual accounts.

MODULE 2.2.1

```
7000 REM************************
7010 REM FORMAT TITLES
7020 REM************************
7030 LET P2=14-INT(LEN(F$)/2)
7040 PRINT @ 32*P1+P2,STRING$(LEN(F$)+2,
CHR$(147))
7050 PRINT @ 32*(P1+1)+P2,CHR$(159);
7060 PRINT F$;CHR$(159)
7070 PRINT @ 32*(P1+2)+P2,STRING$(LEN(F$
)+2,CHR$(156))
7080 RETURN
```

Standard title format module.

MODULE 2.2.2

```
6000 REM************************
6010 REM DATA FILES
6020 REM ***********************
6030 AUDIO ON:MOTOR ON:INPUT "POSITION T
APE, THEN PRESS enter.(MOTOR IS ON)";Q$
6050 MOTOR OFF:INPUT "PLACE RECORDER INT
O CORRECT MODETHEN PRESS enter.";Q$
6060 PRINT "1)SAVE DATA",,"2)LOAD DATA":
INPUT "WHICH DO YOU REQUIRE:";Q
6070 ON Q GOTO 6090,6170
6080 RETURN
6090 MOTOR ON:FOR I=1 TO 10000:NEXT I
6100 OPEN"O",£-1,"ACCOUNTS"
6110 FOR I=0 TO 1
6120 PRINT£-1,C(I)
6130 FOR J=0 TO C(I)-1
6140 PRINT£-1,A$(I,J),A(I,J)
6150 NEXT J,I:CLOSE£-1:RETURN
```

```
6170  OPEN  "I",£-1,"ACCOUNTS"
6180  FOR  I=0  TO  1
6190  INPUT£-1,C(I)
6200  FOR  J=0  TO  C(I)-1
6210  INPUT£-1,A$(I,J),A(I,J)
6220  NEXT  J,I:CLOSE  £-1:RETURN
```

This is entered at this point in order to point out that the easiest way of entering this, and the previous module, is simply to enter a program which already contains them and then making any necessary changes to the variables which are to be saved. Since you now have a program, Banker, with both modules, you can save yourself considerable time by loading that and deleting up to line 5999 (DEL–5999).

You may also find it an advantage to have entered this particular module first in that once you have successfully entered the modules which load data into the main file, you will be in a position to save some data so that when data is lost with the correction of mistakes or the entry of new lines (unfortunate habit, that), you can simply reload from tape rather than type it all in again.

MODULE 2.2.3

```
1000  REM************************
1010  REM  MENU
1020  REM************************
1030  CLS:LET  F$="ACCOUNTANT":LET  P1=1:GO
SUB  7000
1040  PRINT:PRINT  "COMMANDS  AVAILABLE:"
1050  PRINT  "  1)INPUT  NEW  HEADINGS"
1060  PRINT  "  2)CHANGE  AMOUNTS/DELETE  ITE
MS"
1070  PRINT  "  3)PRINT  ACCOUNTS"
1080  PRINT  "  4)INITIALISE  ACCOUNTS"
1090  PRINT  "  5)DATA  FILES"
1100  PRINT  "  6)STOP"
1110  PRINT:INPUT  "WHICH  DO  YOU  REQUIRE";
Z:CLS
1120  IF  Z<4  AND  Z>0  THEN  GOSUB  2000
1130  CLS
1140  ON  Z  GOSUB  2500,4000,5000,1500,6000
,1160
1150  CLS:GOTO  1000
1160  CLS:LET  F$="ACCOUNTANT":LET  P1=6:GO
SUB  7000
1170  END
```

Standard menu module.

MODULE 2.2.4

```
1500  REM*************************
1510  REM  INITIALISE
1520  REM************************
1530  PCLEAR1:CLEAR  10000
1540  DIM  A$(2,100),A(2,100)
1550  GOTO  1000
```

Initialises program arrays. Note that this module must be called before data files can be loaded from tape since the data file module in this program does not clear the memory and initialises the arrays.

MODULE 2.2.5

```
2000 REM***************************
2010 REM CREDIT/DEBIT
2020 REM***************************
2030 PRINT @ 7*32+3,"DO YOU WANT: 1)CRED
IT"
2040 PRINT TAB(16)"2)DEBIT"
2050 INPUT CD:LET CD=CD-1:RETURN
```

Unlike the previous program, there are a number of functions in this program which require to know whether the debit or credit side of the accounts is being addressed. The input which specifies this is therefore made a separate module.

Testing Module 2.2.5

Before moving on to the main body of the program you may wish to check that the modules entered so far are functioning correctly. The menu should work for the data file option and should call up the present module for any function from 1 to 3. Menu function 6 should also be available.

MODULE 2.2.6

```
2500 REM***************************
2510 REM INPUT ITEMS
2520 REM***************************
2530 LET F$="NEW ITEMS":LET P1=1:GOSUB 7
000
2540 PRINT "IS THE ITEM: 1)A SINGLE ITEM
"
2550 PRINT TAB(13)"2) A MAIN HEADING"
2560 PRINT TAB(13)"3) A SUB-HEADING"
2570 INPUT"'0' TO QUIT FUNCTION";TYPE
2580 IF TYPE=0 THEN RETURN
2590 IF TYPE=3 THEN GOTO 3500
```

The input of items to the program is done under three types of heading: main headings, sub-headings and single items. Their nature will be explained under the sections that refer to them. The purpose of this module is simply to have the user specify which is about to be input.

MODULE 2.2.7

```
3000 REM***************************
3010 REM SINGLE ITEM OR MAIN
 HEADING
3020 REM***************************
3030 LET Q=0
3040 PRINT @ 9*32,"";:INPUT "NAME OF ITE
M:";Q$
3050 IF TYPE<>2 THEN PRINT @ 11*32,"";:I
NPUT "AMOUNT FOR ITEM:";Q
3060 PRINT @ 13*32,"";:INPUT "IS THIS CO
RRECT? (Y/N)";R$
3070 IF R$<>"Y" THEN FOR I=9 TO 13:PRINT
 @ 32*I,"":NEXT:GOTO 3040
3080 IF TYPE=2 THEN LET Q$="*"+Q$ ELSE L
ET Q$=" "+Q$
3090 LET A$(CD,C(CD))=Q$
3100 LET A(CD,C(CD))=Q
3110 LET C(CD)=C(CD)+1
3120 CLS:GOTO 2520
```

This module accepts the input of two different types of items, main headings and single items. Main headings are general categories which will have no amounts attached to them in the accounts, they will serve as 'paragraph headings' for a list of items which fall under that particular heading. In household accounts, a main heading might be CAR and it might be followed by sub-headings relating to tax, insurance, maintenance etc. Single items are items which are neither main headings nor items which fall into the groups which the main headings label — they are 'one off' items.

Commentary

Line 3050: Main headings do not have amounts directly attached to them in the accounts.

Line 3070: If details entered are incorrect, this line clears only that part of the screen containing those details — otherwise we would have to return to the previous module to reprint the prompts on the top half of the screen.

Line 3080: The three types of item are labelled in the file which contains the accounts by single characters which are tagged onto the front of the item names (these characters are never printed, they are there for the program's use only). The symbol for a main heading is * and that for a single item is a space. Note how the use of ELSE saves us another IF statement.

Lines 3090 – 3110: The names of items (and the identifying tags) are stored in the array A$, on the credit or the debit side according to the value of CD. Similarly the value attached to the item is stored in the array A. The number of items on each side of the arrays is recorded in another array named C. If you are observant you will have noted that nowhere did we dimension an array called C. Here we are making use of the fact that whenever an array is referred to (and the Dragon can tell an array is being referred to because a subscript will be tagged onto the end of the name) the Dragon assumes until it is told otherwise that the array has 10 elements. So if you want to use an array with less than 10 elements you don't need to declare it in a DIM statement.

Testing Module 2.2.7

You should now be able to input main headings and single items and main headings. You cannot easily display what you have input but you can check in direct mode that the item names have been stored, that they have the correct indicator tag preceding them and that the correct value is attached to them in the corresponding element of the array A (the correct value for a main heading is zero).

MODULE 2.2.8

```
3500 REM****************************
3510 REM SUB-HEADING
3520 REM****************************
3530 PRINT @ 9*32,"";:INPUT "NAME OF MAI
N HEADING:";Q$
3540 LET Q$="*"+Q$
3550 FOR I=0 TO C(CD)-1:IF A$(CD,I)<>Q$
THEN NEXT I:PRINT:PRINT "SORRY, NO HEADI
NG OF THAT NAME.":FOR I= 1 TO 5000:NEXT:
RETURN
3560 LET PLACE=I+1
3570 PRINT @ 11*32,"";:INPUT "NAME OF SU
B-HEADING:";Q$
3580 PRINT @ 13*32,"";:INPUT "AMOUNT FOR
 SUB-HEADING:";Q
3590 PRINT:INPUT "ARE THESE CORRECT (Y/N
):";R$
3600 IF R$<>"Y" THEN FOR I= 9 TO 13:PRIN
T 32*I,"":NEXT:GOTO 3570
3610 LET Q$="$"+Q$
3620 FOR I=C(CD)+1 TO PLACE+1 STEP-1
3630 LET A$(CD,I)=A$(CD,I-1)
3640 LET A(CD,I)=A(CD,I-1)
3650 NEXT I
3660 LET A$(CD,PLACE)=Q$
3670 LET A(CD,PLACE)=Q
3680 LET C(CD)=C(CD)+1
3690 CLS:GOTO 2520
```

This module accepts the third category of item which the program recognises — sub-headings.

Commentary

Lines 3530–3550: In this section the user first inputs the name of the relevant main-heading and to this is added the identifying tag *, which is how the main heading will be recorded in the file. The program now works through the file comparing the specified main heading with those that are actually stored already. Note that in checking that something is present in a file it is always easier, in fact, to check that it is not. For this purpose all that is needed is a one line loop as line 3550. If you remember what was said earlier about the use of IF statements in multi-statement lines you will realise that the end of this loop will only be reached if the item being searched for is not present. If the item is found then the program execution will automatically default to the next line since the condition attached to the IF statement has not been fulfilled.

Line 3560: If the main-heading is found, the variable PLACE is set to the position following it in the file — this will be the position of the sub-heading.

Lines 3620–3680: The items in the file, from position PLACE upwards, are shifted by one space to make room for the next item and the new item is inserted into position PLACE, with the relevant side of C being incremented to record the new item.

Testing Module 2.2.8

As with the last module, it is difficult to test fully until the display module is entered, but you should be able to enter sub-headings, check that main-headings are present and to examine in direct mode that the data has been placed into the file correctly. If all seems well then it would be advisable to save a specimen set of accounts onto tape to save time in testing the display and deletion modules.

MODULE 2.2.9

```
5000 REM***************************
5010 REM PRINT ACCOUNTS
5020 REM***************************
5030 IF CD=0 THEN LET F$="CREDIT" ELSE L
ET F$="DEBIT":LET P1=0:GOSUB 7000
5040 LET TTOTAL=0
5050 LET STOTAL=0
5060 FOR I=0 TO C(CD)-1
5070 LET TTOTAL=TTOTAL+A(CD,I)
5080 IF LEFT$(A$(CD,I),1)="$" THEN PRINT
 TAB(2);
5090 PRINT USING "%             %";MID$(
A$(CD,I),2);
5100 IF LEFT$(A$(CD,I),1)="*" THEN PRINT
:GOTO 5150
5110 IF A(CD,I)=0 THEN GOTO 5150
5120 IF LEFT$(A$(CD,I),1)<>"$" THEN PRIN
T TAB(25); ELSE PRINT TAB(18);
5130 PRINT USING "££££.££";A(CD,I);
5140 IF LEFT$(A$(CD,I),1)="$" THEN LET S
TOTAL=STOTAL+A(CD,I):PRINT
5150 IF STOTAL=0 OR LEFT$(A$(CD,I+1),1)=
"$" THEN GOTO 5200
5160 PRINT TAB(18)STRING$(7,"~")
5170 PRINT TAB(25);
5180 PRINT USING "££££.££";STOTAL;
5190 LET STOTAL=0
5200 IF INKEY$="" THEN GOTO 5200 ELSE NE
XT I
5210 PRINT TAB(25)STRING$(7,"~");
5220 PRINT "TOTAL:";
5230 PRINT TAB(25);
5240 PRINT USING "££££.££";TTOTAL
5250 INPUT "PRESS 'ENTER' TO QUIT:";Q$
5260 RETURN
```

Most display modules for complex data are themselves complex, and this one is no exception. The reason for this is that rather than working on an elegant and simple set of principles which can be easily programmed into one or two lines, such modules work with a mass of different rules and qualifications which reflect the way in which you wish to transform the data into a display — some placed here, some there, some inset, some on a new line, all according to a variety of different conditions.

Commentary

Line 5030: The heading printed depends upon the value of CD.

Line 5070: Throughout the printing of the accounts, a running total of the sums printed so far is stored in the variable TTOTAL.

Lines 5080–5090: If the item to be printed is a sub-heading (identifying tag $) then it is inset two spaces before the name of the item is printed. Note the use of the PRINT USING % formatting command here: this prints the string specified in a space as long as the total distance from the first % to the second % (i.e. the number of spaces plus 2). If the string is too long it is truncated, if it is too short it is padded out with spaces.

Line 5100: If the item is a main heading a spacing line is printed to make it stand out.

Line 5120: The amounts associated with sub-totals are printed at column 18 on the screen, other amounts are printed at column 25.

Line 5130: This PRINT USING command means that pence will always be printed but that the program can only handle amounts up to £9,999.99. Its effect on spacing is similar to the previous PRINT USING command except that the padding added if the number is too short is added to the beginning of the number being printed.

Line 5140: If the item being printed is a sub-heading, then the amount associated with it is added to the variable STOTAL, which serves as a record of the sum of the items under any one main heading.

Lines 5160–5180: At the end of the group represented by a main heading, the sub-total is printed in the main column.

Line 5200: Items are printed one at a time in response to the pressing of any key.

Lines 5210–5240: When all items have been printed, the overall total for the particular side of the accounts in question is printed underneath the main column.

Testing Module 2.2.9

Recalling the set of data which you stored on tape, you should now be able to see it presented in the format described above by calling this module. Don't forget to initialise the program before trying to load the data!

MODULE 2.2.10

```
4000 REM***************************
4010 REM CHANGES AND DELETIONS
4020 REM***************************
4030 FOR I=0 TO C(CD)-1
4040 LET F$="CHANGE OR DELETE":LET P1=0:
GOSUB 7000
4050 IF LEFT$(A$(CD,I),1)<>"$" THEN PRIN
T @ 96,""
4060 PRINT @ 128,""
4070 IF LEFT$(A$(CD,I),1)<>"$" THEN PRIN
T @ 96,MID$(A$(CD,I),2);
```

```
4080 IF LEFT$(A$(CD,I),1)="$" THEN PRINT
 @ 128, MID$(A$(CD,I),2);
4090 IF A(CD,I)=0 THEN GOTO 4120
4100 PRINT TAB(16);
4110 PRINT USING "££££.££";A(CD,I)
4120 PRINT @ 10*32,">'ENTER'=NEXT ITEM"
4130 PRINT ">'CCC'=CHANGE AMOUNT"
4140 PRINT ">'ZZZ'=QUIT FUNCTION"
4150 PRINT ">'DDD'=DELETE ITEM"
4160 INPUT Q$
4170 IF Q$="ZZZ" THEN RETURN
4180 IF Q$="DDD" THEN GOSUB 4500:RETURN
4190 IF Q$=""THEN GOTO 4270
4200 FOR J=10 TO 14:PRINT @ J*32,"":NEXT
4210 PRINT @ 12*32,"";:INPUT "AMOUNT TO
BE ADDED:";Q
4220 PRINT:INPUT "IS THAT CORRECT (Y/N)"
;R$
4230 FOR J=11 TO 14:PRINT @ J*32,"":NEXT
4240 IF R$="N" THEN GOTO 4210
4250 LET A(CD,I)=A(CD,I)+Q
4260 GOTO 4040
4270 NEXT I:RETURN
```

The purpose of this module is to allow the user to examine, change or delete individual items. Once having entered an item, such as 'mortgage' in domestic accounts, this module would be used to add any subsequent payments (or to reduce the payments) rather than entering the item afresh for each occurrence.

Commentary

Lines 4050–4110: The item is displayed, together with its main heading if it is a sub-heading.

Lines 4200–4250: Changes to the amount associated with an item are simply made by inputting the extra sum to be added (or subtracted) from the existing amount.

Testing Module 2.2.10

You should now be able to display the items in the file one by one and to amend the amounts associated with each.

MODULE 2.2.11

```
4500 REM*************************
4510 REM DELETE ITEM
4520 REM*************************
4530 LET PLACE=I:LET GROUP=1
4540 IF LEFT$(A$(CD,PLACE),1)<>"*" THEN
GOTO 4580
4550 LET GROUP=0
4560 LET GROUP=GROUP+1
4570 IF LEFT$(A$(CD,PLACE+GROUP), 1)="$"
THEN GOTO 4560
4580 FOR K=PLACE TO C(CD)-GROUP-1
4590 LET A(CD,K)=A(CD,K+GROUP)
4600 LET A$(CD,K)=A$(CD,K+GROUP)
4610 NEXT K
4620 LET C(CD)=C(CD)-GROUP
4630 RETURN
```

The function of this module is to carry out the deletion of a particular item when it is specified in the previous module.

Commentary

Lines 4530–4620: The reason this module is more complicated than other deletion modules we have entered is represented by the variable GROUP, which is needed to record the number of items that have to be deleted at any one time. The reason that there may be more than one item to be deleted is that the user may have specified the deletion of a main heading, in which case all the sub-headings associated with that main heading are also to be deleted.

Testing Module 2.2.11

If this module is successful in deleting items, including main headings and all the associated sub-headings, then the program is correctly entered and ready for use.

Summary

By now you should be becoming familiar with the techniques involved in adding and deleting items to files without disturbing the overall orderliness. What may have been new in this program is the sheer fiddliness of correctly formatting large amounts of data on the screen. It is worth reviewing the methods used before you continue, since in the next program we shall be displaying far more complex data than anything met here.

Going Further

1) One useful extra facility would be a simple module to calculate and perhaps even print the balance between the two sides of the account.
2) As in the previous program, if you are going to want to store very large numbers of items in this program then you will also want to change the module which displays the items one by one, to allow a more rapid movement through the file.

2.3 BUDGET

Finally in this trio of financial programs we turn to the most complex program you will encounter in this book. Entitled Budget it is a powerful and flexible financial tool which enables the user to plan finances over a 12 month period and to examine the consequences of 'what. . . if' decisions about income and expenditure. Intelligently used, it can provide some surprising insights into a family's finances over the year to come — quite apart from illustrating the problems of working with large bodies of numeric data. The arrays used by the program contain some 650 separate numeric values.

MODULE 2.3.1

```
6000 REM***************************
6010 REM DATA FILES
6020 REM***************************
6030 AUDIO ON:MOTOR ON:INPUT "POSITION T
APE THEN PRESS enter (MOTOR IS ON)";Q$:
MOTOR OFF
6040 INPUT "PUT RECORDER INTO CORRECT MO
DE  THEN PRESS enter";Q$
6050 PRINT "FUNCTIONS AVAILABLE:","1)SAV
E DATA",,"2)LOAD DATA":INPUT "WHICH DO Y
OU REQUIRE ";Q:ON Q GOTO 6070,6270
6060 RETURN
6070 MOTOR ON:FOR I=1 TO 10000:NEXT I:OP
EN "O",£-1,"BUDGET"
6080 PRINT£-1,MO,Y
6090 FOR I=0 TO 11
6100 PRINT £-1,INCOME(0,I),INCOME(1,I),S
UPP(0,I),SUPP(1,I)
6110 NEXT I
6120 PRINT£-1,N(0),N(1)
6130 FOR I=0 TO N(0)-1
6140 PRINT£-1,PAYMENT$(0,I)
6150 FOR J=0 TO 11
6160 PRINT£-1,PAYMENT(0,I,J)
6170 NEXT J
6180 NEXT I
6190 FOR I=0 TO N(1)-1
6200 PRINT£-1,PAYMENT$(1,I)
6210 FOR J=0 TO 11
6220 PRINT£-1,PAYMENTS(1,I,J)
6230 NEXT J
6240 NEXT I
6250 CLOSE£-1
6260 RETURN
6270 CLEAR 5000:PCLEAR 1:LET FLAG=1:GOTO
 1540
6280 OPEN "I",£-1,"BUDGET"
6290 INPUT£-1,MO,Y
6300 FOR I=0 TO 11
6310 INPUT£-1,INCOME(0,I),INCOME(1,I),SU
PP(0,I),SUPP(1,I)
6320 NEXT I
6330 INPUT£-1,N(0),N(1)
6340 FOR I=0 TO N(0)-1
6350 INPUT£-1,PAYMENT$(0,I)
6360 FOR J=0 TO 11
6370 INPUT£-1,PAYMENT(0,I,J)
6380 NEXT J
6390 NEXT I
6400 FOR I=0 TO N(1)-1
6410 INPUT£-1,PAYMENT$(1,I)
6420 FOR J=0 TO 11
6430 INPUT£-1,PAYMENTS(1,I,J)
6440 NEXT J
6450 NEXT I
6460 CLOSE£-1
6470 FOR H=0 TO 1:GOSUB 2500:NEXT
6480 GOTO 1000
```

The length of this data-file module should be sufficient to convince you of the complexity of the program. Budget, more than any other program in this book, benefits from the early saving of some data to tide you over the innumerable pitfalls of entering a program as long as this.

MODULE 2.3.2

```
7000 REM***************************
7010 REM FORMAT TITLES;
7020 REM***************************
7030 LET P2=14-INT(LEN(F$)/2)
7040 PRINT @ 32*P1+P2,STRING$(LEN(F$)+2,
150)
7050 PRINT @ 32*(P1+1)+P2,CHR$(150)+F$+C
HR$(150)
7060 PRINT @ 32*(P1+2)+P2,STRING$(LEN(F$
)+2,150)
7070 LET P2=0
7080 RETURN
```

A standard title-formatting module.

MODULE 2.3.3

```
3500 REM*************************
3510 REM QUESTIONS
3520 REM*************************
3530 PRINT @ P1*32,STRING$(32," ")
3540 PRINT @ P1*32+P2,P$;
3550 INPUT Q$
3560 PRINT @ 416,">>";Q$;"<<"
3570 PRINT @ 448,"";:INPUT "PRESS enter
TO CONFIRM";R$
3580 PRINT @ 416,STRING$(63," ")
3590 IF R$<>"" THEN GOTO 3530
3600 LET P2=0:RETURN
```

You may be struck at first sight by the similarity between this module and the last and indeed the functions of the two are quite similar. This one, instead of printing a decorative heading at some desired place on the screen, prints a prompt, allows for its confirmation or otherwise and returns the resultant input to the main program. Handling almost all of the program's requests for input by this one module saves scores of lines in the main program.

Commentary

Lines 3530–3550: From these three lines it will be clear that three variables are necessary for the proper functioning of this module: PI which is the line on which the prompt is to be printed, P2 which is the position along the line and P$, which is the actual prompt. The module automatically clears the line on which the prompt is to be printed. In this program P2 is always left at zero but there is no reason why this should be so if the module is used in other programs.

Lines 3560–3590: Whatever is input by the user is redisplayed at the bottom of the screen with a request for confirmation. Confirmation is given by pressing ENTER with no character input. Inputting an actual character is interpreted as meaning that the response to the prompt is not confirmed, in which case the prompt is printed again.

Testing Module 2.3.3

This module may be tested on its own by defining P1 and P$ in direct mode then entering GOTO 3500. P2 does not need to be defined.

MODULE 2.3.4

```
1000 REM*************************
1010 REM MENU
1020 REM*************************
1030 CLS:LET F$="HOME BUDGET":LET P1=0:G
OSUB 7000
1040 PRINT "FUNCTIONS AVAILABLE:"
1050 PRINT " 1)INITIALISE"
1060 PRINT " 2)RESET HYPOTHETICAL FIGURE
S"
1070 PRINT " 3)DISPLAY MONTHLY ANALYSIS"
1080 PRINT " 4)CHANGES"
1090 PRINT " 5)NEW BUDGET HEADINGS"
1100 PRINT " 6)DELETE BUDGET HEADING"
```

```
1110 PRINT " 7)RESET MONTH"
1120 PRINT " 8)DATA FILES"
1130 PRINT " 9)STOP"
1140 PRINT:INPUT "WHICH DO YOU REQUIRE  "
;Z:CLS
1150 IF Z<>1 AND Z<>2 AND Z<>7 AND Z<>8
AND Z<>9 THEN GOTO 1180
1160 ON Z GOSUB 1510,3010,1000,1000,1000
,1000,4000,6000,1230
1170 GOTO 1000
1180 PRINT:PRINT "1)REAL"
1190 INPUT "2)HYPOTHETICAL DATA";H:CLS
1200 LET H=H-1:SCREEN 0,H
1210 ON Z-2 GOSUB 2010,5010,3260,5500:GO
TO 1000
1220 GOTO 1030
1230 CLS:LET F$="HOME BUDGET":LET P1=7:G
OSUB 7000
1240 END
```

A standard menu module with the addition of a provision to set a variable called H according to whether real or hypothetical data is to be referred to. The data stored by this program is divided into two categories which are entirely separate from one another, although data in the real side of the arrays can be copied into the other side. What this means in practice is that if the user wishes to enter some speculative data, e.g. 'what would happen if income were to rise in July by £500 p.a. and two new standing orders were to be entered into from September onwards, plus the purchase of a new TV in March', this can be entered into the hypothetical side of the array so that the interaction of these decisions with existing commitments can be examined, without corrupting existing data about confirmed plans for the year ahead.

Commentary

Line 1190: The variable H is the sole indication that will be used by the program as to whether real or hypothetical data is being worked with, and is used to indicate either the 0 or 1 elements of the arrays.

Line 1200: Note the use of the value of H to reset the screen colour set as a reminder of which type of data is being input or displayed.

MODULE 2.3.5

```
1500 REM*************************
1510 REM SET UP REGULAR PAYMENTS
1520 REM*************************
1530 CLEAR5000:PCLEAR 1:LET FLAG=0
1540 DIM PAYMENT$(1,19),MONTHLY(1,19),PA
YMENTS(1,19,11),PTOTALS(1,11),BDEFICIT(1
,11),INCOME (1,11),SUPP(1,11),BALANCE(1,
11)
1550 DATA "JANUARY","FEBRUARY","MARCH","
APRIL","MAY","JUNE","JULY","AUGUST","SEP
TEMBER","OCTOBER","NOVEMBER","DECEMBER"
1560 DIM MONTH$(11):RESTORE
1570 FOR I=0 TO 11:READ MONTH$(I):NEXT
1580 IF FLAG=1 THEN GOTO 6280
1590 INPUT "NUMBER OF CURRENT MONTH:";MO
:LET MO=MO-1
1600 LET Y=MO+11
1610 GOSUB 4500:GOSUB 3260:GOTO 1030
```

This module initialises the various arrays used by the program. It also loads the array MONTH$ with the names of the months of the year.

Commentary

Line 1580: Once again the variable FLAG is used to determine whether this module returns program execution to the main menu or to the data-file module.

Line 1600: The variable Y is used to store the end of the current 12 month period.

Testing Module 2.3.5

Provided that temporary RETURNs are placed at 4500 and 3260 you should now be able to call up this module from the main menu. Having initialised the program you should also be able to call up the data file module to save the empty arrays, stop the program, re-RUN it and reload the empty arrays. These tests will make use of four of the modules you have entered so far.

MODULE 2.3.6

```
  4500 REM****************************
  4510 REM INCOME
  4520 REM****************************
  4530 CLS
  4540 PRINT "INPUT SALARY AS FOLLOWS:"
  4550 FOR I=MO TO Y
  4560 LET I1=I:IF I1>11 THEN LET I1=I1-12
  4570 LET P$=MONTH$(I1)+":"
  4580 LET P1=I+1-MO
  4590 GOSUB 3500
  4600 LET INCOME(H,I1)=VAL (Q$)
  4610 PRINT @ 32*(I+1-MO)+LEN (P$)+1,INCO
ME(H,I1)
  4620 NEXT I
  4630 CLS
  4640 PRINT "OTHER ANTICIPATED INCOME:"
  4650 FOR I=MO TO Y
  4660 LET I1=I+12*(I>11)
  4670 LET P1=I+1-MO
  4680 LET P$=MONTH$(I1)+":"
  4690 GOSUB 3500
  4700 LET SUPP(H,I1)=VAL(Q$)
  4710 PRINT @ 32*(I+1-MO)+LEN(P$)+1,SUPP(
H,I1)
  4720 NEXT I
  4730 GOSUB 2500
  4740 CLS
  4750 RETURN
```

This module accepts inputs for monthly income figures under two headings, main income and supplementary income.

Commentary

Line 4560: The purpose of this line is to take the value of the loop variable I, which can vary from 0 to 22 (since MO can be from 0 to 11) and to convert that value into something in the range 0 to 11 i.e. something that will point to an element in one of our arrays. Note that this means that our arrays will not run from 0 to 11 representing the forthcoming 12 months, they will run from the number of the current month up to 11 and back via zero to the number before the current month.

Lines 4570–4590: Note how flexible our use of Module 3 can be. Here the prompt used is a month name and the line on which the prompt is printed is determined by the variable I1.

Line 4600: The figure for main income is placed into the array INCOME. Note that this array, like all the others, has two sides, numbered 0 and 1. The side into which the data is placed will depend on the value of H. In the present case, this module is always called by the initialisation module and only real data is input (i.e. the value of H is 0). In subsequent modules H could be either 0 or 1 depending on whether the user has specified real or hypothetical data.

Lines 4640–4720: The same process as above, but supplementary income is input and stored in the array SUPP.

Testing Module 2.3.6

Provided that a temporary return is placed at 2500, you should be able to enter details of income when the initialisation module is called from the main menu. You may wish to confirm in direct mode that the figures are in fact stored from INCOME(0,0) to INCOME(0,11) and in the same positions in SUPP.

MODULE 2.3.7

```
3250 REM****************************
3260 REM INPUT OF PAYMENTS
3270 REM****************************
3280 LET F$="INPUT OF BILLS":LET P1=0:GO
SUB 7000
3310 PRINT "PRECEDE NAME OF ITEM WITH A'
*'  IF YOU DO NOT WANT IT BUDGETED.
3320 LET P$="HEADING <'ZZZ' TO QUIT>"
3330 LET P1=7
3340 GOSUB 3530
3350 IF Q$="ZZZ" THEN GOSUB 2500:RETURN
3360 CLS
3370 PRINT "INPUT UNDER:";Q$
3380 LET N(H)=N(H)+1
3390 IF N(H)=20 THEN LET N(H)=N(H)-1:PRI
NT @ 8*32,"NO MORE ROOM IN PAYMENTS FILE
." FOR I=1 TO 5000:NEXT:RETURN
3400 LET PAYMENT$(H,N(H)-1)=Q$
3410 FOR I=MO TO Y
3420 LET I1=I:IF I1>11 THEN I1=I1-12
3430 LET P$="AMOUNT FOR "+MONTH$(I1)+": "
3440 LET P1=I+1-MO
3450 GOSUB 3530
3460 LET PAYMENTS(H,N(H)-1,I1)=VAL (Q$)
3470 PRINT @ 32*(I+1-MO)+14+LEN(MONTH$(I
1)),Q$
3480 NEXT I
3490 CLS:GOSUB 2500:GOTO 3280
```

This module accepts the input of payment headings and the payments associated with them for the 12 months to come.

Commentary

Line 3310: This prompt refers to a later stage in the program where average monthly payments will be calculated for each payment heading and included in an average monthly budget. Attaching an asterisk to the front of a payment heading means that it is excluded from this process and treated as a one-off expenditure.

Line 3390: Only 20 payment headings are allowed for, though this is a completely arbitrary figure and could be increased (within the limits of the memory) if you wish.

Line 3400: The name of the payment heading is stored in one or other half of the array PAYMENT$ — which half is determined by the value of H.

Lines 3410–3480: For each payment heading, 12 monthly payments are requested and placed onto one or other side of the array PAYMENTS. You may note that this is a three-dimensional array: the first dimension determines whether we are dealing with real or hypothetical data, the second dimension refers to the payment number and the third dimension refers to the month.

Testing Module 2.3.7

Provided that there is still a temporary line 2500 RETURN, you should now be able to input payment headings and their associated payments, verifying in direct mode that they have been placed into the correct positions in PAYMENT$ and PAYMENTS.

MODULE 2.3.8

```
2500 REM***************************
2510 REM UPDATE BUDGET
2520 REM***************************
2530 LET T(H)=0
2540 FOR I=0 TO N(H)-1
2550 LET BUDGET=0
2560 IF LEFT$(PAYMENT$(H,I),1)="*" THEN
GOTO 2600
2570 FOR J=0 TO 11:LET BUDGET=BUDGET+PAY
MENTS(H,I,J):NEXT
2580 LET MONTHLY(H,I)=BUDGET/12
2590 LET T(H)=T(H)+MONTHLY(H,I)
2600 NEXT I
2610 LET TTOTAL=0:LET CUM=0
2620 FOR I=MO TO Y
2630 LET I1=I+12*(I>11)
2640 LET PTOTALS(H,I1)=0
2650 FOR J=0 TO N(H)-1:LET PTOTALS(H,I1)
=PTOTALS(H,I1)+PAYMENTS(H,J,I1):NEXT
2660 LET TTOTAL=TTOTAL+PTOTALS(H,I1)
2670 FOR J=0 TO N(H)-1:IF LEFT$(PAYMENT$
(H,J),1)="*" THEN TTOTAL=TTOTAL-PAYMENTS
(H,J,I1)
2680 NEXT J
2690 LET BDEFICIT(H,I1)=T(H)*(I-MO+1)-TT
OTAL
2700 LET CUM=CUM+INCOME(H,I1)+SUPP(H,I1)
-PTOTALS(H,I1)
2710 LET BALANCE(H,I1)=CUM
2720 NEXT I:RETURN
```

This fairly short module is a difficult one to follow until you have had some experience of the program in practice. The purpose of the module is to perform the calculations which the program is designed to provide, on the basis of the income and expenditure figures supplied by the user. The functions and the arrays will be described in full but you may wish to return to them later when you have seen the figures displayed after the entry of the next module.

Commentary

Lines 2530–2600: This loop calculates average monthly payments for every payment heading except for those preceded by an asterisk. The figure is such that over the year it will be sufficient to cover all payments under that heading. For regular monthly payments, this figure will be the same as the payment itself. This budget figure is then stored in the array MONTHLY, in a position corresponding to that of the payment in the array PAYMENT$. In addition, the budget figure for each payment heading is added to what is already contained in one or other of the halves of the two element array T thus making up a total of the individual budget figures. If there are no payment headings preceded by an asterisk, the eventual figure in T will be 1/12 of the total of all payments over the year.

Lines 2620–2720: Having calculated the budget figures, the module now proceeds to perform a number of calculations for each month, as follows:

Line 2650: The total of the payments falling during the month is accumulated in the relevant element of the array PTOTALS.

Line 2660: This total monthly payment is itself accumulated over the 12 months in the variable TTOTAL.

Line 2670: From TTOTAL is subtracted the amount for any payment which has been marked by the user with an asterisk. TTOTAL now contains the accumulated total of all payments since the beginning of the 12 month period which were included in the budgeting calculation.

Line 2690: An element in the array BDEFICIT is now set equal to the difference between the actual amount set aside in the budget and the total of payments which the budget is meant to be covering. The element in BDEFICIT corresponding to any particular month will indicate the extent to which the average monthly budget is ahead or behind the items included in the budget. If all the budgeted items are due for payment in the first month of the period, then the budget will be in deficit until the last month of the year. If all the budgeted items are not due for payment until the last month of the period then the budget will be in surplus for every month up to the last.

Lines 2700–2710: The relevant element of the array BALANCE is set equal to the cumulative difference between total income and total expenditure.

Testing Module 2.3.8

If you have not already done so, it would be best to save some data before testing this module. Re-inserting your temporary RETURN at line 2500 will allow you to input data for income and payment headings (one or two payments are quite sufficient at present). Having done that, the only test that is really practical at the present time is to allow this module to be called and to discover any syntax errors that may have crept into what you have entered. For overall checking of the module's functions it is probably better to wait until the next module is entered and the figures can be displayed.

MODULE 2.3.9

```
2000 REM***************************
2010 REM DISPLAY FIGURES
2020 REM***************************
2030 LET F$="PAYMENTS":LET P1=1:GOSUB 70
00
2040 LET P$="MONTH TO START":LET P1=5:GO
SUB 3530
2050 FOR I=0 TO 11:IF Q$<>MONTH$(I) THEN
 NEXT I:GOTO 2040
2060 CLS: LET M1=I:IF MO-M1-12*(M1>MO-1)
<4 THEN LET M1=MO-4-12*(MO<5)
2070 PRINT "MONTH      ";:FOR J=M1TOM1+3:P
RINT CHR$(159);LEFT$(MONTH$(J+12*(J>11))
,3);:NEXT J:PRINT CHR$(159);CHR$(159);CH
R$(159);" B ";CHR$(159);
2080 PRINT STRING$(32,CHR$(159));
2090 FOR I=0 TO N(H)-1
2100 IF I<>11 THEN GOTO 2130
2110 PRINT @ 13*32,"":INPUT "'ENTER' TO
CLS AND CONTINUE";Q$
2120 FOR J=2 TO 14:PRINT @ J*32,"":NEXT:
PRINT @ 2*32,"";
2130 PRINT USING "%          %";PAYMENT$(H,
I);
2140 FOR J=M1 TO M1+3
2150 PRINT CHR$(159);
2160 IF INT(PAYMENTS(H,I,J+12*(J>11)))>=
1000 THEN PRINT "***";ELSE PRINT USING "
£££";INT(PAYMENTS(H,I,J+12*(J>11)));
2170 NEXT J
2180 PRINT STRING$(3,CHR$(159));
2190 PRINT USING "£££";INT (MONTHLY(H,I)
);
2200 PRINT CHR$(159);
2210 NEXT I
2220 PRINT STRING$(32,CHR$(159));
2230 INPUT "PRESS ENTER FOR ANALYSIS";Q$
2240 FOR I=2 TO 14:PRINT @ I*32,"":NEXT
2250 PRINT @ 64,"TOTAL"
2260 PRINT "BUDGET",,"BUDG.BAL.",,"PAY",
,"OTHER INC",,"TOTAL INC",,"CASH BAL.",
"CUM. BAL."
2270 FOR I=M1 TO M1+3
2280 LET I1=I+12*(I>11)
2290 LET I2=9+4*(I-M1)
2300 IF PTOTALS(H,I1)<0 THEN PRINT @ 64+
I2,CHR$(191);ELSE PRINT @ 64+I2,CHR$(159
);
2310 PRINT USING "£££";INT(ABS(PTOTALS(H
,I1)))
2320 IF T(H)<0 THEN PRINT @ 96+I2,CHR$(1
91);ELSE PRINT @ 96+I2,CHR$(159);
2330 PRINT USING "£££";INT(ABS(T(H)))
2340 IF BDEFICIT(H,I1)<0 THEN PRINT @ 12
```

```
8+I2,CHR$(191);ELSE PRINT @ 128+I2,CHR$(
159);
2350 PRINT USING "£££";INT(ABS(BDEFICIT(
H,I1)))
2360 PRINT @ 160+I2,CHR$(159);
2370 PRINT USING "£££";INT(INCOME(H,I1))
2380 PRINT @192+I2,CHR$(159);
2390 PRINT USING "£££";INT(SUPP(H,I1))
2400 PRINT @ 224+I2,CHR$(159);
2410 PRINT USING "£££";INT(INCOME(H,I1)+
SUPP(H,I1))
2420 IF INCOME(H,I1)+SUPP(H,I1)-PTOTALS(
H,I1)<0 THEN PRINT @ 256+I2,CHR$(191);EL
SE PRINT @ 256+I2,CHR$(159);
2430 PRINT USING "£££";INT(ABS(INCOME(H,
I1)+SUPP(H,I1)-PTOTALS(H,I1)))
2440 IF BALANCE(H,I1)<0 THEN PRINT @ 288
+I2,CHR$(191);ELSE PRINT @ 288+I2,CHR$(1
59);
2450 PRINT USING "£££";INT(ABS(BALANCE(H
,I1)))
2460 NEXT I
2470 PRINT STRING$(32,CHR$(159))
2480 INPUT "DO YOU WISH TO SEE FIGURES
     AGAIN (Y/N)";Q$
2490 IF Q$<>"Y" THEN RETURN ELSE CLS:GOT
O 2070
```

We noted in the last program that display modules for complex data are themselves likely to be complex, and this, the largest module in the program, is no exception. Its purpose is to display, in two separate tables, the details of payments entered so far, the income figures and the analysis which was performed in the course of running the last module.

Commentary

Line 2060: The desired starting month having been checked, this line ensures that the table never starts less than three months before the final month of the 12 month period, since this would render the resulting table meaningless.

Line 2070: The first three letters of the four months to be covered are printed across the top of the screen.

Lines 2090–2210: In this loop a series of lines are printed, each containing a payment heading taken from PAYMENT$, the associated payments for the four months in question taken from PAYMENTS and the monthly budget figure for that heading, taken from MONTHLY. Note that the table is formatted on the basis that the highest figure for a monthly payment will be £999. If this is exceeded, *** is printed to remind the user that this figure cannot be accurately represented in the table. Note also the use of a loop to clear a part of the screen in the event that the number of payments exceeds the capacity of a single screen.

Lines 2270–2460: For each of the four months in question, the following figures are printed in the relevant column:
a) the total payments in each month (2300–2310)
b) the total of budget payments for the month (2320–2330)

c) the difference between the budget figure and the actual payments it is meant to cover (2340–2350)
d) main income (2370)
e) supplementary income (2390)
f) the balance of income over expenditure for the month (2420–2430)
g) the cumulative balance of income over expenditure since the beginning of the 12 month period (2440–2450).

You will note that the position of each item on the line is dictated by the variable I2, based upon the value of the loop variable I. This means that in the case of figures exceeding £999, instead of not being able to print them without disrupting the orderliness of the table, they are printed with a preceding % to show that they are truncated, followed by the first two digits. This flag is automatically provided by the Dragon as a result of our employing the PRINT USING "###" format. Note also that negative balances are indicated by the setting of the square constituting the wall of the column in front of the item to red (CHR$(191)) rather than yellow (CHR$(159)).

Testing Module 2.3.9

Provided that you have recorded a valid set of data, you are now in a position to test this module by loading the data and calling up this module. You should be faced with an orderly table of figures as described in the commentary above. If not, at least you have the recorded data to reduce the tedium of subsequent tests.

MODULE 2.3.10

```
3000 REM****************************
3010 REM SET UP SHADOW ARRAYS
3020 REM****************************
3030 LET T(1)=T(0)
3040 FOR I=0 TO N(0)-1
3050 LET PAYMENT$(1,I)=PAYMENT$(0,I)
3060 LET MONTHLY(1,I)=MONTHLY(0,I)
3070 FOR J=0 TO 11
3080 LET PAYMENTS(1,I,J)=PAYMENTS(0,I,J)
3090 LET PTOTALS(1,J)=PTOTALS(0,J)
3100 LET BDEFICIT(1,J)=BDEFICIT(0,J)
3110 LET INCOME(1,J)=INCOME(0,J)
3120 LET SUPP(1,J)=SUPP(0,J)
3130 LET BALANCE(1,J)=BALANCE(0,J)
3140 NEXT J,I
3150 LET N(1)=N(0)
3160 LET H=1
3170 RETURN
```

Having entered what is, to all intents and purposes a working program, we now go on to add some features which add to the flexibility of our tool. This module, for instance, allows the user to reset the hypothetical side of the table to the data in the real side. This is simply done by copying from one side to the other. Note that calling this module results in the loss of any hypothetical data which is not also present on the real side of the arrays.

Testing Module 2.3.10

Having entered some hypothetical data using menu function 5, you are now in a position to reset the hypothetical side of the arrays to the parallel real set of data, regardless of whether there are more or less items in the real side.

MODULE 2.3.11

```
5500 REM**************************
5510 REM DELETE BUDGET HEAD
5520 REM**************************
5530 LET P$="NAME OF ITEM TO BE DELETED"
:LET P1=1:GOSUB 3500
5540 FOR I=0 TO N(H)-1:IF Q$<>PAYMENT$(H
,I) THEN NEXT I:LET F$="ITEM NOT FOUND":
LET P1=10:GOSUB 7000:FOR I=1 TO 5000:NEX
T:RETURN
5550 LET N(H)=N(H)-1
5560 FOR J=I TO N(H)-1
5570 LET PAYMENT$(H,J)=PAYMENT$(H,J+1)
5580 FOR K=0 TO 11
5590 LET PAYMENTS(H,J,K)=PAYMENTS(H,J+1,
K)
5600 NEXT K
5610 NEXT J
5620 GOSUB 2510
5630 RETURN
```

This straightforward module allows the user to specify a payment heading which, if it is found to be present in the file of payments, is deleted.

Testing Module 2.3.11

You should now be able to delete an item from either side of the payments file.

MODULE 2.3.12

```
5000 REM**************************
5010 REM CHANGES
5020 REM**************************
5030 LET F$="CHANGES":GOSUB 7000
5040 PRINT "COMMANDS AVAILABLE:"
5050 PRINT " 1)CHANGE EXISTING BUDGET HE
AD"," 2)CHANGE MAIN INCOME"," 3)CHANGE A
DDITIONAL INCOME"
5060 LET P$="WHICH DO YOU REQUIRE":LET P
1=8:GOSUB 3500
5070 LET H$=Q$
5080 CLS
5090 IF H$="1" THEN GOSUB 5130
5100 IF H$="2" OR H$="3" THEN GOSUB 5320
5110 GOSUB 2500
5120 RETURN
5130 LET F$="NAME OF BUDGETARY HEADING T
O BE CHANGED":LET P1=1:GOSUB 3500
5140 FOR I=0 TO N(H)-1:IF Q$<>PAYMENT$(H
,I) THEN NEXT I:LET F$="NO HEADING OF TH
AT NAME":LET P1=12:GOSUB 7000:FOR I=1 TO
 5000:NEXT:RETURN
5150 LET B=I
5160 CLS
5170 LET P$="NEW FIGURE OR 'Z' TO LEAVE"
:LET P1=12
5180 FOR I=MO TO Y
5190 LET I1=I+12*(I>11)
5200 PRINT @ 32*(I-MO),MONTH$(I1);
5210 PRINT TAB( 10) " ";
5220 PRINT USING "££££.££";PAYMENTS(H,B,
I1)
5230 GOSUB 3500
5240 IF Q$="Z" THEN GOTO 5290
5250 LET PAYMENTS(H,B,I1)=VAL (Q$)
5260 PRINT @ 32*(I-MO)+15,"<";
```

```
5270 PRINT USING "££££.££";PAYMENTS(H,B,
I1);
5280 PRINT ")"
5290 NEXT
5300 GOSUB 2500
5310 RETURN
5320 IF H$="2" THEN PRINT "MAIN INCOME"
ELSE PRINT "SUPPLEMENTARY INCOME"
5330 LET P$="NEW FIGURE OR 'Z' TO LEAVE"
5340 LET P1=13
5350 FOR I=MO TO Y
5360 LET I1=I+12*(I>11)
5370 PRINT @ 32*(I-MO+1),MONTH$(I1);
5380 PRINT TAB(10) "(";
5390 IF H$="2" THEN PRINT INCOME(H,I1) E
LSE PRINT SUPP(H,I1)
5400 GOSUB 3500
5410 IF Q$="Z" THEN GOTO 5460
5420 IF H$="2" THEN LET INCOME(H,I1)=VAL
(Q$) ELSE LET SUPP(H,I1)=VAL(Q$)
5430 PRINT @ 32*(I-MO+1)+15,"(";
5440 PRINT USING "££££.££";VAL(Q$);
5450 PRINT ")"
5460 NEXT I
5470 RETURN
```

The purpose of this module is to allow the user to specify changes to any figure input for payments, main income or supplementary income.

Commentary

Lines 5180–5290: In this section the payments stored under a particular payment heading are displayed one by one. To leave a value unchanged, the user must input **Z**; to change a value it is only necessary to input the new value. Note the usefulness of our prompt module here, since the prompt itself can be defined outside the loop and used for each repetition of the loop without further definition.

Lines 5320–5460: Throughout this section of the program the array to be addressed (either INCOME or SUPP) is determined by H$ — this saves considerable space over having a separate section for each income type, though the section could be even shorter if the two types of income were contained in separate halves of the same array.

Testing Module 2.3.12

You should now be able to change any of the data which you have previously input.

MODULE 2.3.13

```
4000 REM*****************************
4010 REM REGISTER MONTH
4020 REM*****************************
4030 LET F$="HOME BUDGET":LET P1=0:GOSUB
 7000
4040 LET P$="WHAT MONTH IS IT":LET P1=5:
GOSUB 3500
4050 IF Q$=MONTH$(MO) THEN RETURN
4060 FOR I=0 TO 11:IF Q$<>MONTH$(I) THEN
 NEXT I:CLS:PRINT @ 9*32,"THERE MUST BE
SOME MISTAKE.       I DON'T KNOW OF A MONT
H CALLED  ";Q$;"!":GOTO 4030
4070 LET M2=I
4080 IF M2<MO THEN LET M2=M2+12
```

```
4090 FOR I=MO TO M2-1
4100 LET I1=I+12*(I>11)
4110 CLS
4120 LET F$="UPDATE":LET P1=0:GOSUB 7000
4130 PRINT "PLEASE INPUT AMOUNTS FOR NEX
T",MONTH$(I1)
4140 FOR J=0 TO N(0)-1
4150 LET P1=5
4160 LET P$=PAYMENT$(0,J)+" ("+STR$ (PAY
MENTS(0,J,I1))+"):"
4170 GOSUB 3530
4180 LET PAYMENTS(0,J,I1)=VAL (Q$)
4190 NEXT J
4200 LET P$="MAIN INCOME (""+STR$(INCOME(
0,I1))+"):":LET P1=7:GOSUB 3500
4210 LET INCOME(0,I1)=VAL (Q$)
4220 LET P$="ADDITIONAL INCOME (""+STR$ (
SUPP(0,I1))+"):":LET P1=9:GOSUB 3500
4230 LET SUPP(0,I1)=VAL (Q$)
4240 NEXT I
4250 LET MO=M2+12*(M2>11):LET Y=MO+11:LE
T H=0
4260 GOSUB 2500:GOSUB 3010:RETURN
```

Our final module is one which allows the user to avoid the necessity to make piecemeal changes every time the month changes. The function of the module is to delete the data for any months which the user defines as past and to request inputs under each payment head and for the two income types for all the months necessary to make up a 12 month period from the new current month.

Commentary

Line 4090: This loop represents the months from the previous current month (MO) to the new month which has just been input, but one year ahead i.e. if the program has not been used for two months and it is updated from January to March, it will request inputs for next January and February.

Lines 4140–4190: In this loop, all the payment headings are presented for updating. Note that the variable H is not used to define which part of the arrays since in resetting the data it is only the real side of the arrays which is addressed. At the end of this module a call is made to the module which resets the hypothetical arrays — this may be omitted provided that it is remembered that the hypothetical arrays will now contain out of date information.

Lines 4200–4230: Main income and supplementary income are updated.

Line 4250: MO is set equal to the current month, reduced so that it falls into the range 0 – 11 if necessary.

Testing Module 2.3.13

You should now be able to update the program one or more months and be prompted to give the necessary information to accomplish this. If this module functions correctly the program is ready for use.

Summary

This long program is a powerful tool, properly used, although it takes some practice to get the most out of it. Taken seriously it can give you some surprising information about the state of your finances throughout the year — when things will be tight and when there might be a bit to spread around, how payments could be re-arranged to ensure a little more at Christmas or for holidays, what might be the effect overall of a new commitment or of increased income.

Remember, however, that this book is intended to set your Dragon to work for you. If you have successfully overcome the problems of debugging this program then there is no reason why you should not go on to adapt it to other uses which require the flexible input and manipulation of data, together with clear presentation and the possibility of running two parallel sets of information if desired. Simple changes to Module 8 could lead to a very different type of program using almost the same arrays but calculating something entirely different. The Dragon is yours, and so is the confidence you have gained in entering this program. The program itself is only a foundation for putting your Dragon and your confidence to work.

Going Further

1) The program would be more flexible if the user had the option of copying the hypothetical data into the real side of the arrays. If you think about it that should only involve a very small change in the program.
2) As hinted in the commentary on Module 12, real savings in the length of this program could be made by declaring one more complex array and inputting data to the various parts of the array according to a small number of new variables. One place to start might be in combining main and supplementary income into one array.
3) At the moment, the user has to work through the whole 12 monthly payments, even if only one is to be changed. How difficult would it be to add an escape from this process, or even make it possible to access a single month's payment on command?

CHAPTER 3
Drawing on the Dragon

After the rigours of the last program it is with a sigh of relief that we turn to the topic of the Dragon's excellent graphics capabilities. I should hasten to add that this chapter is in no way intended to be exhaustive, for the Dragon's capabilities in this field could (and no doubt will) be the subject of a book in their own right. Nevertheless, in this chapter we shall tackle such areas as the creation and saving of simple pictures or maps for use by later programs, the drawing of geometric shapes, the saving of screen memory on tape and the design of complex patterns up to 10000*10000 pixels.

The programs you will find in this chapter include Artist, a text screen graphics tool; Tangrams, a program which allows you to play the ancient Chinese shape game; Doodle, which allows the owners of Dragon joysticks to turn their screen into a sketch pad and Designer, a sophisticated tool for the drawing of large-scale plans.

3.1 ARTIST

While it is true that the high-resolution graphics capabilities of the Dragon are some of the finest to be found in any micro computer on the market, it should not be forgotten that there is also a useful set of graphics characters available in the text mode. Indeed, given the difficulty of placing text and high-resolution graphics on the screen at the same time, there are many tasks which, unless they can be accomplished with the low resolution graphics characters, are not going to be accomplished at all.

The purpose of the present program, apart from giving you the ability, for the sheer fun of it, to draw multi-coloured pictures on your screen, is to act as a feeder for two later programs in this book which require designs as part of their data and indeed, to provide easily recallable designs for programs of your own which might benefit from the drawing of one or two simple pictures.

MODULE 3.1.1

```
1000 REM************************
1010 REM INITIALISE
1020 REM************************
1030 DIM CORNER(3):CLS0:POKE(1024+14*32)
,22
1040 LET X=16:LET Y=8
```

This module initialises the screen display for the subsequent modules.

Commentary

Line 1030: In this program we shall be moving a cursor around the screen and using a second cursor, on the 15th line of the display, to indicate the graphics character currently in use. In both cases this is more conveniently done by POKEing the character into the screen memory than by printing onto the screen. The POKE command has the effect of placing a number specified in the command into a specified memory location. In our case the memory location chosen will be that part of the Dragon's memory which is used to store the contents of the screen. Any number POKEd there will be interpreted as the character having that ASCII code (see Appendix A of the Dragon manual). As a result the chosen character will appear on the screen. All such POKEing will begin at a base of 1024, which is the first location of the memory for the screen in text mode and to this base will be added a number between 0 and 511, representing the number of locations on the text screen. In this particular case, having cleared the screen and set it black, the POKE command places a white V on the 15th line for later use as a cursor.

Line 1040: X and Y represent the co-ordinates of the main cursor on the 32*16 text screen.

Testing Module 3.1.1

The screen should be set to black and a green V should appear on the 15th line of the display.

MODULE 3.1.2

```
1500 REM****************************
1510 REM CURSOR MOVE
1520 REM****************************
1540 FOR I=0 TO 15:POKE(1024+32*15+2*I),
(128+I):POKE(1024+32*15+2*I+1),175:NEXT
1550 LET T$=INKEY$:IF T$<>"" THEN GOTO 2
030
1560 LET P=PEEK (1024+Y*32+X)
1570 POKE(1024+Y*32+X),106:FOR I=1 TO 25
:NEXT
1580 POKE(1024+Y*32+X),P:FOR I=1 TO 25:
NEXT
1590 GOTO 1550
```

The main purpose of this module is to provide a flashing cursor at the co-ordinates contained in X and Y, until such time as the user inputs a command.

Commentary

Line 1540: Re-examination of Appendix A will serve to remind you that the Dragon has a set of 128 low resolution graphics characters, representing 16 basic characters times the eight possible colours available in this mode. The purpose of this line is to display across the bottom of the screen one complete set of the characters in the colour green. These will later be used to select the character to be POKEd onto the screen.

Lines 1550–1590: This short module bears some close study if you have not come across anything similar before since, in some form or other it is found in many of the subsequent programs of this book and will find its way into many of the programs you will go on to write for yourself. Its purpose is first of all to provide a waiting state, during which time a flashing cursor will be displayed on the screen.

Line 1550: This line uses the extremely useful INKEY$ function to detect whether the user has made any input to the program. Unlike INPUT, which requires the user to press ENTER before an input is recognised, the INKEY$ function constantly scans the whole of the keyboard to see whether a key is being depressed and, if it finds one that is, it is labelled as a string called INKEY$. If no key is being depressed then INKEY$ is simply an empty string or "" in Basic (note that there is no space between these two quotation marks). It is usual to set another string — I always use T$ — equal to INKEY$ before going on to use it, for the simple reason that by the time subsequent program lines have been reached it may well be that INKEY$ will have changed due to the finger being lifted from the key.

Line 1560: Parallel to the POKE command is PEEK, which simply looks at a particular memory location and returns the number which is to be found there. It is used here since when we move our cursor around in the later stages of the program we do not wish it to obliterate any parts of the design we have built up which it passes across. Accordingly, the original contents of the screen location where the cursor is to flash are first placed into the variable P.

Lines 1570–1580: These two lines provide a single flash, on and off, of the cursor. In the first line the code value of an asterisk is POKEd into the screen location specified by X and Y. After a short pause, the original character, whose code value is stored in the variable P, is replaced there. The two short loops at the end of the lines are there to make the flashing slow enough to be visible as a regular on-off rhythm.

Line 1590: This cycle is repeated for as long as a key is not depressed.

Testing Module 3.1.2

A flashing asterisk should appear towards the centre of the screen. Pressing any key should result in an undefined line error.

MODULE 3.1.3

```
2000 REM****************************
2010 REM EDIT COMMANDS
2020 REM****************************
2030 LET X=X-(T$=CHR$(9))+(T$=CHR$(8)):L
ET X=X+(X>31)-(X<0)
2040 LET Y=Y-(T$=CHR$(10))+(T$=CHR$(94))
:LET Y=Y+(Y>13)-(Y<0)
2060 IF T$>="0" AND T$<="7" THEN POKE (1
024+32*Y+X),128+Y1/2+16*VAL(T$)
2070 IF T$="8" THEN POKE (1024+32*Y+X),1
28
2080 IF T$="£" THEN GOSUB 3000:GOTO 1540
2090 IF T$="," OR T$="." THEN GOSUB 2530
2100 IF T$="S" THEN GOSUB 3060:GOTO 1540
2110 IF T$="M" THEN GOSUB 3500
2120 GOTO 1550
```

The purpose of this module is to decide what action to take on the basis of the key which the user has depressed.

Commentary

Lines 2030–2040: In order to understand these two lines you must first know something of the way in which the Dragon understands the truth or falsity of conditions — expressions like A = B or X > Y. Try entering the following lines:
9999 INPUT X: IF X THEN PRINT 'X'
10000 GOTO 9999

Running these two lines will reveal that the Dragon only considers the IF statement to have been fulfilled if the value input for X is not zero. This is an important lesson — for the Dragon, true means simply not equal to zero and false means equal to zero. How does that apply to a condition such as X = Y?

The answer is that X = Y is interpreted in exactly the same way: if X is equal to Y then the expression is given a value (actually − 1) and if X is not equal to Y the expression is given a value of zero. What this means is that such conditions can actually be used as variables in the course of a program, even though they can only have two possible values, 0 and − 1. That is exactly what happens in these two lines. The value of the four conditions is used to alter the variables X and Y if, and only if, one of the arrowed keys on the Dragon keyboard has been depressed (the character codes referred to in the four conditions are those of the right, left, down and up arrows respectively). If one of those keys has been depressed, then one and only one of these conditions will have a value of −1, the others having a value of zero. After this alteration to the values of one or other of the co-ordinates, the values of the four conditions are again used to check that neither X nor Y have passed out of the normal bounds of the 32*14 screen available to the cursor. If X, for instance, is greater than 31, it will automatically be reduced by 1 by the first condition of the second statement in line 2030. Remember not to be confused by the seemingly contradictory + and − signs — a condition is −1 if it is true, not 1.

Lines 2060 – 2110: The key depressed need not, of course, have been one of the arrowed keys, in which case T$ (the key depressed) may activate one of the lines in this section which either perform a direct act or allocate program execution to another module.

Line 2060: We have already printed a cursor and a line of graphics characters at the bottom of the screen. When that cursor is moved the value of its position on the 15th line of the display will be held in the variable Y1. Y1 will also be the value of the graphics character pointed to (above a base of 128). What this line does is to POKE a graphics character in a colour corresponding to one of the colour codes between 0 and 7 onto the screen at a location specified by the position of the cursor. This is done in response to any input between 0 and 7.

Line 2070: Following from inputs of 0 to 7, input of 8 will result in the erasure of any character present in the cursor position.

Line 2080: Pressing # allows the user to define one corner of a rectangular area on the screen which can later be saved on tape. This will be explained later.

Line 2090: In the event that the key < or > is pressed the second cursor is moved — this will be explained later.

Line 2100: Pressing of S will result in a defined area of the screen being saved to tape.

Line 2110: Pressing M will also result in the screen display being saved, but in a different format.

Line 2120: If the key depressed is none of the above, the program execution simply returns to the flashing cursor.

Testing Module 3.1.3

You should now be able to move the cursor around the screen, though none of the other program functions is yet available.

MODULE 3.1.4

```
2500 REM***************************
2510 REM CHOOSE CHARACTER
2520 REM***************************
2530 POKE 1024+14*32+Y1,128
2540 LET Y1=Y1-2*(T$=".")+2*(T$=","):LET
 Y1=Y1-2*(Y1<0)+2*(Y1>30)
2550 POKE(1024+14*32+Y1),22
2560 RETURN
```

The purpose of this module is to allow the user to select a graphics character for POKEing onto the screen.

Commentary

Lines 2530–2550: If you have understood the previous cursor move module then you will see that this is a simplified version (since this second cursor moves only along the line, not up and down. Notice that here the value of the conditions employed are multiplied by 2 since the cursor moves in 2-space steps. As mentioned before, the value of YI, from 0 to 15, also corresponds to the value of the graphics character it is pointing to (+ 128).

Testing Module 3.1.4

You should now be able to select a graphics character which can be placed onto the screen in any one of eight colours by subsequently pressing a key from 0 to 7. Pressing 8 should erase the character over which the flashing cursor has been placed. Note that the character placed on the screen will not be visible until the cursor has been moved from that position.

MODULE 3.1.5

```
3000 REM***************************
3010 REM SAVE DESIGN
3020 REM***************************
3030 LET T1$=INKEY$:IF T1$="" THEN GOTO
3030
3040 IF T1$>"0" AND T1$<"3" THEN LET COR
NER(VAL(T1$)*2-2)=X:LET CORNER(VAL(T1$)*
2-1)=Y
3050 RETURN
3060 IF CORNER(0)>CORNER(2)-2 OR CORNER(
1)>CORNER(3)-2 THEN PRINT @ 15*32,"RECTA
NGLE IMPROPERLY DEFINED.";:FOR I=1 TO 10
00:NEXT:RETURN
3070 IF (CORNER(2)-CORNER(0)-1)*(CORNER(
3)-CORNER(1)-1)>240 THENPRINT @ 15*32,"D
ESIGN TOO LARGE.":FOR I=1 TO 1000:NEXT I
:RETURN
3080 FOR I=1 TO 2:POKE (1024+32*CORNER(I
*2-1)+CORNER(I*2-2)),175:NEXT I
3090 PRINT @ 15*32,"THESE POINTS OK (Y/N
):?";
3100 LET Q$=INKEY$:IF Q$="" THEN GOTO 31
00
3110 IF Q$<>"Y" THEN FOR I=1 TO 2:POKE(1
024+32*CORNER(I*2-1)+CORNER(I*2-2)),128:
NEXT I:RETURN
3120 LET DESIGN$=STRING$(9," "):MID$(DES
IGN$,1 )=STR$(CORNER(1 )+1 ):MID$(DESIGN$,4
)=STR$(CORNER(0 )+1 ):MID$(DESIGN$,7 )=STR$
(CORNER(2 )-CORNER(0 )-1 )
3130 FOR I=CORNER(1 )+1 TO CORNER(3 )-1
3140 FOR J=CORNER(0 )+1 TO CORNER(2 )-1
3150 LET DESIGN$=DESIGN$+CHR$(PEEK(1024+
32*I+J))
3160 POKE(1024+32*I+J),106
3170 NEXT J
3180 NEXT I
3190 CLS0
3200 LET Y=VAL(LEFT$(DESIGN$,3))
3210 LET X=VAL(MID$(DESIGN$,4,3))
3220 LET Z=VAL(MID$(DESIGN$,7,3))
3230 FOR I=Y TO Y+(LEN(DESIGN$)-9)/Z-1
3240 FOR J=X TO X+Z-1
3250 POKE (1024+32*I+J),ASC(MID$(DESIGN$
,10+(I-Y)*Z+J-X,1))
3260 NEXT J,I
3270 PRINT @ 15*32,"THIS IS WHAT IS BEIN
G SAVED."
3280 FOR I=1 TO 1000:NEXT
3290 MOTOR ON:AUDIO ON:INPUT "POSITION T
APE THEN PRESS enter (MOTOR IS ON):";Q$
3300 MOTOR OFF:INPUT "PLACE RECORDER INT
O RECORD MODE";Q$
3310 MOTOR ON:FOR I=1 TO 10000:NEXT
3320 OPEN "O",£-1,"ARTIST"
3330 PRINT £-1,DESIGN$:FOR I=10 TO LEN(D
ESIGN$):PRINT£-1, ASC(MID$(DESIGN$,I,1))
:NEXT I
3340 CLOSE £-1
3350 STOP
```

This program is not solely intended to permit the user to doodle on the screen in text mode. Its other purpose is to act as a feeder for later programs which need some pictorial output. This module is one of two which saves the design that you have created for later use. In this particular case the design is saved in such a way that only that part of the screen which actually has the design on it needs to be remembered.

Commentary

Lines 3030–3050: Having pressed # while the cursor was flashing, the user can now input 1 or 2 to designate two opposite corners of the rectangular areas of screen to be saved later.

Line 3060: The main part of this module, called by pressing the S key, begins here with a check that the two corners defined by the user do in fact describe a valid rectangle. For the rectangle to be acceptable, all that is necessary is for the corner numbered 1 to be above and to the left of the corner numbered 2.

Line 3070: This particular format for saving a design is intended for small scale designs and a string will eventually be used for storing the design in the program which later picks up the design from tape. Since the maximum length of a string in Dragon Basic is 255 characters, a check is made that the size of the design will not make it impossible to store it in one string.

Lines 3080–3100: The points defined as the corners of the rectangle to be saved are marked on the screen for the user to confirm. Note that the two points so marked are actually immediately outside the rectangle to be saved.

Line 3120: In this line begins the process of building up the string which will be used to recreate the design at a later date. The first nine places of the string are given over to recording the co-ordinates of the top left-hand corner of the specified rectangle, followed by the width. They are placed into the string using the STR$ function which translates a number into a string. Note that we do not use LET in placing these figures into the string. That is because when defining a part of a string using the format MID$(A$,1)="xxx" the inclusion of LET at the beginning of the command actually results in a syntax error. This is a quirk of Dragon Basic which can confuse anyone new to the machine.

Lines 3130–3180: These two loops scan through the screen positions falling within the defined rectangle and store the characters found therein DESIGN$. To show the progress of the loops an asterisk is placed in each location as it is dealt with, but this line is unnecessary if you feel that you do not need such reassurance.

Lines 3200–3260: These lines ape the process which will be carried out by later programs which reprint the design stored in this format. Their effect is first of all to extract the co-ordinates of the top left hand corner of the specified rectangle, together with the width. These starting points are then used, in conjunction with the loop variables to replace the characters in DESIGN$ in the original screen locations from which they were taken. The sole purpose of this is to give an example of the method of recalling such a design and to reassure the user that the design has been properly recorded.

Lines 3280–3340: Having come this far in the cunning construction of a string to store the design, further progress is barred by an irritating limitation in the Dragon's Basic. All that should really be necessary to store the design on tape should be the instruction PRINT #–1, DESIGNS. Unfortunately, the Dragon steadfastly refuses to recognise the existence of the graphics characters, which are not standard to the ASCII (American Standard Codes for Information Interchange) character set, when saving and loading data. Consequently, while we can save the first nine characters of DESIGN$ (i.e. the numbers representing co-ordinates and width) all the graphics characters themselves have to be transformed into their code values and saved as numbers. This limitation is one of the most disappointing on the Dragon, since quite apart from present uses, the ability to store numbers in the range 0 to 255 in single characters and save them in that form is one of the commonest ways of reducing the amount of memory used for data storage on home micros.

Testing Module 3.1.5

This module can only really be tested when we have entered a subsequent program which will pick up the design from tape and reprint it. Provided that the method of defining the desired rectangle works satisfactorily, together with the associated error-checks, and that once the screen has been cleared the design is reprinted in its original form, you can be fairly confident that the whole of the module is working properly. The only way to be absolutely sure would be to enter the relevant parts of the later program Words which would recall the design from tape and reprint it.

MODULE 3.1.6

```
3500 REM***************************
3510 REM SAVE MAP
3520 REM***************************
3530 PRINT @ 14*32,"";:MOTOR ON:AUDIO ON
:INPUT "POSITION TAPE (enter):";Q$
3540 MOTOR OFF:PRINT @ 14*32,"";:INPUT "
START RECORDER (enter):";Q$
3550 MOTOR ON:FOR I=1 TO 10000:NEXT
3560 OPEN "O",£-1,"MAP"
3570 FOR I=1 TO 14*32:PRINT £-1,PEEK(102
3+I):NEXT I
3580 CLOSE £-1
3590 STOP
```

For applications which need the use of a larger design, one which would be too big to store in a single string, this method of storing a design stores the whole of the screen display, again in the form of the code values of the individual characters. The first 14 lines are stored.

Testing Module 3.1.6

Once again, this module can only be effectively tested when a later program, Where, has been entered.

Summary

A great deal of space has been devoted to the commentary on this program for the simple reason that the techniques used here will be found to have applications far beyond the present program — or indeed only graphics applications. In the programs that follow we shall be moving cursors around screens and inputting one-key commands by means of INKEY$ with gay abandon, so do ensure that you have understood what you have entered. Apart from the techniques, however, the program is a good example of the way in which a program with an interesting function (drawing pictures) can be made into a useful tool with a little thought. A major point in building up a library of programs is not that there should be a wide variety of totally self-contained programs, but that the programs should all contribute to each other's usefulness by the ability to exchange data where appropriate.

Going Further

1) The program might be more useful if, instead of simply stopping once a design has been saved, it were to clear the screen and reprint the design, then return to the cursor module so that the design can be further developed if desired.
2) A one key instruction for clearing the screen might be another useful addition.

ARTIST: Summary of one-key functions

With flashing cursor:

0 to 7	prints graphics characters specified by bottom cursor, in colour indicated by key value.
8	erases character over which cursor is positioned.
#	moves to program section which allows definition of rectangle to be saved.
' or ''	moves bottom cursor to indicate a different graphics character.
S	saves specified rectangle.
M	saves first 14 lines of display.

Without flashing cursor: i.e. after input of #.

1	defines top left-hand corner of rectangle to be saved.
2	defines bottom right-hand corner of rectangle to be saved.

3.2 DOODLE

We turn our attention now to a short program which is mainly for fun but which also contains a useful lesson when it comes to the saving of graphics displays. The name of the program is Doodle, and the intention is to allow you to do just that, employing the joysticks that can be cheaply purchased as accessories to your Dragon.

MODULE 3.2.1

```
1000 REM***************************
1010 REM EXECUTE DRAWING
1020 REM***************************
1030 PMODE 0,1:PCLS:SCREEN 1,1
1040 LET X=2:LET Y=2
1050 FOR I=0 TO 3:LET J(I)=JOYSTK(I):NEX
T
1060 IF INKEY$="S" THEN GOSUB 6000
1070 LET X=X-2*(J(2)>56)+2*(J(2)<6):LET
X=X+2*(X>254)-2*(X<2)
1080 LET Y=Y-2*(J(3)>56)+2*(J(3)<6):LET
Y=Y+2*(Y>190)-2*(Y<2)
1090 PSET(X,Y,1)
1100 PSET(X,Y,(1+(PEEK(65280)<>255)))
1110 IF INKEY$="C" THEN PCLS
1120 GOTO 1050
```

You should immediately recognise this as a variety of the cursor move module which you came across in the last program. Its purpose is to allow the user to move a small dot around the screen, inking in or deleting lines along the way. In this module apart from the simple techniques necessary to use the joysticks, we also make use of the PMODE and SCREEN commands for the first time.

These commands seem quite awesome at first sight but their use is really quite simple. The plain fact is that the more individual dots a computer is capable of placing on the same sized screen, the more memory must be devoted to remembering just where those dots actually are. For most applications, the kind of memory necessary to create and sustain a display of up to 256*192 pixels is simply an expensive luxury. We noted in the course of entering the Unifile program that 4,500 memory spaces could be saved by cutting down the amount of memory devoted to the screen. All that PMODE really does is to specify how many dots the Dragon will be capable of drawing on the screen and consequently their size, since however many dots may be printed they will always end up filling the screen if they are all of them present at the same time. The number of dots and their size is shown in the table on page 93 of the Dragon manual. PMODE also sets, by the way, the place in the memory where the image on the screen is to be stored but this need not concern us at the moment. The second figure in our PMODE commands will be I.

In this program we have chosen to use PMODE 0,1. We could equally have chosen to use PMODE I,I which would have made available more colours but increased the amount of memory necessary, since the more colours a particular point of the screen may be set, the more there is to remember about that point. The smallest dot (or pixel) which we shall be able to draw in this PMODE is actually composed of four of the pixels which would be available in PMODE 4 which is the highest resolution available.

Having set the PMODE and cleared the area of memory which will be used (with PCLS) it only now remains to set the SCREEN. It is probably easier to remember the function of SCREEN if you actually think of it as WINDOW, for that is the function that it performs.

At any time you have at least two windows available to you called SCREEN I and SCREEN 0. SCREEN 0 looks out onto the area of memory which stores what is displayed on the text screen (that is the type of display we have been using up to now), while SCREEN I looks out onto the area of memory used to store any displays created when one of the PMODEs is in operation. The only other complication is that our window can be set to two colours, that is to say that it can look at a design in two ways and, though the same design will be seen, a different set of colours will come through. Thus, SCREEN 1,0 means 'look through the window at that part of the memory which PMODE says is in use and interpret what is there using colour set zero'. Depending on the PMODE in use, colour set zero will either consist of black and green or of green, yellow, blue and red.

There is another colour set, 1, which will either be black and buff or buff, cyan, magenta and orange.

With that brief introduction in mind we turn to examine the program lines that actually use these commands.

Commentary

Line 1030: Bearing in mind what has been said above, you should have no difficulty interpreting this line. All that happens is that we choose a PMODE where the number of pixels that can be set is 128 across by 96 down. We clear the area of memory that will store this screen (try not clearing it to see the necessity for this) then we look through the window that points to this part of the memory — before running the program the Dragon was showing the view through SCREEN 0, as it always does when not instructed otherwise.

To satisfy yourself that SCREEN has no other function but to look at a part of the memory, you might like to try the experiment of removing this instruction, running the module (when you have debugged it) and playing with the joystick, stopping the program and then inserting temporary line 9999SCREEN 1,1:GOTO 9999. Now GOTO 9999 (not RUN) and you will see the design you were creating in the memory but which was invisible because you were looking through the wrong screen.

Line 1050: This line reads the four joystick inputs, each of which provides a value between 0 and 63, depending on the position of the joystick handle. Due to some strange quirk, although only one joystick is connected (in the case of this program the left) all the values for the two possible joysticks must be read or the result is nonsense.

Lines 1070–1080: Here the X and Y co-ordinates of the left-hand joystick are used to move the dot with which a line is drawn. Note that the move is two positions at a time. This is necessary because no matter what the size of the smallest point in the current PMODE, the screen is always defined as being 255*192 when it comes to specifying addresses. Since our pixel is actually 2*2, a single move involves a move of 2 spaces in terms of its address.

Lines 1090–1100: When the moving dot arrives in a particular position that position is PSET, that is, switched on to show colour 1 (green). The next line then reads the memory location which registers whether the button on the joysticks is being pressed, and uses the value obtained in a condition, whose value is subtracted from the colour. If the button is being pressed, the value PEEKed will not be 255 and the condition will be true — the condition is actually PEEK (65280) < > 255 — and the point will be recoloured in colour zero (black) and will disappear. In this manner lines can be erased.

Line 1110: Pressing C will result in the screen being cleared.

Testing Module 3.2.1

Using the module as it stands you should be able to doodle a design on the screen, inking in lines or erasing them at will.

MODULE 3.2.2

```
6000 REM****************************
6010 REM DATA FILES
6020 REM****************************
6030 MOTOR ON:AUDIO ON:CLS:INPUT "POSITI
ON TAPE THEN PRESS enter  <MOTOR IS ON>:
";Q$
6040 MOTOR OFF:INPUT "PLACE RECORDER INT
O CORRECT MODETHEN PRESS enter:";Q$
6050 PRINT "FUNCTIONS AVAILABLE:","1)SAV
E DESIGN",,"2)LOAD DESIGN":INPUT "WHICH
 DO YOU REQUIRE:";Q:ON Q GOTO 6070,6100
6060 RETURN
6070 MOTOR ON:FOR I=1 TO 10000:NEXT:CSAV
EM "PIC",1536,7679,6144
6080 SCREEN 1,0:AUDIO OFF
6090 RETURN
6100 PCLS:CLOADM "PIC",0
6110 AUDIO OFF:SCREEN 1,0:RETURN
```

Whether or not this module will be of much use to you will depend on whether you wish to save high resolution designs, created by this program or some other of your own devising. It is included with this program purely because it uses a facility which is shamefully neglected in the manual supplied with the Dragon (it's not the only thing to receive that treatment as you'll no doubt have found to your cost). The facility I refer to in this case is known as CSAVEM, which is actually meant to enable a user to store machine-code programs onto tape but whose function is simply to save a chunk of memory straight on to tape. Since the design that has been created using this program is merely a chunk of memory, we can save it and reload it whenever we wish — and the same process can be applied to any other design produced in high resolution modes.

Commentary

Line 6070: The CSAVEM command only requires that you specify the memory address to start saving, the memory address to finish and the total number of bytes (or memory locations) involved. The video memory for high resolution begins at 1536 and, in the highest resolution PMODE, continues up to address 7679. Although we do not need all that space for PMODE 0, it was actually reserved when we switched the Dragon on and we have not reduced the amount by use of PCLEAR, so it is safe to save it without interfering with any of the memory used by the actual program.

This routine, with these figures, is therefore good for any PMODE where video memory is set to start in the first possible location i.e. up to PMODE 4,1.

Line 6100: The reverse of CSAVEM is CLOADM, which loads from tape some data and places it into a specific area of memory specified by the user. The position into which the data is loaded is specified by the offset figure in the CLOADM command. In our case we want to load the data back into screen memory, so the offset is zero, which results in anything loaded being placed into exactly the same place as it originally came from.

Line 6110: To emphasise the point about SCREEN, notice how the reloaded picture appears instantaneously when this command is reached.

Testing Module 3.2.2

You should now be able to create a design using the first part of the program and then to save it to tape. Stop the program then RUN it again and call up this module to reload the data you have just saved. You should find that your original design appears on the screen immediately after loading has ceased.

Summary

This program is an indication of the benefit to be gained from keeping an eye out for simple techniques to accomplish tasks that you have set yourself — even if they do sometimes arise from unusual sources. Machine-code programs, and the techniques associated with them are completely outside the scope of this book, and yet the simplest method of storing a picture that you will find comes straight out of the set of commands provided for use with machine code. The moral is that almost anything you can learn about your Dragon, no matter how obscure it may seem at the time, may well come in useful at some future date.

DOODLE: Summary of one-key commands

S	execution of program is diverted to data file module.
C	screen is cleared.

3.3 TANGRAMS

This program will both serve to allow you to play the ancient Chinese shape game of Tangrams and to introduce the subject of the Dragon's outstandingly useful DRAW command. Using this command we shall take a potentially long and complex program on most other home micros and reduce its length dramatically since almost all arduous calculation about angles is performed automatically.

If you are not familiar with the basic idea behind DRAW, you will find it described in Chapter 10 of the Dragon manual and it would be a good idea to refer back to that before attempting to enter this program.

MODULE 3.3.1

```
1000 REM******************************
1010 REM INITIALISE
1020 REM******************************
1030 CLEAR 15000:PMODE 4,1:PCLS:SCREEN 1
,0
1040 LET ANGLE$="UERFDGLHUERFDGLH":LET R
O=1:LET F=1
1050 DIM PATTERN$(6):LET S=2
1060 LET X=129:LET Y=96
1070 GOSUB 4000
```

Nothing in this initialisation module should now be beyond you. The function of the variables will be explained during the next module.

MODULE 3.3.2

```
4000 REM****************************
4010 REM ROTATION
4020 REM****************************
4030 IF F<>1 THEN LET S=1
4040 LET SHORT=INT(10*(SQR(2)^(S))+.5):L
ET LONG=SHORT
4050 IF ROTATE/2<>INT(ROTATE/2) THEN LET
 LONG=INT(LONG*SQR(2)+.5):LET SHORT=LONG
/2
4060 LET LONG=2*INT(LONG/2):LET SHORT=2*
INT(SHORT/2)
4070 IF F=1 THEN LET D$="B"+MID$(ANGLE$,
RO,1)+STR$(SH/2)+";"+MID$(ANGLE$,3+RO,1)
+STR$(SH)+";"+MID$(ANGLE$,6+RO,1)+STR$(L
O)+";"+MID$(ANGLE$,1+RO,1)+STR$(SH)+";B"
+MID$(ANGLE$,4+RO,1)+STR$(SH/2):RETURN
4080 IF F=2 THEN LET D$="B"+MID$(ANGLE$,
RO,1)+STR$(SH/2)+";"+MID$(ANGLE$,3+RO,1)
+STR$(SH)+";"+MID$(ANGLE$,6+RO,1)+STR$(L
O)+";"+MID$(ANGLE$,7+RO,1)+STR$(SH)+";"+
MID$(ANGLE$,2+RO,1)+STR$(LO)+";B"+MID$(A
NGLE$,4+RO,1)+STR$(SH/2):RETURN
4090 IF F=3 THEN LET D$="B"+MID$(ANGLE$,
RO,1)+STR$(SH/2)+";"+MID$(ANGLE$,5+RO,1)
+STR$(SH)+";"+MID$(ANGLE$,6+RO,1)+STR$(L
O)+";"+MID$(ANGLE$,1+RO,1)+STR$(SH)+";"+
MID$(ANGLE$,2+RO,1)+STR$(LO)+";B"+MID$(A
NGLE$,+RO,1)+STR$(SH/2):RETURN
4100 IF F=4 THEN LET D$="B"+MID$(ANG$,RO
,1)+STR$(SH)+";"+MID$(ANG$,2+RO,1)+STR$(
SH)+";"+MID$(ANG$,4+RO,1)+STR$(LO)+";"+M
ID$(ANG$,6+RO,1)+STR$(LO)+";"+MID$(ANG$,
RO,1)+STR$(LO)+";"+MID$(ANG$,2+RO,1)+STR
$(SH)+";8"+MID$(ANG$,4+RO,1)+STR$(SH):RE
TURN
```

I'm afraid that there is no getting away from the fact that this is a dense and complex module, and one which is quite difficult to enter without errors. Fortunately, by its very nature, the error messages it will generate will give a very good clue as to where the errors lie. The function of the module is to calculate certain variables and then, on the basis of these to construct strings which can be used to draw the three types of geometrical shapes used in Tangrams: triangles, parallelograms and squares.

Commentary

Line 4030: The variable F will be used to store the type of figure to be drawn. 1 means a triangle, 2 and 3 mean parallelograms (two types to take account of the fact that a parallelogram is not symmetrical and thus is a different shape when it is turned over) while 4 indicates a square. The variable S refers to size and will be explained in relation to the following lines.

Lines 4040–4050: These two lines are necessary to cope with the fact that the triangles in Tangrams are of three sizes, each twice the area of the next size down and also the problem that arises out of the fact that lines DRAWn the same length take on different lengths according to the angle at which they are placed on the screen. Consider the example of a triangle drawn according to the following instruction U10;F10;L10. According to the instructions given in the string it would appear that all the sides should be the same length and yet examination of the directions specified will show that what would be DRAWn is a right angled triangle, where clearly the hypotenuse is longer than the other two sides.

The solution to this apparent paradox is that all three sides of the triangle do contain the same number of pixels, but that the pixels themselves, because they are laid out on a rectangular grid on the screen, are further apart diagonally than they are up or across. The upshot of this is that to DRAW even a simple figure like a triangle on the screen and then to rotate it through 45 degrees, as this program is capable of doing, the length of the sides in pixels must be recalculated each time.

These two lines begin by calculating the length of the short side of a triangle when DRAWn with the hypotenuse diagonal on the screen — the lengths of the sides of each size of triangle are SQR(2) * the length of the sides of the next size down. The hypotenuse (LONG) is the same length in pixels.

In the second line, account is taken of the effect of drawing the triangle at an orientation which makes the hypotenuse vertical or horizontal. In this case, the length of the hypotenuse must be again multiplied by SQR(2) to achieve a sensible result (i.e. a triangle with the same area).

Since the square's sides and the long side of the parallelogram are the same length as the hypotenuse of the smallest triangle, these two lines have also dealt with them.

Line 4060: This line ensures that there are an even number of pixels in the side to be printed, which in turn ensures that the drawing position ends up at the same place that it started when the figure is finished.

Lines 4070–4100: These lines appear very daunting but all they in fact do is:

1) specify a blank move from the current draw position to the perimeter of the figure to be drawn.
2) alternating between short and long sides, as defined by lines 4040 and 4050, add the necessary figures and the directions for each side to the string which is being built up.
3) specify a blank move back to the initial drawing position once the figure is completed.

You will note that the actual directions are not specified, they are letters taken from the string ANGLE$ on the basis of the variable RO, which as can be seen from the earlier part of the module, is short for ROTATE. For any given figure, the relative positions of the letters in ANGLE$ which indicate the directions of the sides will always remain the same but the starting point, that is the direction of the first side, will differ according to the degree of rotation. Thus at the end of whichever of these lines is executed a string called D$ will have been created, capable of drawing one of the figures at a size specified by S and at an orientation specified by RO. Nothing in the string will indicate where the figure is to be drawn, or the colour.

Testing Module 3.3.2

To test the module it will be necessary to enter line 2030 of the program and then to enter a temporary line 2040 GOTO 2040. Running the program should now result in the drawing of a small triangle. To test the drawing of the other figures, the value of F can be altered to any figure from 2 to 4. Rotation can be checked by altering RO to read anywhere from 1 to 8. Size of the triangles should be capable of ranging from 1 to 3 as represented by the variable S.

MODULE 3.3.3

```
2000 REM****************************
2010 REM DRAWING ROUTINE
2020 REM****************************
2030 DRAW "C1;BM"+STR$(X)+","+STR$(Y)+";
"+D$
2040 LET T$=INKEY$:IF T$="" THEN GOTO 20
40
2050 LET D$="C0;BM"+STR$(X)+","+STR$(Y)+
";"+D$:DRAW D$
2060 LET X=X-(T$=CHR$(9))+(T$=CHR$(8))-1
0*(T$=".")+10*(T$=",") :IF X>225 THEN LET
 X=225
2070 IF X<29 THEN LET X=29
2080 LET Y=Y-(T$=CHR$(10))+(T$=CHR$(94))
-10*(T$="A")+10*(T$="Z"):IF Y>162 THEN L
ET Y=162
2090 IF Y<29 THEN LET Y=29
2100 IF T$="R" THEN LET ROTATE=ROTATE+1+
8*(ROTATE>8)
2110 IF F=1 THEN LET S=S-(T$="S"):IF S>4
 THEN LET S=S-3
2120 IF T$>"0" AND T$<"5" THEN LET F=VAL
(T$)
2130 IF T$="D" OR T$="C" OR T$=CHR$(13)
THEN GOSUB 3000
2140 GOSUB 4000:GOTO 2030
```

This is the main loop of the program and you will recognise some features shared with a cursor move module. The function of the module is to allow the current shape to be moved around the screen, rotated, printed permanently and recorded, or for another figure to replace the current one.

Commentary

Line 2030: A section is tagged onto D$ specifying that drawing will start at the position indicated by the co-ordinates X and Y and that the colour in which it shall be drawn is 1 or green.

Line 2040: A waiting state until a key is pressed.

Lines 2060 – 2090: The limits to movement expressed in these four lines express the need to ensure space for the largest possible figure to be drawn.

Line 2100: Input of R rotates the figure through 45 degrees clockwise.

Line 2110: If the current figure is a triangle, input of S will shuttle the variable S through the range 1-3.

Line 2120: Input of a number in the range 1 to 4 will select the corresponding figure.

Line 2130: Input of D or C or ENTER will result in calling the next module.

Testing Module 3.3.3

You should now be able to move, rotate, exchange or, in the case of triangles, to change the size of your figure at will.

MODULE 3.3.4

```
3000 REM***************************
3010 REM REDRAW CURRENT PATTERN
3020 REM***************************
3030 IF NN<7 AND T$=CHR$(13) THEN MID$(D
$,2)="1":LET PATTERN$(NN)=D$:LET NN=NN+1
:RETURN
3040 PCLS:FOR I=0 TO NN-1
3050 DRAW PATTERN$(I)
3060 IF T$="D" THEN LET T1$=INKEY$:IF T1
$="" THEN GOTO 3060
3070 IF T$="▶" AND T1$="▶" THEN FOR J=I
TO NN-1:LET PATTERN$(J)=PATTERN$(J+1):NE
XT J:LET NN=NN-1:LET T$="":GOTO 3040
3080 NEXT I:RETURN
```

The purpose of this module is to allow shapes to be permanently stored in an array so that if they are erased by another shape being moved across them, they can be redrawn. The module also accomplishes such a redrawing of all permanently entered figures and allows their deletion.

Commentary

Line 3030: The actual Tangram game is played with seven pieces — two small triangles, one medium, two large, one square and one parallelogram. This line enters the current figure, including its position, into the string array PATTERN$, provided that seven pieces have not already been used.

Lines 3040–3080: In this loop, each figure contained in PATTERN$ is redrawn on the cleared screen. If C has been input, that is all that is done. If D has been input then the user is given the opportunity to delete the figure just drawn by inputting D again. Pressing any other key leaves the figure in the array.

Testing Module 3.3.4

You should now be able to enter figures permanently into the array, to redraw the pattern if it is corrupted by the movement of figures and to delete figures from the design built up so far.

Summary

The speed and simplicity of this program are a tribute to the Dragon's abilities. Very few other home micros would be able to cram so many functions into a program of this size. The program is also an indication of the sheer flexibility of the DRAW command when applied to strings which are created in the program rather than having to be specified before the program is run.

Going Further

1) This program could easily be expanded to make allowance for other types of shape — a few more lines in Module 2 is all that it would take.
2) More complex designs could be built up if the size of PATTERN$ were to be increased.
3) No check is made, if you do want to play Tangrams, that the correct pieces are being used — only that no more than seven are in the design. Could you add such a check?
4) If you are proud of the designs you have created, or of your solutions to Tangram problems, you may well want to add a data file module to the program.

TANGRAMS: Summary of one-key commands

↑ and Q	move current figure 1 and 10 pixels upwards respectively.
↓ and A	move the current figure 1 and 10 pixels down respectively.
> and →	move the current figure 1 and 10 spaces to the right respectively.
< and ←	move the current figure 1 and 10 spaces to the left respectively.

R	rotates the current figure on the screen.
S	alters the size of the current figure if it is a triangle.
1-4	specifies the type of figure to be drawn.
C	redraws the total design so far.
ENTER	places the current figure, at its current position, into the array of pieces.
D	calls up the deletion function: pieces are displayed one by one with the opportunity to delete them by further input of D. Any other key leaves a piece in permanent array.

3.4 DESIGNER

I have a special fondness for this program simply because the ideas on which it is based are not my own: they were taken from an excellent book, *The Principles of Interactive Computer Graphics* by William M. Newman and Robert F. Sproull. The reason that I say fondness is that the program serves as a reminder to me of how much there is always to learn about the principles of programming and how many fields lie waiting to be opened up for no more cost than the price of a few books. Based on two simple procedures from that book, this program will allow you to define a design of up to 10,000 by 10,000 pixels in size, to examine that design at various scales and to rotate all or part of it on the screen. Once its use is mastered it is capable of being used in a variety of applications where it is desirable to be able to change and manipulate designs quickly and easily.

MODULE 3.4.1

```
6000 REM****************************
6010 REM DATA FILES
6020 REM****************************
6030 MOTOR ON:AUDIO ON:CLS:INPUT"POSITIO
N TAPE THEN PRESS enter  (MOTOR IS ON):"
;Q$
6040 MOTOR OFF:INPUT "PLACE RECORDER IN
CORRECT MODE  THEN enter:";Q$
6050 PRINT:PRINT "FUNCTIONS AVAILABLE:",
"1)SAVE DESIGN",,"2)LOAD DESIGN":INPUT "
WHICH DO YOU REQUIRE:";Q:ON Q GOTO 6070,
6130
6060 RETURN
6070 MOTOR ON:FOR I=1 TO 10000:NEXT
6080 OPEN "O",£-1,"DESIGN"
6090 PRINT £-1,LL
6100 FOR I=0 TO LL-1:FOR J=0 TO 3:PRINT
£-1,COORDS(I,J):NEXT J,I
6110 CLOSE £-1
6120 RETURN
6130 OPEN "I",£-1,"DESIGN"
6140 INPUT £-1,LL
6150 FOR I=0 TO LL-1:FOR J=0 TO 3:INPUT
£-1,COORDS(I,J):NEXT J,I
6160 CLOSE £-1
6170 RETURN
```

A standard data file module.

MODULE 3.4.2

```
1000 REM****************************
1010 REM INITIALISE
```

```
1020 REM****************************
1030 PCLEAR4:PCLS:PMODE4,1
1040 LET UPPER=0:LET LOWER=191:LET LEFT=
0:LET RIGHT=255
1050 DIM COORDS(200,4)
1060 LET T1=-1
1070 DIM X1(4,4):DIM X2(4,4)
1080 DRAW "BM0,0;F4;BU4;G4":GET (0,0)-(4
,4),X1,G:PCLS
```

The initialisation module — the variables will be explained as they are used.

Commentary

Lines 1070–1080: This is our first use of the Dragon's GET command, which allows a specified area of the screen to be read into an array and later replaced anywhere on the screen by the use of PUT. In this case what is read from the screen is a small x which will be used later as a cursor but this process cannot be seen as SCREEN has not yet been set to point to the graphics area of memory.

Testing Module 3.4.2

This module cannot be properly tested until other modules have been entered.

MODULE 3.4.3

```
2000 REM****************************
2010 REM MAIN PROGRAM
2020 REM****************************
2030 LET X=LEFT+128:LET Y=UPPER+96:LET R
IGHT=LEFT+255:LET LOWER=UPPER+191
2040 SCREEN 1,0
2050 LET L1=X-LEFT:LET U1=Y-UPPER
2060 FOR I=0 TO LL-1
2070 LET X1=COORDS(I,0)
2080 LET Y1=COORDS(I,1)
2090 LET X2=COORDS(I,2)
2100 LET Y2=COORDS(I,3)
2110 GOSUB 4000
2120 NEXT I
2130 IF T1<>-1 AND T1>=LEFT+2 AND T1<LEF
T+254 AND T2>=UPPER+2 AND T2<UPPER+190 T
HEN CIRCLE(T1-LEFT,T2-UPPER),2
2140 LET T$=INKEY$:IF T$<>"" THEN GOTO 2
210
2150 GET (L1-2,U1-2)-(L1+2,U1+2),X2,G
2160 FOR I=1 TO 25:NEXT I
2170 PUT (L1-2,U1-2)-(L1+2,U1+2),X1,OR
2180 FOR I=1 TO 25:NEXT I
2190 PUT (L1-2,U1-2)-(L1+2,U1+2),X2,PSET
2200 GOTO 2140
2210 IF T$>="0" AND T$<"4" THEN LET POWE
R=VAL(T$)
2220 IF T$="M" THEN LET MO=1
2230 IF T$="X" THEN LET MO=0
2240 LET TEMP=0
2250 LET TEMP=TEMP-10^POWER*(T$=CHR$(9))
+10^POWER*(T$=CHR$(8))
2260 IF MO=1 THEN LET LEFT=LEFT+TEMP ELS
E LET X=X+TEMP
2270 IF LEFT<0 THEN LET LEFT=0
2280 IF LEFT>9745 THEN LET LEFT=9745
2290 IF X-LEFT>253 THEN LET X=LEFT+253
2300 IF X-LEFT<2 THEN LET X=LEFT+2
2310 LET TEMP=0
2320 LET TEMP=TEMP-10^POWER*(T$=CHR$(10)
)+10^POWER*(T$=CHR$(94))
2330 IF MO=1 THEN LET UPPER=UPPER+TEMP E
LSE LET Y=Y+TEMP
```

```
2340 IF UPPER<0 THEN LET UPPER=0
2350 IF UPPER>9745 THEN LET UPPER=9745
2360 IF Y-UPPER>189 THEN LET Y=UPPER+189
2370 IF Y-UPPER<2 THEN LET Y=UPPER+2
2380 IF T$="F" THEN LET T1=X:LET T2=Y
2390 IF T$="T" AND T1<>-1 THEN LET COORD
S(LL,0)=T1:LET COORDS(LL,1)=T2:LET COORD
S(LL,2)=X:LET COORDS(LL,3)=Y:LET T1=-1:L
ET LL=LL+1
2400 IF T$="R" OR T$="D" THEN GOSUB 3000
2410 IF T$="S" THEN GOSUB 6000
2420 PCLS
2430 IF MO=1 THEN GOTO 2030 ELSE GOTO 20
40
```

This main program module allows a flashing cursor to be moved around the screen, the area of the design to which the screen points to be moved, lines to be defined and later modules to be called.

Commentary

Line 2030: X and Y are the co-ordinates of the flashing cursor. UPPER, LOWER, LEFT and RIGHT are the addresses of the boundaries of the screen, expressed in terms of the overall 10000 * 10000 pixel space available for the design. The screen area starts in the upper right-hand corner of the total area available.

Line 2050: Ll and Ul are the co-ordinates of the cursor on the screen.

Lines 2060–2120: As the program progresses, the lines entered into the design will be stored in the array COORDS. This loop will print out each line entered so far, when necessary, calling up a later module to do the actual drawing of lines.

Line 2130: Tl and T2 represent the address of the start of a line that is currently being defined. If T1 is set to −1 it means that no line is currently being defined. If Tl is not equal to −1 and the co-ordinates represented by Tl and T2 fall within the boundaries of the screen, then a small circle is printed at the location.

Lines 2140–2200: This section again makes use of GET to save what is on the screen in the place where the cursor is to be printed. The cursor is now PUT onto the screen with the OR attribute set — in other words its printing will not erase anything that is already there on the screen in that position. Then the original contents of the location are restored by PUTting back what was saved in line 2150. Note that because both GET instructions terminated with G, meaning 'store full graphic detail', the PUT instructions have also to have an attribute specified, such as OR or PSET — even though the latter only means 'put what is in the array on the screen'.

Line 2210: Input of a number in the range zero to three, sets a variable called POWER which will later be used to determine the move to be made by cursor or screen. The move will be 10 to the power of POWER pixels (i.e. 1 to 1000).

Line 2220: Input of M is stored in the variable MO (for move) is interpreted as meaning that any move specified will be a move of the screen across the design.

Line 2230: Conversely, input of X is interpreted as meaning that any move will be a move of the cursor across the screen.

Lines 2240–2370: The cursor co-ordinates or the screen co-ordinates are changed according to the above rules if one of the arrowed keys is input.

Line 2380: Input of F defines the beginning of a line to be drawn at the point currently occupied by the cursor.

Line 2390: Input of T defines the destination of a line, provided that the co-ordinates of the origin of the line have already been entered.

Line 2400: Inputs of R or D specify rotation or deletion of parts of the design and require the calling of other modules.

Line 2410: Input of S results in the calling of the data file module.

Line 2430: The design is redrawn, with the cursor reset in the middle of the screen if MO indicates that the screen move mode is set.

Testing Module 3.4.3

You should now be able to move the cursor around the screen, but not many more meaningful tests are possible until the later modules are entered. You may at least satisfy yourself that the co-ordinates of lines are being entered by defining some starts and finishes and checking that the addresses have been placed in the array COORDS. Note that a temporary line 4000 RETURN will be necessary.

MODULE 3.4.4

```
4000 REM*************************
4010 REM DRAW LINES
4020 REM*************************
4030 LET LOWER=UPPER+191:LET RIGHT=LEFT+
255
4040 IF (X1<LEFT AND X2<LEFT) OR (X1>RIG
HT AND X2>RIGHT) OR (Y1>LOWER AND Y2>LOW
ER) OR (Y1<UPPER AND Y2<UPPER) THEN LET
OUT=1:RETURN
4050 IF Y1<UPPER THEN LET EDGE=UPPER
4060 IF Y1>LOWER THEN LET EDGE=LOWER
4070 IF Y1<UPPER OR Y1>LOWER THEN LET X1
=X1+(X2-X1)*(EDGE-Y1)/(Y2-Y1):LET Y1=EDG
E
4080 IF Y2<UPPER THEN LET EDGE=UPPER
4090 IF Y2>LOWER THEN LET EDGE=LOWER
4100 IF Y2<UPPER OR Y2>LOWER THEN LET X2
=X2+(X1-X2)*(EDGE-Y2)/(Y1-Y2):LET Y2=EDG
E
4110 IF X1>RIGHT THEN LET EDGE=RIGHT
4120 IF X1<LEFT THEN LET EDGE=LEFT
4130 IF X1<LEFT OR X1>RIGHT THEN LET Y1=
Y1+(Y2-Y1)*(EDGE-X1)/(X2-X1):LET X1=EDGE
4140 IF X2>RIGHT THEN LET EDGE=RIGHT
4150 IF X2<LEFT THEN LET EDGE=LEFT
4160 IF X2>RIGHT OR X2<LEFT THEN LET Y2=
Y2+(Y1-Y2)*(EDGE-X2)/(X1-X2):LET X2=EDGE
4170 IF X1-LEFT>=0 AND X2-LEFT>=0 AND X1
-LEFT <=255 AND X2-LEFT<=255 AND Y1-UPPE
R>=0 AND Y2-UPPER>=0 AND Y1-UPPER<=191 A
ND Y2-UPPER<=255 THEN LINE(X1-LEFT,Y1-UP
PER)-(X2-LEFT,Y2-UPPER),PSET
4180 RETURN
```

The purpose of this module is to take two sets of co-ordinates, X1/Y1 and X2/Y2 and to decide whether any part of a line drawn between the two points so defined would pass across the screen as it is now placed. If any part of the line would fall upon the screen it is drawn, otherwise it is ignored.

Commentary

Line 4040: If both XI and X2 or both YI and Y2 are off the screen in the same direction then no part of the line can fall onto the screen.

Lines 4050–4060: If a line starts above or below the area covered by the screen, these two lines reset the variable EDGE to coincide with the top or bottom of the screen.

Line 4070: For lines which begin above or below the screen, this line calculates the horizontal position at which the line will pass through the top or bottom edge. The formula in the first half of the line says nothing more complex than that if, for instance, the line in question passes through the top edge of the screen halfway through its vertical component, it will also be halfway through its horizontal component. Clearly this will only hold true for straight lines.

Lines 4080–4160: The same kind of process is carried out with regard to variables Y2, Xl and X2.

Line 4170: Since it's possible for a line not to lie entirely above, below or to one side of the screen and yet still not pass across the screen itself, this line makes one final check that the co-ordinates calculated do in fact lie entirely on the screen and, if they do, draws the line indicated.

Testing Module 3.4.4

You should now be in a position to define lines and see them drawn on the screen and also to move the screen over the design. You could do that before, it's simply that you couldn't see it happening.

MODULE 3.4.5

```
3000 REM***************************
3010 REM SCALE/ROTATE/DELETE
3020 REM***************************
3030 CLS:PCLS:SCREEN 0,1
3040 INPUT "ANGLE THROUGH WHICH DESIGN I
S TOBE ROTATED:";ANGLE:LET ANGLE=ANGLE*3
.1415926/180
3050 PRINT:INPUT "SCALE FACTOR TO DIVIDE
          DIMENSIONS BY:";SCALE:IF SCALE
=0 THEN LET SCALE=1
3060 SCREEN 1,0
3070 FOR I=0 TO LL-1
3080 LET X3=(COORDS(I,0)-X)/SCALE
3090 LET Y3=(COORDS(I,1)-Y)/SCALE
3100 LET X4=(COORDS(I,2)-X)/SCALE
3110 LET Y4=(COORDS(I,3)-Y)/SCALE
3120 LET X1=X+INT(X3*COS(ANGLE)+Y3*SIN(A
NGLE))
3130 LET Y1=Y+INT(Y3*COS(ANGLE)-X3*SIN(A
NGLE))
3140 LET X2=X+INT(X4*COS(ANGLE)+Y4*SIN(A
NGLE))
3150 LET Y2=Y+INT(Y4*COS(ANGLE)-X4*SIN(A
NGLE))
3160 GOSUB 4000
3170 IF T$<>"D" THEN GOTO 3220
3180 LET T1$=INKEY$:IF T1$="" THEN GOTO
3180
3190 IF T1$="D" THEN FOR J=I TO LL-1:FOR
 K=0 TO 3:LET COORDS(J,K)=COORDS(J+1,K):
NEXT K:NEXT J:LET LL=LL-1
3200 IF T1$="D" THEN LET I=I-1
3210 IF T1$="Q" THEN RETURN
3220 NEXT I
3230 IF INKEY$="" THEN GOTO 3230
3240 RETURN
```

The purpose of this module is to reproduce the design on a smaller or larger scale, as specified, and to rotate it around the current cursor position. The module is much less complex than it looks at first sight.

Commentary

Line 3040: Rotation is input in degrees and translated into radians.

Lines 3080–3110: The co-ordinates of the start and finish of each line are recalculated in terms of their distance from the cursor position and simply divided by the scale specified. Note that it is perfectly possible for the scale to be less than one, thereby magnifying the design.

Lines 3120–3150: The procedure for moving a point with, for instance, co-ordinates of X and Y through angle A, is to apply the formula X2=X*COS A + Y* SIN A and Y2=X*SIN A + Y*COS A. This is applied in theselines to X3, Y3, X4 and Y4, which are the co-ordinates in terms of the cursor position. When these altered variables are added to X1 and Y1, they define a scaled and rotated pattern.

Lines 3170–3230: If this module has been called by the input of D during the course of the previous module, then the lines are drawn one by one, giving the user the opportunity to input D against any which are to be deleted. Input of Q at any time, returns to the main module. When the scaled and rotated design is finished on the screen, it remains until a key is depressed before returning to the main module.

Testing Module 3.4.5

You should now be able to reproduce any lines entered to a specified scale and rotated through any desired angle. You should also be able to call up this module for the purpose of deletions.

Summary

Given a little imagination, this program can be a useful tool in a variety of applications. You can plan layouts, draw maps or simply mess about. In fact, with a program like this one loaded you can make your Dragon simulate many of the capabilities of far more expensive graphics computers beloved of engineers and scientists in many fields.

The program is also a reminder of the wealth of ideas that lie waiting to be translated into action from the wide variety of books on computing that are available today.

Going Further

1) The end of Module 3 has been left a little messy. It works properly but it is cumbersome in th t it requires the redrawing of all the lines on each move of the cursor. What are the conditions that would have to be satisfied in order to make it practical to skip the redrawing of the lines? It's not as straightforward as it looks at first.

DESIGNER: Summary of one-key commands

With flashing cursor:

0-3	sets cursor move to equivalent power of 10.
M	next move specified will be move of screen over design.
X	next move specified will be of cursor within screen limits.
F	line to be drawn starting from cursor position.
T	line to be drawn to this position (only after F).
S	call data file module.

R	call module which scales or rotates design.
D	as above but with option to delete individual lines.

Arrowed keys — appropriate move of cursor or screen.

After initial input of D:

D	delete line just drawn.
Q	return to main module.

CHAPTER 4
Easy education

In this chapter we shall consider three programs which enable the Dragon to make its contribution in the field of home education. The first of these, MultiQ, is a program designed to allow the user to input a series of questions and answers, which are then used as the basis of randomly generated multiple choice tests. The second program is Words, a basic reading tutor and, lastly, Where? teaches the locations of cities in any country in the world you care to program in.

The object of the programs is to give you some idea of what can be accomplished in the field without too much effort. Even so, unless you intend to buy a range of software on cassette, with specialist programs dedicated to individual subjects and coming complete with their own files of data, the usefulness of your educational data will always depend on the amount of work you are prepared to put into them. The best multiple choice question program in the world is not much use unless at some stage you are prepared to sit down and feed in enough questions to make it interesting.

If you are prepared to give such programs the data to work with, they can often be spectacularly successful for the simple reason that they work at the pace of the student, show no signs of impatience, give no reward for short cuts or cheating and are always ready for just one more try at any time of day or night.

4.1 MULTIQ

This program is a favourite of mine. When I wrote it I was satisfied that it was a competent piece of work that would do the job that it was designed for. It was not until I entered a mass of questions and answers and tried it out on people that I realised that such programs make learning as addictive as any game.

Like Unifile, this program is a chameleon, designed to change its colour to suit your need. At one moment you may wish it to be a French tutor, offering a variety of French words as possible translations for an English word. Later on you may have it asking fairly complex questions on 19th century history, giving a series of dates as possible answers. The aim of the program is to enable you to do all these and more without having to make changes in the program itself.

MODULE 4.1.1

```
6000 REM****************************
6010 REM DATA FILES
6020 REM****************************
6030 AUDIO ON:MOTOR ON:PRINT:INPUT "POSI
TION TAPE THEN PRESS enter  (MOTOR IS ON
)";Q$:MOTOR OFF
6040 PRINT:INPUT "PLACE RECORDER IN CORR
ECT MODE  THEN PRESS enter";Q$
6050 PRINT:PRINT "FUNCTIONS AVAILABLE:",
"1)SAVE DATA",,"2)LOAD DATA":INPUT "WHIC
H DO YOU REQUIRE:";Q:ON Q GOTO 6070,6150
6060 RETURN
6070 MOTOR ON:FOR I=1 TO 10000:NEXT I
6080 OPEN"O",£-1,"MULTIQ"
6090 PRINT£-1,ITEMS
6100 FOR I=1 TO ITEMS-2:PRINT£-1,A$(I),B
$(I):NEXT I
6110 FORI=0 TO 9:PRINT£-1,D$(I),D(0,I),D
(1,I):NEXT I
6120 PRINT£-1,NAME$(0),NAME$(1)
6130 CLOSE£-1
6140 RETURN
6150 RUN 6160
6160 PCLEAR 1:CLEAR 18000:DIM A$(499):DI
M B$(499):DIM D(1,9):DIM D$(9):DIM NAME$
(1)
6170 OPEN"I",£-1,"MULTIQ"
6180 INPUT£-1,ITEMS
6190 FOR I=1 TO ITEMS-2:INPUT£-1,A$(I),B
$(I):NEXT I
6200 FOR I=0 TO 9:INPUT£-1,D$(I),D(0,I),
D(1,I):NEXT I
6210 INPUT£-1,NAME$(0),NAME$(1)
6220 CLOSE£-1
6230 LET A$(ITEMS-1)=CHR$(255):LET A$(0)
=CHR$(0)
6240 GOTO 1000
```

A standard data file module.

MODULE 4.1.2

```
7000 REM****************************
7010 REM FORMAT TITLES
7020 REM****************************
7030 LET P2=14-INT(LEN(F$)/2)
7040 PRINT @ 32*P1+P2,STRING$(LEN(F$)+2,
CHR$(185))
7050 PRINT @ 32*(P1+1)+P2,CHR$(185)+F$+C
HR$(185)
7060 PRINT @ 32*(P1+2)+P2,STRING$(LEN(F$
)+2,CHR$(185))
7070 RETURN
```

A standard title formatting module.

MODULE 4.1.3

```
1000 REM****************************
1010 REM MENU
1020 REM****************************
1030 CLS:LET F$="MULTIQ":LET P1=1:GOSUB
7000
1040 PRINT: PRINT "COMMANDS AVAILABLE:"
1050 PRINT "    1)INPUT NEW ITEMS"
1060 PRINT "    2)SEARCH/DELETE"
1070 PRINT "    3)ENTER NEW TYPES"
1080 PRINT "    4)GENERATE QUESTIONS"
1090 PRINT "    5)DISPLAY OR RESET SCORE"
1100 PRINT "    6)DATA FILES"
1110 PRINT "    7)INITIALISE"
1120 PRINT "    8)STOP"
1130 PRINT:INPUT "WHICH DO YOU REQUIRE:"
;Z:CLS
1140 ON Z GOSUB 2000,3000,1640,3500,4000
,6000,1500,1160
```

```
1150 GOTO 1000
1160 LET F$="MULTIQ":LET P1=6:GOSUB 7000
1170 STOP
```

A standard menu module.

MODULE 4.1.4

```
1500 REM************************
1510 REM INITIALISE
1520 REM************************
1530 PCLEAR 1:CLEAR 18000
1540 DIM NAME$(1),Q(5)
1550 DIM A$(499):DIM B$(499)
1560 LET A$(0)=CHR$(0):LET A$(1)=CHR$(25
5)
1570 DIM D$(9),D(1,9)
1580 LET ITEMS=2
1590 LET F$="TEST STRUCTURE":LET P1=1:GO
SUB 7000
1600 PRINT:INPUT "NAME FOR ANSWER:";NAME
$(0)
1610 PRINT:INPUT "NAME FOR QUESTION:";NA
ME$(1)
1620 PRINT:INPUT "ARE THESE CORRECT (Y/N
):";Q$:CLS
1630 IF Q$<>"Y" THEN GOTO 1500
1640 LET F$="TYPES":LET P1=0:GOSUB 7000
1650 PRINT TAB(19)"'ZZZ' TO QUIT";
1660 PRINT "TYPES INPUT SO FAR:-";
1670 IF TYPES<>0 THEN PRINT:FOR I=0 TO T
YPES-1:PRINT I+1;")";D$(I):NEXT I:ELSE P
RINT "NONE":PRINT
1680 IF TYPES<10 THEN INPUT "INPUT NEW T
YPE:";Q$:ELSE PRINT "ONLY 10 TYPES ALLOW
ED":FOR I=1 TO 2000:NEXT I:GOTO 1000
1690 IF Q$="ZZZ" THEN GOTO 1000 ELSE LET
 D$(TYPES)=Q$:LET TYPES=TYPES+1:CLS:GOTO
 1640
1700 GOTO 1000
```

You may notice the similarity between this module and the equivalent one in Unifile, since the object of both is to initialise variables and to store certain user- defined prompts for later use in the program. The use of the variables will be discussed in the course of the commentary on the program.

Commentary

Lines 1640–1690: Each answer may, if the user wishes, be given one of ten types whose names are user-defined. These types may be used later on to make the tests generated more difficult. The types input should reflect natural groupings into which the questions and answers fall. Types do not have to be input and, if they are not, no reference is made to types when inputting data.

Testing Module 4.1.4

You should now be able to input a format for the program, including the names of up to ten types.

MODULE 4.1.5

```
2000 REM************************
2010 REM REM INPUT OF NEW ITEMS
```

```
2020 REM***************************
2030 LET F$="NEW ITEMS":LET P1=1:GOSUB 7
000
2040 PRINT: PRINT "'ZZZ' TO QUIT."
2050 PRINT:PRINT NAME$(0);":";:INPUT T1$
2060 IF ITEMS>=500 THEN PRINT:PRINT "NO
ROOM FOR MORE ITEMS.":GOTO 2230
2070 IF T1$="ZZZ" THEN GOTO 2230
2080 PRINT:PRINT NAME$(1);":";:INPUT T2$
2090 IF D$(0)="" THEN LET T=0:GOTO 2180
2100 CLS:LET F$="TYPE":LET P1=0:GOSUB 70
00
2110 FOR I=0 TO TYPES-1:PRINT I+1;">";D$
(I):NEXT I
2120 PRINT NAME$(0);":";T1$
2130 PRINT NAME$(1);":";T2$
2140 INPUT "TYPE FOR THIS ITEM:";T
2150 CLS:LET F$="NEW ITEM":LET P1=1:GOSU
B 7000
2160 PRINT:PRINT NAME$(0);":";T1$
2170 PRINT:PRINT NAME$(1);":";T2$
2180 IF D$(0)<>"" THEN PRINT:PRINT "TYPE
:";D$(T-1)
2190 PRINT:INPUT "ARE THESE CORRECT (Y/N
):";Q$:CLS:IF Q$<>"Y" THEN GOTO 2000
2200 IF D$(0)<>"" THEN LET D(0,T-1)=D(0,
T-1)+1:LET T1$=CHR$(48+T-1)+T1$:ELSE LET
 T1$=" "+T1$
2210 GOSUB 2500
2220 LET ITEMS=ITEMS+1:GOTO 2000
2230 LET SUM=1
2240 FOR I=0 TO 9
2250 LET D(1,I)=SUM
2260 LET SUM=SUM+D(0,I)
2270 NEXT I
2280 RETURN
```

Once again the similarities between this module and the equivalent one in Unifile should be obvious. Prompts already defined by the user are used to structure what is input.

Commentary

Line 2200: The array D is used to store two sets of figures. In D(0,etc) is stored the number of items in each of the types defined by the user. The type of the answer is attached to the answer by means of a single character flag which is a number from 0 to 9. Note the use of the CHR$ function to achieve this — using STR$ would mean having to deal with the space that this function tags onto the front of numbers.

Lines 2230–2270: When the user quits the module, the second half of the array D is updated. This holds the start positions of each type group. This is arrived at by simply successively adding the number of items in each type group to the variable SUM.

Testing Module 4.1.5

Insertion of a temporary line 2500 RETURN should enable you to input items to the program under your specified headings, though these will not be stored anywhere.

MODULE 4.1.6

```
2500 REM*************************
2510 REM BINARY SEARCH
2520 REM*************************
```

```
2530 LET POWER=INT(LOG(ITEMS-1)/LOG(2))
2540 LET SEARCH=2^POWER
2550 FOR I=POWER-1 TO 0 STEP -1
2560 IF A$(SEARCH)<T1$ THEN LET SEARCH=S
EARCH+2^I
2570 IF A$(SEARCH)>T1$ THEN LET SEARCH=S
EARCH-2^I
2580 IF SEARCH<1 THEN LET SEARCH=1
2590 IF SEARCH>ITEMS-1 THEN LET SEARCH=I
TEMS-1
2600 NEXT I
2610 IF A$(SEARCH)<T1$ THEN LET SEARCH=S
EARCH+1
2620 FOR I=ITEMS TO INT(SEARCH)+1 STEP -
1:LET A$(I)=A$(I-1):LET B$(I)=B$(I-1):NE
XT I
2630 LET A$(SEARCH)=T1$:LET B$(SEARCH)=T
2$
2640 RETURN
2650 RETURN
```

A standard binary search module. Note that items are stored in alphabetical order of answers and that, since the type of the item is attached to the front of the answer as a single character, the items are in fact stored in order of type. If you do not enter any types, the items will be stored in straight alphabetical order of answer.

Testing Module 4.1.6

You should now be able to input answers which will be properly inserted into the main arrays, A$ and B$.

MODULE 4.1.7

```
3000 REM*************************
3010 REM USER SEARCH
3020 REM*************************
3030 LET F$="SEARCH":LET P1=1:GOSUB 700
3040 PRINT @ 15*32,"TOTAL ITEMS ";ITEMS
2
3050 PRINT @ 3*32," >'ENTER' FOR NEXT I
EM"
3060 PRINT " >POSITIVE OR NEGATIVE NUMB
R TO  MOVE POINTER"
3070 PRINT " >'DDD' TO DELETE ITEM"
3080 PRINT " >'ZZZ' TO QUIT FUNCTION"
3090 PRINT STRING$(32,CHR$(156))
3100 LET SEARCH=1
3110 PRINT @ 9*32,"ENTRY NO:-";SEARCH
3120 PRINT MID$(A$(SEARCH),2)
3130 PRINT B$(SEARCH)
3140 LET TEMP=VAL(LEFT$(A$(SEARCH),1))
3150 IF LEFT$(A$(SEARCH),1)<>" " THEN PR
INT D$(TEMP)
3160 INPUT "WHICH DO YOU REQUIRE";S$
3170 IF S$="DDD" THEN LET D(0,TEMP)=D(0
TEMP)-1:FOR I=SEARCH TO ITEMS-2:LET A$(
)=A$(I+1):LET B$(I)=B$(I+1):NEXT I:LET
TEMS=ITEMS-1:GOSUB 2230:RETURN
3180 IF S$="ZZZ" THEN RETURN
3190 IF S$<>"" THEN GOTO 3230
3200 LET SEARCH=SEARCH+1
3210 IF SEARCH=ITEMS-1 THEN RETURN
3220 GOTO3110
3230 LET SEARCH=SEARCH+VAL(S$)
3240 IF SEARCH>ITEMS-2 THEN LET SEARCH=I
TEMS-2
3250 IF SEARCH<1 THEN LET SEARCH=1
3260 GOTO 3110
```

A simple search on the lines of previous programs, but with the added facility that the user is able to specify a forward or backward leap through the file.

Testing Module 4.1.7

You should now be able to page through the items you enter, jumping backwards or forwards in the file and to delete items.

MODULE 4.1.8

```
3500 REM****************************
3510 REM RANDOM QUESTIONS
3520 REM****************************
3530 LET QUESTION=0
3540 LET F$="QUESTIONS":LET P1=1:GOSUB 7
000
3550 PRINT:INPUT "DO YOU WISH POSSIBLE A
NSWERS TO BE DRAWN ONLY FROM THE SAME
   ANSWER TYPE (Y/N) ";Q$:CLS
3560 IF Q$="Y" THEN LET QUESTION=1
3570 LET Q1=RND(ITEMS-2)
3580 LET Q2=RND(5)-1
3590 LET Q(Q2)=Q1
3600 IF QUESTION=0 OR D(0,VAL(LEFT$(A$(Q
1),1)))<5 THEN LET START=0:LET NUMBER=IT
EMS-2:ELSE LET START=D(1,VAL(LEFT$(A$(Q1
),1)))-1:LET NUMBER=D(0,VAL(LEFT$(A$(Q1)
,1)))
3610 FOR I=0 TO 4
3620 IF I=Q2 THEN GOTO3690
3630 LET PLACE=START+RND(NUMBER)
3640 IF PLACE=Q(Q2) THEN GOTO 3630
3650 FOR J=0 TO I
3660 IF PLACE=Q(J) THEN GOTO 3630
3670 NEXT J
3680 LET Q(I)=PLACE
3690 NEXT I
3700 PRINT NAME$(1);":"
3710 PRINT B$(Q(Q2))
3720 PRINT STRING$(32,CHR$(181));
3730 PRINT NAME$(0);":"
3740 FOR I=1 TO 5
3750 PRINT I;") ";MID$(A$(Q(I-1)),2)
3760 NEXT I
3770 PRINT "WHICH DO YOU THINK IS THE RI
GHT ANSWER?":INPUT"TYPE IN THE NUMBER:";
ANSWER
3780 LET QTOTAL=QTOTAL+1
3790 IF ANSWER<>Q2+1 THEN PRINT "INCORRE
CT. THE RIGHT ANSWER WAS:";Q2+1;") ";MID
$(A$(Q(Q2)),2):GOTO 3820
3800 PRINT "correct.":PLAY "T504L3C;P20:L
2C;L4O3A£;A;L2A;L4G£;A;L2A;P10;L4G£;A;L2
A;L4G£:A;O4L2C;O3L4A;O4C;O3L2A£"
3810 LET RIGHT=RIGHT+1
3820 INPUT "'ENTER FOR NEW QUESTION OR
      'ZZZ' TO QUIT FUNCTION.";Q$:CLS:IF Q
$="ZZZ" THEN RETURN ELSE GOTO 3570
```

This module is the core of the program. Its function is to generate the random tests according to instructions laid down by the user.

Commentary

Line 3550: Tests can take two forms. Potential answers can be drawn from the whole file of possible answers, in which case the test is likely to be fairly easy, for the simple reason that a fair number of absurd answers may be generated if the questions and answers cover a wide range. Answers may, however, be drawn only from the same type if the user so specifies. In this case, answers are likely to be more similar and the tests accordingly more difficult.

Lines 3570–3590: These three lines generate a random number which is the address of an item in the file, then a random place for it in the array Q. Note that RND(5)–1 is not quite as silly as it sounds — it is not equal to RND(4), since RND(4) can never equal zero.

Line 3600: This line determines whether the user has asked for the harder type of test and whether there are in fact five items in the group from which the first random question has been chosen. If both conditions are met then the variable START is set to the first item in the group and the variable NUMBER is set to equal the number of items within the group. If the user has not specified the harder test, or if there are not five items in the group, then START is set to the beginning of the file and NUMBER to the total number of items within the file.

Lines 3610–3690: The rest of the array Q is filled with the addresses of answers randomly chosen from the area of the file indicated by START and NUMBER, with checks to see that answers are not duplicated.

Lines 3700–3770: The question and the five possible answers are printed on the screen, with a prompt to input the number of the correct answer.

Lines 3790–3800: Depending on whether the right answer is given, the user is either simply informed of the right answer or is rewarded with a cheery tune to indicate success.

Testing Module 4.1.8

If you have previously saved some data, you should now be in a position to generate some tests, either hard or easy.

MODULE 4.1.9

```
4000 REM**************************
4010 REM SCORE
4020 REM***************************3410 LET
 F$="SCORE":LET P1=1:GOSUB 7000
4030 PRINT:PRINT "TOTAL QUESTIONS:";QTOT
AL
4040 PRINT "CORRECT ANSWERS:";RIGHT
4050 PRINT: PRINT "SCORE:";INT(((RIGHT-Q
TOTAL/5)/(QTOTAL*.8))*100);"%"
4060 PRINT:INPUT "DO YOU WISH TO ZERO SC
ORE (Y/N)";Q$:IF Q$<>"Y" THEN RETURN EL
SE LET QTOTAL=0:LET RIGHT=0:RETURN
```

During the course of the previous module the variables QTOTAL and RIGHT were updated for each question and for each right answer respectively. They are now used to make an assessment of the user's performance, with allowance made for the 20% correct answers that could be obtained by simply pressing the same button each time.

Testing Module 4.1.9

You should now be able to obtain an assessment of your performance in a test and to reset your score if you wish. If this module functions correctly then the program is ready for use.

Summary

This is actually quite a powerful program, but remember that you will only confirm that for yourself by entering enough data to make it enjoyable. The program is also a reminder that wherever possible, if you are going to write a complex program, you may as well go a little further and make it a general purpose one, thus saving yourself a great deal of work in the future.

Going further

1) As presently constituted, the program checks to see that the same answer is not displayed twice for a question, but not that two answers from different positions in the file are actually identical. Could you insert a check into Module 7 to ensure that identical answers are not printed?
2) The question of rewards for success is an interesting one — adults seem to find success its own reward when playing with, I mean using, this program. For children, however, all manner of rewards are possible. What about tagging a short game onto the program which would be accessed for three minutes every time 10 right answers had been supplied.

4.2 WORDS

Once you have a program that works well, you soon find that it suggests other uses to you. Such was the case with MultiQ and the result was this simple aid to learning to read which, with the help of an adult, can be fun and a step forward for kids in the earliest stages of reading. The only real difference between this program and MultiQ is that the questions take the form of simple pictures and the answers are possible words to go with the pictures.

The pictures are no more than the output of another program we have already discussed, Artist, picked up from tape and loaded into this program's dictionary. The capacity of the program as presented here is 100 words, though another set could be picked up from tape if so desired.

Designs meant to be used by this program need to use only the bottom 10 lines of the screen, since the top six are used to set the questions.

MODULE 4.2.1

```
6000 REM*****************************
6010 REM DATA FILES
6020 REM****************************
6030 MOTOR ON AUDIO ON:INPUT "POSITION T
APE THEN PRESS enter  (MOTOR IS ON):";Q$
:MOTOR OFF
```

```
6040 PRINT:INPUT "PLACE RECORDER IN CORR
ECT MODE  THEN PRESS enter";Q$
6050 PRINT:PRINT "FUNCTIONS AVAILABLE:",
"1)SAVE DATA",,"2)LOAD DATA":INPUT "WHIC
H DO YOU REQUIRE:";Q:ON Q GOTO 6070,6180
6060 RETURN
6070 MOTOR ON:FOR I=1 TO 10000:NEXT I
6080 OPEN "O",£-1,"WORDS"
6090 PRINT £-1,ITEMS
6100 FOR I=0 TO ITEMS-1
6110 PRINT £-1,A$(I,2),A$(I,1),LEN (A$(I
,1))
6120 FOR J=10 TO LEN(A$(I,1))
6130 PRINT £-1,ASC(MID$(A$(I,1),J))
6140 NEXT J
6150 NEXT I
6160 CLOSE£-1
6170 RETURN
6180 PCLEAR1:CLEAR 20000:LET FLAG=1:GOTO
 1540
6190 OPEN "I",£-1,"WORDS"
6200 INPUT £-1,ITEMS
6210 FOR I=0 TO ITEMS-1
6220 INPUT £-1,A$(I,2),A$(I,1),NN:IF LEN
(A$(I,1))=8 THEN LET A$(I,1)=A$(I,1)+" "
6230 FOR J=10 TO NN
6240 INPUT £-1,C:LET A$(I,1)=A$(I,1)+CHR
$(C)
6250 NEXT J,I
6260 CLOSE£-1
6270 GOTO 1000
```

A standard data-file module.

MODULE 4.2.2

```
7000 REM*************************
7010 REM FORMAT TITLES
7020 REM*************************
7030 LET P2=14-INT(LEN(F$)/2)
7040 PRINT @ 32*P1+P2,STRING$(LEN(F$)+2,
185)
7050 PRINT @ 32*(P1+1)+P2,CHR$(185)+F$+C
HR$(185)
7060 PRINT @ 32*(P1+2)+P2,STRING$(LEN(F$
)+2,185)
7070 RETURN
```

A standard title-formatting module.

MODULE 4.2.3

```
1000 REM*************************
1010 REM MENU
1020 REM*************************
1030 CLS:LET F$="WORDS":LET P1=1:GOSUB 7
000
1040 PRINT: PRINT "COMMANDS AVAILABLE:"
1050 PRINT "    1)INPUT NEW ITEMS"
1060 PRINT "    2)SEARCH/DELETE"
1070 PRINT "    3)GENERATE QUESTIONS"
1080 PRINT "    4)DISPLAY OR RESET SCORE"
1090 PRINT "    5)DATA FILES"
1100 PRINT "    6)INITIALISE"
1110 PRINT "    7)STOP"
1120 PRINT:INPUT "WHICH DO YOU REQUIRE:"
;Z:CLS
1130 IF Z<6 THEN ON Z GOSUB 2000,2500,35
00,4000,6000:GOTO 1000
1140 ON Z-5 GOTO 1500,1160
1150 GOTO 1000
1160 LET F$="WORDS":LET P1=6:GOSUB 7000
1170 STOP
```

A standard menu module.

MODULE 4.2.4

```
1500 REM**************************
1510 REM INITIALISE
1520 REM**************************
1530 PCLEAR 1:CLEAR 20000:LET FLAG=0
1540 DIM Q(4)
1550 LET LAST=1
1560 DIM A$(100,2)
1570 IF FLAG=0 THEN GOTO 1000 ELSE GOTO
6190
```

This module initialises the program variables, including the main array A$. You may note in relation to the use of this array that I have committed the cardinal crime of ignoring the zero element.

MODULE 4.2.5

```
3000 REM**************************
3010 REM PRINT DESIGN
3020 REM**************************
3030 LET Y=VAL(LEFT$(DESIGN$,3))
3040 LET X=VAL(MID$(DESIGN$,4,3))
3050 LET Z=VAL(MID$(DESIGN$,7,3))
3060 FOR I=Y TO Y+(LEN(DESIGN$)-9)/Z-1
3070 FOR J=X TO X+Z-1
3080 POKE(1024+32*I+J),ASC(MID$(DESIGN$,
10+(I-Y)*Z+J-X,1))
3090 NEXT J,I
3100 RETURN
```

If you remember the Artist program then you will also remember the function of this module, since it is the same as the program section in the earlier program which reassures the user that the correct design has been stored in a string. For commentary refer to Artist.

MODULE 4.2.6

```
2000 REM**************************
2010 REM INPUT OF NEW ITEMS
2020 REM**************************
2030 CLS:LET F$="NEW ITEMS":LET P1=0:GOS
UB 7000
2040 MOTOR ON: AUDIO ON:INPUT "POSITION
TAPE THEN PRESS enter  (MOTOR IS ON):";Q
$
2050 MOTOR OFF:INPUT "PUT RECORDER INTO
PLAY MODE THENPRESS enter";Q$
2060 OPEN "I",£-1,"DESIGN"
2070 INPUT £-1,DESIGN$:IF LEN(DESIGN$)=8
 THEN LET DESIGN$=DESIGN$+" "
2080 IF EOF(-1) THEN GOTO 2120
2090 INPUT £-1,N
2100 LET DESIGN$=DESIGN$+CHR$(N)
2110 GOTO 2080
2120 CLOSE£-1
2130 CLS0
2140 GOSUB 3000
2150 PRINT @ 0,"";INPUT "DO YOU WANT TH
IS (Y/N):";Q$:IF Q$<>"Y" THEN RETURN
2160 INPUT "WORD TO GO WITH THIS PICTURE
 ";W$
2170 INPUT "IS THIS CORRECT (Y/N):";Q$:I
F Q$<>"Y" THEN GOTO 2160
2180 LET A$(ITEMS,1)=DESIGN$:LET A$(ITEM
S,2)=W$:LET ITEMS=ITEMS+1
2190 INPUT "ANOTHER PICTURE (Y/N):";Q$:I
F Q$<>"Y" THEN RETURN
2200 GOTO 2030
```

The purpose of this module is to load individual designs created by the Artist program and to allow them to be labelled with a word and then stored in the main array. Note that there is no sort —items are inserted one after another.

Commentary

Lines 2080–2120: You may note something new here, the use of IF EOF(–1). All this means is that instead of reading a variable from tape which tells the programs how many characters are to be read and added to DESIGN$, we simply go on reading until the End of File marker is found, then stop.

Testing Module 4.2.6

We have come a long way into the program without testing anything but most of the material has been familiar, so there should be no major problems. You should now be in a position, having initialised the program, to pick up designs from tape which were created earlier using Artist and to supply a name to go with them, though you cannot yet display the file into which they are placed except in direct mode.

MODULE 4.2.7

```
2500 REM***************************
2510 REM USER SEARCH
2520 REM***************************
2530 IF ITEMS=0 THEN RETURN
2540 LET S=0
2550 CLS0:LET DESIGN$=A$(S,1):GOSUB 3000
:PRINT @ 15*32,A$(S,2);
2560 PRINT @ 0,">>enter FOR NEXT ITEM"
2570 PRINT ">>POS. OR NEG. NUMBER TO MOV
E."
2580 PRINT ">>'DDD' TO DELETE ITEM."
2590 PRINT ">>'ZZZ' TO QUIT FUNCTION."
2600 INPUT Q$
2610 IF Q$="DDD" THEN FOR I=S TO ITEMS-1
:LET A$(I,1)=A$(I+1,1):LET A$(I,2)=A$(I+
1,2):NEXT:LET ITEMS=ITEMS-1
2620 IF Q$="ZZZ" THEN RETURN
2630 IF Q$="" THEN LET S=S+1
2640 LET S=S+VAL(Q$):IF S>ITEMS-1 THEN L
ET S=ITEMS-1
2650 IF S<0 THEN LET S=0
2660 GOTO 2550
```

A simple user search module.

Testing Module 4.2.7

You should now be able to page through any items you load and delete at will.

MODULE 4.2.8

```
3500 REM***************************
3510 REM RANDOM QUESTIONS
3520 REM***************************
3530 LET Q1=RND(ITEMS)-1
3540 LET Q2=RND(5)-1
```

```
3550 LET Q(Q2)=Q1
3560 FOR I=0 TO 4
3570 IF I=Q2 THEN GOTO3640
3580 LET PLACE=RND(ITEMS)-1
3590 IF PLACE=Q(Q2) THEN GOTO 3580
3600 FOR J=0 TO I
3610 IF PLACE=Q(J) THEN GOTO 3580
3620 NEXT J
3630 LET Q(I)=PLACE
3640 NEXT I
3650 LET DESIGN$=A$(Q1,1):CLS0:GOSUB 300
0
3660 FOR I=0 TO 4:PRINT @ I*32,A$(Q(I),2
);:NEXT
3670 PRINT @ 5*32,"";:INPUT "WHICH ONE (
Z=STOP):";C$:IF C$="Z" THEN CLS:FOR I=0
TO 479:PRINT @ I, CHR$(RND(128)+127):NEX
T:PRINT @ 8*32+10,"goodbye";:FOR I=1 TO
5000:NEXT:CLS:STOP
3680 LET QSUM=QSUM+1
3690 IF C$<>A$(Q1,2) THEN CLS0:PRINT @ 8
,"WRONG":FOR I=1 TO 1000:NEXT:GOTO 3500
3700 PRINT "correct":PLAY "T5O4L3C;P20;L
2C;L4O3A£;A;L2A;L4G£;A;L2A;P10;L4G£;A;L2
A;L4G£;A;O4L2C;O3L4A;O4C;O3L2A£"
3710 LET RIGHT=RIGHT+1
3720 GOTO 3500
```

Equivalent to the random question module in MultiQ, except that there is no provision for the harder type of test, so the module is simpler. The module also says a rainbow goodbye when the user quits — it does not return to the main menu so that there is a smaller chance of someone inadvertently wiping out the data.

Testing Module 4.2.8

The program should now choose a random design and print it at the bottom of the screen, then print five words to choose from, one of which must be input.

MODULE 4.2.9

```
4000 REM************************
4010 REM SCORE
4020 REM************************
4030 IF QSUM=0 THEN RETURN
4040 LET F$="SCORE":LET P1=1:GOSUB 7000
4050 PRINT:PRINT "TOTAL QUESTIONS:";QSUM
4060 PRINT "CORRECT ANSWERS:";RIGHT
4070 PRINT: PRINT "SCORE:";INT(((RIGHT-Q
SUM/5)/(QSUM*.8))*100);"%"
4080 PRINT:INPUT "DO YOU WISH TO ZERO SC
ORE (Y/N):";Q$: IF Q$<>"Y" THEN RETURN E
LSE LET QSUM=0:LET RIGHT=0:RETURN
```

The score-keeping module, as in the previous program.

Testing Module 4.2.9

You should now be able to receive a score for any tests undertaken.

Summary

This again is a program which requires some work if it is to be of any use since the small designs it uses do take some time to create. In creating the designs it is especially important to prepare them properly in advance before sitting down with the Artist program. Use of squared paper can save a lot of frustration when it comes to actually creating the designs before they are entered.

If you don't have any children the right age, then why not design some crude representations of electrical symbols, and have the program set some tests about them?

Going further

1) Your children may be more familiar with lower-case letters. Could you alter the program so that the output is in lower-case?
2) The question of rewards rears its head even more pronouncedly in relation to this program. Try to think how winning could be made a bit more of a thrill.

4.3 WHERE?

This is an uncomplicated program which quite effectively tests your knowledge of geography, or at least of the location of cities in a range of countries. The program makes use of the second format of design output by the Artist program, the map save.

MODULE 4.3.1

```
7000 REM***************************
7010 REM FORMAT TITLES
7020 REM***************************
7030 LET P2=14-INT(LEN(F$)/2)
7040 PRINT @ 32*P1+P2,STRING$(LEN(F$)+2,
185)
7050 PRINT @ 32*(P1+1)+P2,CHR$(185)+F$+C
HR$(185)
7060 PRINT @ 32*(P1+2)+P2,STRING$(LEN(F$
)+2,185)
7070 RETURN
```

A standard title formatting module.

MODULE 4.3.2

```
6000 REM***************************
6010 REM DATA FILES
6020 REM***************************
6030 MOTOR ON:AUDIO ON:INPUT "POSITION T
APE THEN PRESS enter  (MOTOR IS ON):";Q$
:MOTOR OFF
6040 PRINT:INPUT "PLACE RECORDER IN CORR
ECT MODE  THEN PRESS enter";Q$
6050 PRINT:PRINT "FUNCTIONS AVAILABLE:",
"1)SAVE DATA",,"2)LOAD DATA":INPUT "WHIC
H DO YOU REQUIRE ";Q:ON Q GOTO 6070,6150
6060 RETURN
6070 MOTOR ON:FOR I=1 TO 10000:NEXT I
6080 OPEN "O",£-1,"WHERE"
6090 PRINT £-1,CTOTAL
6100 FOR I=0 TO CTOTAL-1:FOR J=0 TO 13
6110 FOR K=1 TO 32
6120 PRINT £-1,ASC(MID$(B$(I,J),K,1))
6130 NEXT K,J,I
6140 CLOSE£-1:RETURN
6150 PCLEAR1:CLEAR 18000:LET FLAG=1:GOTO
 1540
6160 OPEN "I",£-1,"WHERE"
6170 INPUT £-1,CTOTAL
```

```
6180 FOR I=0 TO CTOTAL-1:FOR J=0 TO 13
6190 LET B$(I,J)="":FOR K=1 TO 32
6200 IF EOF(-1) THEN GOTO 6230
6210 INPUT £-1,C:LET B$(I,J)=B$(I,J)+CHR
$(C)
6220 NEXT K,J,I
6230 CLOSE£-1:GOTO 1000
```

A standard data-file module.

MODULE 4.3.3

```
1000 REM*************************
1010 REM MENU
1020 REM*************************
1030 CLS:LET F$="WHERE":LET P1=1:GOSUB 7
000
1040 PRINT:PRINT "COMMANDS AVAILABLE :"
1050 PRINT "    1)LOAD NEW COUNTRY"
1060 PRINT "    2)RECORD NEW CITIES"
1070 PRINT "    3)SET QUESTIONS"
1080 PRINT "    4)DATA FILES"
1090 PRINT "    5)INITIALISE"
1100 PRINT "    6)STOP"
1110 PRINT:INPUT "WHICH DO YOU REQUIRE:"
;Z:CLS
1120 ON Z GOSUB 2000,2500,3000,6000,1500
,1150
1140 GOTO 1000
1150 CLS:LET F$="WORDS":LET P1=1:GOSUB 7
000
1160 LET F$="PROGRAM TERMINATED":LET P1=
10:GOSUB 7000
1170 END
```

A standard menu module.

MODULE 4.3.4

```
1500 REM*************************
1510 REM INITIALISE
1520 REM*************************
1530 PCLEAR 1:CLEAR 22000:LET FLAG=0
1540 DIM B$(20,13)
1550 IF FLAG=1 THEN GOTO 6160 ELSE GOTO
1000
```

The country maps generated by Artist will be stored in the array B$, each of its 21 elements holding 14 lines of an individual map.

MODULE 4.3.5

```
2000 REM*************************
2010 REM LOAD NEW COUNTRY
2020 REM*************************
2030 LET F$="NEW COUNTRIES":LET P1=1:GOS
UB 7000
2040 PRINT:INPUT "PLACE TO LOAD THIS COU
NTRY:";PLACE:LET PLACE=PLACE-1
2050 PRINT:MOTOR ON:AUDIO ON:INPUT "POSI
TION TAPE THEN PRESS enter (MOTOR IS ON
):";Q$
2060 MOTOR OFF:INPUT "PUT RECORDER INTO
PLAY MODE THENPRESS enter";Q$
2070 OPEN "I",£-1,"MAP"
2080 FOR I=0 TO 13:LET B$(PLACE,I)="":FO
R J=0 TO 31
2090 IF EOF(-1) THEN GOTO 2120
2100 INPUT £-1,CC:LET B$(PLACE,I)=B$(PLA
CE,I)+CHR$(CC)
2110 NEXT J,I
2120 CLOSE£-1
2130 CLS0:FOR I=0 TO 13:PRINT @ I*32,B$(
PLACE,I);:NEXT
```

```
2140 FOR I=1 TO 5000: NEXT
2150 PRINT @ 15*32,"";:INPUT "ACCEPTABLE
 (Y/N): ";Q$:IF Q$="Y" THEN LET CTOTAL=CT
OTAL+1
2160 RETURN
```

This module picks up the maps designed with the use of Artist and loads them into the array B$. Note that the user is asked to specify the position at which the map is to be loaded — there is no automatic mechanism for allocating it a place.

Testing Module 4.3.5

You should now be in a position to load into the program some maps created with the aid of Artist. The first of these should be loaded at one. You should also be prompted to supply a country name to correspond with the map.

MODULE 4.3.6

```
2500 REM***************************
2510 REM RECORD CITIES
2520 REM***************************
2530 INPUT "NUMBER OF COUNTRY:";COUNTRY:
LET COUNTRY=COUNTRY-1
2540 CLS0:FOR I=0 TO 13:PRINT B$(COUNTRY
,I);:NEXT:LET X=0:LET Y=0
2550 LET T$=INKEY$:IF T$<>"" THEN GOTO 2
600
2560 LET P=PEEK(1024+Y*32+X)
2570 POKE(1024+Y*32+X),106:FOR I=1 TO 25
:NEXT
2580 POKE(1024+Y*32+X),P:FOR I=1 TO 25:N
EXT
2590 GOTO 2550
2600 LET X=X-(T$=CHR$(9))+(T$=CHR$(8)):L
ET X=X+32*(X>31)-32*(X<0)
2610 LET Y=Y-(T$=CHR$(10))+(T$=CHR$(94))
:LET Y=Y+14*(Y>13)-14*(Y<0)
2620 PRINT @ 15*32,"X=";X;"    Y=";Y;"    "
;
2630 IF T$<>CHR$(13) THEN GOTO 2550
2640 CLS:PRINT "RECORD THE FOLLOWING DET
AILS IN A data STATEMENTIN THE SECTION
BEGINNING AT LINE ";10000+(COUNTRY+1)*10
0;":"
2650 PRINT "CITY NAME,X CO-ORDINATE,Y CO
-ORDINATE"
2660 PRINT:PRINT "THE NUMBER IN THE DATA
 STATMENT AT ";10000+(COUNTRY+1)*100;" S
HOULD BE INCREASED  BY 1."
2670 PRINT "THE SECOND ITEM IN THAT LINE
     SHOULD BE THE COUNTRY NAME E.G.:";
2680 PRINT "5,FRANCE"
2690 PRINT "THIS WOULD MEAN THAT 5 CITIE
S    HAD BEEN RECORDED FOR FRANCE."
2700 PRINT:PRINT "X=";X;"///Y=";Y;:INPUT
"(enter TO CONT.)";Q$
2710 RETURN
```

The purpose of this module is to move a flashing cursor around any of the maps stored by the program and to display the co-ordinates of the cursor. On the input of ENTER, the user is given instructions as to the manner in which details of countries and cities are to be recorded in DATA statements. Note that the best use is made of this module by first ensuring that the maps currently available are all loaded into the program and saved using the data-file module.

Then use this module to record the co-ordinates of any cities you wish to enter, keeping the co-ordinates and names on a piece of paper until you have completed them all. Entering the DATA statements referred to every time you have established the co-ordinates of a new city will mean that you will lose all the program data and it will have to be loaded from tape each time.

The DATA about countries is recorded in a section at the end of the program which, for the sake of clarity you are instructed to begin at 10000. Each country has a space of 100 lines, beginning at 10100 for the country in file space zero, 10200 for the country in file space one and so on. A typical entry for a country would look like this:

10200 DATA 5,NEVERLAND
10202 DATA CITY1,12,8
10204 DATA CITY2,5,7
10206 DATA CITY3,6,10
10208 DATA CITY4,11,3
10210 DATA CITY5,15,27

What this means is that the country is called Neverland, that it has five cities and that their names and X,Y co-ordinates (as determined by using this module) are as shown in the next five lines.

By employing DATA statements we do away with the need for modules to insert and delete data or to page through it— the work is done by the user. DATA statements can be abused, and often are, but where data is not expected to have to be entered frequently, they can be a time saver in terms of programming and a space saver in terms of the program functions that can be dispensed with.

Testing Module 4.3.6

You should now be able to specify a map that is contained within the file and move a flashing cursor around it. Pressing ENTER should lead to instructions being displayed about the necessary DATA statements.

MODULE 4.3.7

```
3000 REM************************
3010 REM GENERATE QUESTIONS
3020 REM************************
3030 LET COUNTRY=RND(CTOTAL)-1
3040 GOSUB 3170
3050 CLS0:FOR I=0 TO 13:PRINT B$(COUNTRY
,I);:NEXT
3060 POKE(1024+32*Y+X),106
3070 FOR I=1 TO 3
3080 PRINT @ 14*32,STRING$(63,CHR$(128))

3090 LET QTOTAL=QTOTAL+1:PRINT @ 14*32,"
";:INPUT "NAME OF CITY:";Q$:IF Q$=M$ THE
N GOTO 3130
3100 LET WRONG=WRONG+1:PRINT @ 15*32,"WR
ONG";:FOR J=1 TO 1000:NEXT
3110 NEXT I
3120 PRINT @ 15*32,"CITY WAS ";M$;:INPUT
 "   <enter>";Q$:GOTO 3140
```

```
3130 PRINT @ 0,M$;":CORRECT";:FOR I=1 TO
 1000:NEXT
3140 PRINT @ 15*32,"%=";(QTOTAL-WRONG)/Q
TOTAL*100;"  "
3150 INPUT "ANOTHER GO (Y/N):";Q$:IF Q$<
>"Y" THEN RETURN
3160 INPUT "SAME COUNTRY (Y/N):";Q$:IF Q
$<>"Y" THEN GOTO 3030 ELSE GOTO 3040
3170 RESTORE:FOR I=0 TO COUNTRY:READ N,N
$:IF I<COUNTRY THEN FOR J=1 TO N:READ N$
,N,N:NEXT J,I
3180 FOR I=1 TO RND(N):READ M$,X,Y:NEXT:
RETURN
```

This module selects a random country and, within that country a random city. A marker is placed at this point and the user is requested to supply the appropriate name.

Commentary

Lines 3170–3180: Note that to extract data from the middle of a section of DATA statements it is necessary to READ the data from the beginning. In this case the two loops in line 3170 read all the country and city data up to the country specified in randomly generated number COUNTRY and then, on the basis of the number of cities specified for that country, line 3180 reads a random number of cities to arrive at the chosen city.

Testing Module 4.3.7

The program should now generate questions, allowing three attempts before supplying the answer. The user should be given the opportunity to specify whether the next question should be drawn from the same country — if not, the random function may well pick up the same country again anyway — this is not an error.

Summary

This program raises the interesting question of how far it is desirable to go in building all the necessary functions into the program rather than allowing the user to do some of the work. Would the program have been better with extra modules to cope with the addition, display and deletion of data for city names and locations? Well in many ways it would have been better, but would the improvement have been worth the extra time and the loss of enough memory for two country maps? If you think so, you have enough examples to work on to insert your own extra modules.

CHAPTER 5
High resolution text

Having examined some of the Dragon's very real capabilities in the fields of both high and low resolution graphics, we turn our attention to an area where the machine's performance is somewhat lacking compared to some other popular micro-computers — the mixing of text (that is letters and numbers) and high resolution graphics on the screen at the same time.

Many of you may be aware that one solution to this irritating limitation is to use the flexible DRAW command to literally draw letters on the screen in the high resolution PMODEs. The real disadvantage of this method is the necessity to go through the painfully slow process of building up the fairly complex strings that will be drawn and writing them into each new program which requires some text. In the two programs which follow we shall attempt to overcome this drawback by providing a simple method of creating the desired characters, of storing them for subsequent use and of compiling them into character sets for subsequent use by other programs. In other words we shall attempt to substantially extend the Dragon's capabilities.

5.1. CHARACTERS

The purpose of this program is to allow you to build up any character you wish which is capable of being fitted into an area on the screen of 32*32 pixels. The actual size of the character when printed on the screen will depend upon the PMODE and the scale in use when it is DRAWn.

MODULE 5.1.1

```
1000 REM***************************
1010 REM INITIALISE
1020 REM***************************
1030 PMODE0,1:PCLEAR 2:CLEAR 1000:PMODE
1,1:SCREEN 1,0:PCLS4
1040 DIM A(31,31):DIM B(31,31)
1050 DIM C(127,7)
1060 DRAW "BM0,0"
1070 FOR I=1 TO 2
1080 FOR J=1 TO 16:DRAW "C1;R3;BR1;C2;R3
;BR1":NEXT J
1090 DRAW "BM-128,+2":NEXT I
1100 FOR I=1 TO 2:FOR J=1 TO 16:DRAW "C2
;R3;BR1;C1;R3;BR1":NEXT J
1110 DRAW "BM-128,+2":NEXT I
1120 GET (0,0)-(127,7),C
1130 PCLS4
```

The purpose of this module is to initialise the program variables and to set up an array which will be used later in the program to reduce the time taken to print a 32*32 chequerboard design by use of GET and PUT.

Commentary

Line 1030: Since we shall be working with strings we shall need to set aside more than the basic minimum of string space. The remaining commands merely set aside sufficient memory space to work in PMODE 1 using the first colour set.

Lines 1060–1110: These lines initialise the DRAWing position to the top left hand corner of the screen and then DRAW the first two lines of a chequerboard, one square at a time. You will note once again how a series of DRAW commands placed on different lines are executed as if they were part of the same string.

Line 1120: The area of the screen DRAWn upon is 128*8 pixels and this rectangle is now stored in the array C using the GET command. It would not be possible to store the whole 32*32 matrix in such an array since even to store only 1/16th of it requires over 5,000 bytes of memory.

The heavy memory demand involved in the use of GET is the main drawback to an otherwise useful feature of the Dragon.

Testing Module 5.1.1

The functions of the various arrays can only be checked later in the process of entering the program but at this stage the module should visibly draw the first two lines of a chequerboard on the screen and then clear the screen.

MODULE 5.1.2

```
5000 REM****************************
5010 REM FUNCTIONAL SUBROUTINES
5020 REM****************************
5030 LET D$="BM"+STR$(X)+","+STR$(Y)+";"
+";D3;R1;U3;R1;D3;R1;U3":RETURN
```

The sole purpose of this module is to define a short string which draws an inked in square at an appropriate position in the array as defined by the variables X and Y.

Commentary

Line 5030: This line serves as a useful reminder that the strings used to control the DRAW command do not have to be cut and dried before running the program. All the string handling capabilities of the Dragon can be brought to bear. In this case, values for X and Y are inserted into the string using the STR$ function. The line is included as a separate one-line subroutine simply because it is called more than once in the program and it saves space if it is not spelt out in several places.

Testing Module 5.1.2

The line can be tested after the entry of the next module.

MODULE 5.1.3

```
2500 REM************************
2510 REM DRAW GRID
2520 REM************************
2530 PCLS2:FOR I=0 TO 120 STEP 8:PUT (0,
I)-(127,I+7),C:NEXT I
2540 DRAW "C1;BM128,0;D128;L128"
2550 FOR Y=0 TO 124 STEP 4:FOR X=0 TO 12
4 STEP 4
2560 IF A(Y/4,X/4)<>0 THEN GOSUB 5030:DR
AW "C0;"+D$
2570 NEXT X,Y:LET X=0:LET Y=0:RETURN
```

This module places on the screen the whole 32*32 grid that will be used to define characters. When later modules have been entered it will also ink in the squares which define a character.

Commentary

Line 2530: Using the array C, which holds two lines of the chequerboard design, this line prints the 32*32 grid by PUTting the contents of the array onto the screen in 16 consecutive locations. This is considerably faster than DRAWing the grid.

Lines 2550 – 2570: Using two loops to increment the values of X and Y, the array A is examined to see if the array element corresponding with each element in the grid contains something other than a zero. If it does, then Module 2 is called up and the current values of X and Y incorporated into D$, which then DRAWs an inked in square at the appropriate point.

Testing Module 5.1.3

The program should now be capable of placing the 32*32 element grid on the screen, then stopping with the RETURN without GOSUB error. If you wish, you can feed some ones into the array A in direct mode, then GOTO 2500. The corresponding squares on the grid should have been inked in. Note that it takes time to examine the whole array — some 20 seconds — so that a pause does not mean that the program is malfunctioning.

MODULE 5.1.4

```
1500 REM****************************
1510 REM CREATE DESIGN
1520 REM****************************
1530 GOSUB 2500
1540 LET X=0:LET Y=0
1550 LET T$=INKEY$:IF T$<>"" THEN GOTO 1
620
1560 GOSUB 5030
1570 DRAW "C0;"+D$
1580 FOR I=1 TO 25:NEXT
1590 DRAW "C"+STR$(1-((X+Y)/8<>INT((X+Y)
/8)))+D$
1600 FOR I=1 TO 25:NEXT
1610 GOTO 1550
1620 IF A(Y/4,X/4)=1 THEN DRAW "C0;"+D$
1630 LET X=X-4*(T$=CHR$(9))+4*(T$=CHR$(8
)):IF X>124 THEN LET X=124
1640 IF X<0 THEN LET X=0
1650 LET Y=Y-4*(T$=CHR$(10))+4*(T$=CHR$(
94)):IF Y>124 THEN LET Y=124
1660 IF Y<0 THEN LET Y=0
1670 IF T$="0" THEN DRAW "C"+STR$(1-((X+
Y)/8<>INT((X+Y)/8)))+D$:LET A(Y/4,X/4)=0
1680 IF T$="1" THEN DRAW "C0;"+D$:LET A(
Y/4,X/4)=1
1690 IF T$="R" THEN GOSUB 2030
1700 IF T$="M" THEN GOSUB 5030:DRAW "C4"
+D$:GOSUB 2070
1710 IF T$="I" THEN GOSUB 2200
1720 IF T$="E" THEN GOSUB 3000
1730 IF T$="S" THEN GOSUB 6000:SCREEN 1,
0
1740 GOTO 1550
```

This module is designed to allow the user to move a flashing cursor around the grid printed by the last module, inking in or erasing squares at will. Having satisfactorily designed a character, a variety of other program functions can be called up by the use of single-key codes. The cursor-moving techniques employed will be familiar from previous programs in this book.

Commentary

Lines 1550–1610: A variation of our standard flashing cursor routine. The cursor is first drawn and then redrawn to the background colour of the square it occupies. The whole process cries out for the use of GET and PUT but unfortunately the smallest rectangle which can be PUT back onto the screen in this PMODE is twice as long horizontally as one of our grid elements.

Line 1620: Having left the flashing cursor routine at the touch of a key, this line checks that the element which has just been redrawn to the background colour does not have to be inked in according to the information stored in the array A.

Lines 1630–1660: These lines, as will be recognised from previous programs, move the cursor around the screen. In this case the cursor moves in four pixel steps, anywhere within the limits of the grid. As usual, logical conditions are used to control the movement and the required input is one or other of the arrowed keys on the keyboard.

Line 1670: The 0 key is used to erase any inked-in element over which the cursor is currently flashing. This is done by simply redrawing it in the background colour. The relevant element in the array A must also be reset to zero, otherwise the square will be inked in again every time Module 3 is called up.

Line 1680: Pressing I inks in the square and sets the corresponding element in the array A.

Line 1690: Input of R rotates the whole grid 90 degrees anti-clockwise when the next module has been entered.

Line 1700: Input of M will later allow the design to be moved around in the grid.

Line 1710: Input of I transforms the design into its mirror-image.

Line 1720: Input of E extracts the string necessary to DRAW the character which has been created.

Line 1730: Input of S results in the string created being saved to tape. Note that because this will involve instructions being printed using the text screen, the SCREEN command must be used on return to retrieve the high resolution display.

Testing Module 5.1.4

At this point you should be able to move the flashing cursor around the grid, inking in or erasing squares at will. None of the other functions are yet available and their use will result in an error report undefined line.

MODULE 5.1.5

```
2000 REM**************************
2010 REM ARRAY MANIPULATIONS
2020 REM**************************
2030 FOR I=0 TO 31:FOR J=0 TO 31:LET B(J
,31-I)=A(I,J):NEXT J,I
2040 FOR I=0 TO 31:FOR J=0 TO 31:LET A(I
,J)=B(I,J):LET B(I,J)=0:NEXT J,I
2050 GOSUB 2500:RETURN
2060 REM**************************
2070 DRAW "C3;BM150,40:U12;F6;E6;D12":FO
R I=1 TO 100:NEXT
2080 LET MX=0:LET MY=0:LET X1=0:LET X2=0
:LET Y1=0:LET Y2=0
2090 LET T1$=INKEY$:IF T1$="" THEN GOTO
2090
2100 IF T1$="1" THEN LET MY=Y*-1:LET MX=
X*-1:LET X1=X:LET X2=124:LET Y1=Y:LET Y2
=124:DRAW "BM150,64;R5;BL3;U12;G1"
2110 IF T1$="2" THEN LET MY=Y*-1:LET MX=
124-X:LET Y1=Y:LET Y2=124:LET X1=0:LET X
2=X:DRAW "BM162,60;L12:U6;R12:U6:L12"
2120 IF T1$="3" THEN LET MY=124-Y:LET MX
=X*-1:LET X1=X:LET X2=124:LET Y1=0:LET Y
2=Y:DRAW "BM150,60:R12;U6:L8:R8;U6;L12"
2130 IF T1$="4" THEN LET MY=124-Y:LET MX
=124-X:LET X1=0:LET X2=X:LET Y1=0:LET Y2
=Y:DRAW "BM162,60:L6:U3:D6;U3:L6:U12"
```

```
2140 IF T1$<"1" OR T1$>"4" THEN FOR I=25
 TO 65:DRAW "C4;BM150,"+STR$(I)+";R12":N
EXT I:RETURN
2150 LET X1=X1/4:LET X2=X2/4:LET Y1=Y1/4
:LET Y2=Y2/4:LET MX=MX/4:LET MY=MY/4
2160 FOR I=Y1 TO Y2:FOR J=X1 TO X2:LET B
(I+MY,J+MX)=A(I,J):NEXT J,I
2170 FOR I=0 TO 31:FOR J=0 TO 31:LET A(I
,J)=B(I,J):LET B(I,J)=0:NEXT J,I
2180 GOSUB 2500:RETURN
2190 REM*************************
2200 FOR I=0 TO 31:FOR J=0 TO 31:LET B(I
,J)=A(I,31-J):NEXT J,I
2210 FOR I=0 TO 31:FOR J=0 TO 31:LET A(I
,J)=B(I,J):LET B(I,J)=0:NEXT J,I
2220 GOSUB 2500:RETURN
```

This module performs three of the functions called from the previous module, namely rotation, inversion and movement of the design within the grid. All the manipulations are performed by employing a second array, B, to which is transferred the data from the array A, suitably modified. The array B is then copied back into A.

Commentary

Lines 2030–2050: Examination of the subscripts for the arrays A and B in the first line will reveal that those three lines accomplish the rotation of the data stored in the array A by 90 degrees, that is to say that element 0,0 is moved to position 31,0 and so on. Having redefined the array A, Module 3 is recalled to draw the modified grid.

Lines 2060–2180: This subsection accomplishes the movement of the design within the grid. In order to understand this function it is first of all necessary to visualise the corners of the grid numbered in the following manner:

```
1  2
3  4
```

On calling up this section by the use of the M key in the previous module, the user is asked to specify a corner. If corner 4 is specified, then a rectangle is defined with two opposite corners consisting of grid corner 1 (the corner opposite to 4) and the current position of the cursor. This rectangle is then moved so that the corner defined by the cursor is relocated in grid corner 4. This may sound complex but a little experimentation will show that it is in fact a neat and simple means of moving the contents of the grid around. It is important to remember that if the design is to be moved down two lines, the bottom two lines of the design will be lost and similarly for moves in other directions.

Lines 2070: This line draws a large M next to the grid to show that the move function has been called — it seemed like a good idea at the time. The empty loop in this line serves the important function of separating the input named T$ in the previous module and one called T1$ which is about to be called for. Without this delaying loop there is a danger that if the user's finger lingers on the M key when calling up this function, the

INKEY$ function at line 2090 will define T1$ as M too. This delay is necessary whenever using a succession of INKEY$ inputs.

Line 2080: MX and MY are the variables which will be used to record the distance the defined rectangle must be moved. X1,Y1,X2 and Y2 will record the opposite corners of the defined rectangle.

Lines 2100–2130: These variables are set according to the corner specified as the destination of the move and the current position of the cursor. Again for no particular reason, the number of the corner chosen as a destination is drawn next to the grid.

Line 2140: If an erroneous input is made when the program is expecting a corner to be specified, the M is erased and control is returned to Module 4.

Lines 2150–2170: Having established the size of the rectangle to be moved and the amount of movement necessary, these values are divided by four so that they can be applied to the array A and the transformation accomplished in transferring the contents to the array B.

Testing Module 5.1.5

The three functions specified in the commentary should now be available.

MODULE 5.1.6

```
3000 REM***************************
3010 REM EXTRACT STRING
3020 REM***************************
3030 FOR I=0 TO 31:FOR J=0 TO 31:LET B(I
,J)=A(I,J):NEXT J,I
3040 LET DI$="HUELRGDF":LET E$=""
3050 LET X=0:LET Y=0:LET D1=0:LET D2=0:L
ET DIR=0
3060 FOR I=0 TO 31:FOR J=0 TO 31:IF B(I,
J)=0 THEN GOTO 3250
3070 LET E$=E$+"BM":IF J-X>0 THEN LET E$
=E$+"+" ELSE LET E$=E$+"-"
3080 LET E$=E$+MID$(STR$(ABS(J-X)),2)+",
"
3090 IF I-Y>0 THEN LET E$=E$+"+" ELSE LE
T E$=E$+"-"
3100 LET E$=E$+MID$(STR$(ABS(I-Y)),2)+";
R0;"
3110 LET X=J:LET Y=I
3120 LET B(Y,X)=0
3130 IF Y+D1>=0 AND Y+D1<=31 AND X+D2>=0
 AND X+D2<=31 THEN IF B(Y+D1,X+D2)<>0 TH
EN GOTO 3190
3140 FOR K=-1 TO 1:FOR L=-1 TO 1
3150 IF X+L>31 OR X+L<0 OR Y+K>31 OR Y+K
<0 THEN GOTO 3170
3160 IF B(Y+K,X+L)<>0 THEN LET D1=K:LET
D2=L:GOTO 3190
3170 NEXT L,K:IF DIR<>0 THEN LET E$=E$+M
ID$(DI$,DIR,1)+MID$(STR$(NN+1),2)+";"
3180 LET DIR=0:LET D1=0:LET D2=0:LET NN=
0:GOTO 3250
3190 LET T1=3*(D1+1)+D2+2:IF T1>4 THEN L
ET T1=T1-1
3200 IF T1=DIR THEN LET NN=NN+1
3210 IF T1<>DIR AND DIR<>0 THEN LET E$=E
$+MID$(DI$,DIR,1)+MID$(STR$(NN+1),2)+";"
:LET NN=0
3220 LET DIR=T1
3230 LET X=X+D2:LET Y=Y+D1
3240 GOTO 3120
3250 NEXT J,I
3260 IF NN<>0 AND DIR<>0 THEN LET E$=E$+
MID$(DI$,DIR,1)+MID$(STR$(NN+1),2)
3270 DRAW "S8;C3;BM150,60;"+E$
3280 IF INKEY$="" THEN GOTO 3280
3290 FOR I=0 TO 63:DRAW "C2;BM150,"+STR$
(60+I)+";R64":NEXT I
3300 LET X=0:LET Y=0:DRAW "S4":RETURN
```

Having established the functions necessary to define and manipulate a character on the grid, we come to the heart of the program, the module which takes the design which the user has created and transforms it into a string which, when DRAWn, will reproduce the desired character or design.

Commentary

Line 3030: Since elements in the design will be erased from the array as they are incorporated into the string, the process is actually carried out on a copy of the main array.

Line 3040: The letters contained in DI$ are the eight directions which can be handled by the DRAW command. E$ will contain the string defining the design or character.

Line 3050: X and Y are used to register co-ordinates on the grid. D1 and D2 are used to record the vertical and horizontal elements of the direction in which a line is currently being DRAWn.

Lines 3060 and 3250: The loop defined by these two lines scans through the grid, ignoring empty squares.

Lines 3070–3120: For reasons that will be seen later, the fact that program execution has arrived at this point shows that the square currently defined by I and J is inked in but that it does not follow on in a continuous line from any part of the design previously recorded in E$. The location of the square is therefore recorded in the form of a B(lank) M(ove) within the string. The first square to be recorded in this fashion will always be the top left hand square in the design and its position will be defined in relation to the top left hand corner. Other squares to be recorded in the BM format will be defined in relation to wherever DRAWing last left off. The drawing position is updated to the current square and the square is erased so that it cannot figure twice in the design.

Line 3130: If the element at Y + D1,X + D2 is not zero, then since D1 and D2 contain the direction in which a line is currently being drawn, the loop examining surrounding squares is jumped around.

Lines 3140–3170: If a current direction cannot be continued, this loop examines surrounding squares to see if there is any direction in which DRAWing may continue. If no such continuation is found then to E$ is added the direction and length of the line which has been traced in the design.

Lines 3190–3200: If it is possible to draw from the current square, the direction is checked to see if it is the direction of a line currently being drawn, if so the variable NN is incremented. If it is a new direction, the direction and length of the previously traced line are added to E$. The value attached to any particular direction is calculated by the formula at line 3190 and this value corresponds to the position of the relevant letter in DI$ (defined at line 3040). It may be worth noting in passing that this formula can come in useful in a variety of circumstances where a direction on a rectangular grid requires to be recorded. The values which the line will produce for the eight possible directions are as follows:

```
1  2  3
4  *  5
6  7  8
```

Compare this with the letters specified in DI$ and you will see why they are arranged as they are. The variables D1 and D2 are vertical and horizontal elements of the direction and range between − 1 and + 1.

Line 3260: This line simply ensures that any DRAWing left unfinished at the end of the loop is completed.

Line 3270: The design is now DRAWn next to the grid, using the new E$ which has been created. DRAWing it at scale 8 ensures that its proportions, though not its size, are the same as the design created on the grid.

Lines 3280–3300: The design is displayed until a key is pressed, then control is returned to Module 4. Note that the scale for DRAWing must be returned to the normal 4 before a RETURN is made, otherwise subsequent use of the DRAW command will produce oversize results.

Testing Module 5.1.6

Having defined a design on the grid, you should now be able to call up this module by pressing key E and, after a lengthy pause, see it displayed at $\frac{1}{2}$ scale. Stopping the program will allow you to examine the E$ which the module has created. Note that no check is made that your design is not too complex to be drawn by a string of up to 255 characters, so that too full a grid might result in an error, though this is unlikely to happen.

MODULE 5.1.7

```
6000 REM***************************
6010 REM SAVE CHARACTER TO TAPE
6020 REM***************************
6030 MOTOR ON:AUDIO ON:CLS:INPUT "POSITI
ON TAPE THEN PRESS enter  <MOTOR IS ON>:
":Q$
6040 MOTOR OFF:INPUT "START RECORDING TH
EN enter ":Q$
6050 MOTOR ON:FOR I=1 TO 10000:NEXT
6060 OPEN "O",£-1,"CHAR"
6070 PRINT £-1,E$
6080 CLOSE£-1
6090 RETURN
```

The function of this module is to allow the design which the user has created to be saved on tape in the form of a string. You will note that the module is more simple than many of the data file modules of earlier programs, this is because its sole purpose is to save a single string.

Testing Module 5.1.7

You should now be able to save E$ on tape. This can be verified by calling up this module, then stopping the program and clearing the variables. Insert at 8888 a single line instruction to open an input file by the name of CHAR and input E$, not forgetting to close the file. You may then print out E$ in direct mode, or DRAW it to check that it has been satisfactorily recorded and reclaimed. If this is successful then the program is complete and you are ready to proceed to the second half of the high resolution text section.

CHARACTERS: Summary of single key functions

With cursor flashing:

0	erases square on grid where cursor is situated.
1	inks in square on grid where cursor is situated.
R	rotates design within grid by 90 degrees anti-clockwise.
M	calls subroutine which moves design within grid.
I	transforms design within grid into its mirror image.
E	creates string which will duplicate design if DRAWn.
S	saves design on tape.

With large M drawn to the right of the grid:
1,2,3 or 4 specifies the corner towards which the design is to be moved.

Summary

This program is an odd one in that, as it stands, it is almost completely useless. That is to say, all it accomplishes is to store strings defining small scale designs or characters onto tape, hardly a stunning feat. In combination with other programs, however, which will pick up the characters created and compile them into usable character sets, and modules which will allow you to use such character sets easily in high resolution PMODEs, the program becomes an indispensable tool which enables the Dragon to exceed its normal capabilities.

Going Further

1) A character creator is hardly much use unless you are prepared to sit down and define some characters. Though this may seen an incredibly difficult and boring task at first glance, a moment's reflection will suffice to realise that a complete set of characters, already defined in pixels, is laid out before you in the listings given in this book. Alternative styles of

lettering can be found in the program listings in any computer magazine. With such examples to work from, you really should have no difficulty in building up a collection of worthwhile characters.

2) The program as given does not necessarily always make the best use of the 255 characters of string space available for E$. This is because a blank move always uses the BM notation, which requires at least seven characters (BM + 2, − 2) and possibly nine. An interesting challenge would be to insert a routine to test whether such a blank move could be covered by one of the single-letter DRAW instructions. The blank move given above for instance could just as easily have been defined by BE2, which would result in a considerable saving.

5.2 DICTIONARY

Having created characters it now only remains for them to be combined in such a way as to be useful for subsequent programs. The program which follows is designed to accomplish this by holding in memory up to 100 characters at one time, with the possibility of more being picked up from tape in batches of one hundred. The characters so stored can then be combined into collections such as ABCDEFGHI . . . etc to provide material for high resolution programs which require text. In later programs we shall examine practical modules for using such character sets without constantly having to specify DRAW commands in the program.

MODULE 5.2.1

```
1000 REM*************************
1010 REM MENU
1020 REM*************************
1030 CLS:PRINT @ 42,"dictionary"
1040 PRINT:PRINT "FUNCTIONS AVAILABLE:"
1050 PRINT "     1>DISPLAY DICTIONARY"
1060 PRINT "     2>DISPLAY CHARACTER SET"
1070 PRINT "     3>LOAD/SAVE DATA"
1080 PRINT "     4>INITIALISE"
1090 PRINT "     5>STOP"
1100 PRINT:INPUT "WHICH DO YOU REQUIRE:"
;Z:CLS
1110 IF Z<4 THEN ON Z GOSUB 2000,2500,60
00:GOTO 1000
1120 ON Z-3 GOTO 1500,1140
1130 GOTO 1000
1140 CLS:PRINT @ 7*32+10,"dictionary":PR
INT:PRINT "        PROGRAM TERMINATED"
1150 STOP
```

A standard menu module.

MODULE 5.2.2

```
1500 REM*************************
1510 REM INITIALISE
1520 REM*************************
1530 PCLEAR 4:CLEAR 15000
1540 DIM DI$(120):DIM CHAR$(40)
1550 LET CI=0:LET DI=0
1560 GOTO 1000
```

This module sets aside sufficient memory for the necessary PMODE and reserves the rest of the available memory for strings as well as setting up the necessary variables.

Commentary

Line 1540: The main dictionary of characters will be held in the string array DI$. The number of elements which this array will be capable of holding will depend on the complexity of the characters and, therefore, the length of the strings required to DRAW them. The character set currently being compiled will be held in the string array CHAR$.

Line 1550: CI and DI record the number of characters stored in the character set and the dictionary.

MODULE 5.2.3

```
2000 REM**************************
2010 REM DISPLAY DICTIONARY
2020 REM**************************
2030 LET S=0
2040 PMODE 4,1:PCLS:SCREEN 1,0
2050 FOR I=S TO S+31
2060 DRAW "BM"+STR$(32*((I-S)-8*INT((I-S
)/8)))+","+STR$(45*INT((I-S)/8))+";"+DI$
(I)
2070 NEXT I
2080 LET S1=S-32*INT(S/32)
2090 LET T$=INKEY$:IF T$<>"" THEN GOTO 2
150
2100 FOR I=1 TO 2
2110 DRAW "C"+STR$(I)+";BM"+STR$(32*(S1-
8*INT(S1/8))+8)+","+STR$(45*INT(S1/8)+40
)+";E3;F3"
2120 FOR J=1 TO 25:NEXT J
2130 NEXT I
2140 GOTO 2090
2150 LET S1=S1-(T$=CHR$(9))+(T$=CHR$(8))
:LET S1=S1-(S1<0)+(S1>31)
2160 IF T$="D" THEN FOR I=S+S1 TO DI-1:L
ET DI$(I)=DI$(I+1):NEXT I:LET DI=DI-1:GO
TO 2040
2170 IF T$="C" THEN IF CI<=40 THEN LET C
HAR$(CI)=DI$(S+S1):LET CI=CI+1
2180 IF T$=CHR$(10) THEN LET S=S-32*(S<1
28):GOTO 2040
2190 IF T$=CHR$(94) THEN LET S=S+32*(S>3
1):GOTO 2040
2200 IF T$="Q" THEN RETURN
2210 GOTO 2090
```

The purpose of this module is to display the dictionary of characters page by page and to move a cursor around the page allowing the user to specify characters for a number of simple operations.

Commentary

Line 2060: The fairly involved figures which are to be included in the string to be DRAWn simply specify that each character to be drawn will be placed 32 pixels to the right of the last, or at the start of the screen and 45 pixels down if the end of a line has been reached. This allows for the full 32*32 grid on which the character was designed plus room for a moving cursor.

Line 2080: While the variable S records the absolute position of the character currently pointed to within the dictionary, S1 is used to indicate the position of the cursor on the screen.

Line 2100–2130: A flashing cursor routine which uses the value of the loop variable I to set the colour with which the cursor is DRAWn and thus needs only the one line to DRAW and reDRAW to invisibility.

Line 2150: The cursor move line, based on the left and right arrowed keys.

Line 2160: Input of D will result in the deletion of the character to which the cursor is pointing from the dictionary.

Line 2170: Input of C adds the character to which the cursor is pointing to the current character set.

Lines 2180–2190: The up and down arrows are used to move to the previous or following page of the dictionary.

Line 2200: Input of Q returns program execution to the menu.

Testing Module 5.2.3

Since no characters have yet been loaded from tape, it is difficult to test this module but since there are almost bound to be errors in entering it we shall adopt the temporary expedient of entering some simple specimen characters with the following line:
8888 LET D$ = "BM + 1, + 0;R0;":FOR H = 0 TO 7:LET E$ = "":FOR I = 0 TO 13:LET E$ = E$ + D$:LET DI$(H*14 + I) = E$:NEXT I:NEXT H:LET DI = 110

This line, provided that the program has been initialised, can be called in direct mode or even called as a subroutine from the initialisation module and will load the dictionary with 112 characters which are actually sets of 14 lines of increasing length traversing the 32*32 pixel space diagonally from the top left corner.

Having run line 8888, calling up this module should display the first page of the dictionary and allow the full range of functions specified in the commentary.

MODULE 5.2.4

```
2500 REM****************************
2510 REM DISPLAY CHARACTER SET
2520 REM****************************
2530 PMODE 4,1:PCLS:SCREEN 1,0
2540 FOR I=0 TO CI
2550 DRAW "BM"+STR$(32*(I-8*INT(I/8)))+"
,"+STR$(32*INT(I/8))+";"+CHAR$(I)
2560 NEXT I
2570 LET T$=INKEY$:IF T$="" THEN GOTO 25
70
2580 IF T$="D" THEN LET CI=0
2590 RETURN
```

Having begun to build up a character set from the main dictionary, this module allows the user to display the current state of the character set.

Commentary

Line 2580: Input of D while the character set is being displayed will result in the character set being deleted. Note that this is achieved simply by setting CI to zero. There is no need to physically wipe out the character set. Pressing any key other than D will return to the menu.

Testing Module 5.2.4

You should now be able to create a character set from the main dictionary and display that character set.

MODULE 5.2.5

```
6000 REM***************************
6010 REM DATA FILES
6020 REM**************************
6030 MOTOR ON:AUDIO ON:CLS:INPUT "POSITI
ON TAPE THEN PRESS enter  (MOTOR IS ON):
";Q$
6040 MOTOR OFF:INPUT "PLACE RECORDER INT
O CORRECT MODETHEN PRESS enter";Q$
6050 PRINT:PRINT "FUNCTIONS AVAILABLE:",
"1)SAVE CHARACTER SET","2)LOAD NEW CHARA
CTER","3)SAVE DICTIONARY","4)LOAD DICTIO
NARY":INPUT "WHICH DO YOU REQUIRE:";Q:ON
 Q GOTO 6070,6150,6210,6280
6060 RETURN
6070 MOTOR ON:FOR I=1 TO 10000:NEXT
6080 OPEN "O",£-1,"CHARSET"
6090 PRINT £-1,CI
6100 FOR I=0 TO CI-1
6110 PRINT £-1,CHAR$(I)
6120 NEXT I
6130 CLOSE £-1
6140 RETURN
6150 IF DI=100 THEN RETURN
6160 OPEN "I",£-1,"CHAR"
6170 INPUT £-1,D$
6180 CLOSE£-1
6190 LET DI$(DI)=D$:LET DI=DI+1
6200 RETURN
6210 MOTOR ON:FOR I=1 TO 10000:NEXT I:OP
EN "O",£-1,"DICT"
6220 PRINT £-1,DI
6230 FOR I=0 TO DI-1
6240 PRINT £-1,DI$(I)
6250 NEXT I
6260 CLOSE £-1
6270 RETURN
6280 OPEN "I",£-1,"DICT"
6290 INPUT £-1,DI
6300 FOR I=0 TO DI-1
6310 INPUT £-1,DI$(I)
6320 NEXT I
6330 CLOSE £-1
6340 GOTO 1000
6350 LET D$="BM+1,+0;R0;":FOR H=0 TO 2:L
ET E$="":FOR I=0 TO 13:LET E$=E$+D$:LET
DI$(H*14+I)=E$:NEXT I:NEXT H:LET DI=40
6360 RETURN
```

This is our standard datafile handling module expanded to take account of the fact that we now wish to load or save four different sets of data — individual characters from tape, character sets to tape, the dictionary from tape and the dictionary to tape.

Commentary

Lines 6070–6140: This section saves the current character set to tape, together with the variable CI, which indicates how many characters it contains.

Lines 6150–6200: This section loads a single character from tape and stores it in the dictionary.

Lines 6210–6270: This section stores the current dictionary onto tape.

Lines 6280–6340: This section loads a dictionary from tape. Note that a new dictionary can be loaded during the creation of a character set, thus allowing the character set to draw upon a wider range of characters than can be contained within one dictionary.

Testing Module 5.2.5

You should now be able to pick up characters created by the previous program, compile them into a dictionary or dictionaries and, using these dictionaries compile your own character sets and save them on tape. If these functions are all available, the program is correctly entered and ready for use.

DICTIONARY: Summary of single-key functions

With flashing cursor:
Left and right arrows move cursor.
Up and down arrows move display to new page of dictionary.

D	deletes character above cursor from dictionary.
C	adds character above cursor to current character set.
Q	returns control to menu.

No flashing cursor (character set display):

D	deletes current character set.

Any other key returns to menu.

Summary

This is an uncomplicated program for the simple reason that it is designed to leave the maximum amount of space for the strings containing the actual characters themselves. Once entered you are ready to embark on the task of creating and compiling sets of characters for use in high resolution mode. As previously mentioned, later programs will take you further by showing some practical ways to use such character sets without having to specify the DRAWing of each character separately.

Going further

1) As with the character creator itself, this program will only come into its own when you get around to compiling a dictionary or two.

2) Text is not the only area where the programmer might benefit from having a set of characters available in high resolution modes. What about developing sets of symbols for electronic diagrams, for instance. Remember that, using the DRAW command such symbols can be rotated, so that a single symbol is all that will be necessary for each component, no matter what its orientation may be. You could, perhaps, add the ability to DRAW such characters to a program such as Designer thus allowing symbols and text to be made an integral part of the designs created using that program.

CHAPTER 6
Handy programs

In this chapter we turn our attention to a collection of programs under the general heading of 'utilities', designed to display a few of the wide ranging uses to which the Dragon can be put around the home.

In most of these programs we shall be employing techniques which we have already come across and, apart from such explanation as is necessary to understand the functioning of the program, comments will be accordingly brief.

6.1 NAME AND NUMBER

Once again a general purpose tool which enables a dictionary to be built up of items together with the units in which they are usually measured and an associated quantity. Current working lists can be constructed out of the dictionary items. At first sight this might not be a very inspiring prospect, but on reflection you may find that there are already a number of areas where such a program could come in useful. One might be the field of calory control, where the Dragon is capable of storing a dictionary of up to 500 foods, together with the units in which they are usually measured and the calories per unit. A day's calorific intake can be easily calculated by using the current list facility to construct a list of the day's or the week's food and automatically calculate the number of calories involved. The program is also useful in calculating total prices for orders where the total stock of items does not exceed 500 types.

MODULE 6.1.1

```
6000 REM*************************
6010 REM DATA FILES
6020 REM*************************
6030 AUDIO ON:MOTOR ON:PRINT:INPUT "POSI
TION TAPE THEN PRESS enter  (MOTOR IS ON
)";Q$:MOTOR OFF
6040 PRINT:INPUT "PLACE RECORDER IN CORR
ECT MODE  THEN PRESS enter";Q$
6050 PRINT:PRINT "FUNCTIONS AVAILABLE:",
"1)SAVE DATA",,"2)LOAD DATA":INPUT "WHIC
H DO YOU REQUIRE:";Q:ON Q GOTO 6070,6140
6060 RETURN
6070 MOTOR ON:FOR I=1 TO 10000:NEXT I
6080 OPEN"O",£-1,"NNUMBER"
6090 PRINT £-1,NAME$,QUANTITY$
6100 PRINT £-1,CURR:FOR I=0 TO CURR-1:PR
INT £-1,T$(I,0),T$(I,1),T(I):NEXT
6110 PRINT £-1,ITEMS:FOR I=0 TO ITEMS-1:
PRINT £-1,A$(I,0),A$(I,1),C(I):NEXT
6120 CLOSE£-1
```

```
6130 RETURN
6140 CLEAR 10000:LET FLAG=1:GOTO 1540
6150 OPEN"I",£-1,"NNUMBER"
6160 INPUT £-1,NAME$,QUANTITY$
6170 INPUT £-1,CURR:FOR I=0 TO CURR-1:IN
PUT £-1,T$(I,0),T$(I,1),T(I):NEXT
6180 INPUT £-1,ITEMS:FOR I=0 TO ITEMS-1:
INPUT £-1,A$(I,0),A$(I,1),C(I):NEXT
6190 CLOSE£-1
6200 GOTO 1000
```

A standard data-file module.

MODULE 6.1.2

```
7000 REM************************
7010 REM FORMAT TITLES
7020 REM************************
7030 LET P2=14-INT(LEN(F$)/2)
7040 PRINT @ 32*P1+P2,STRING$(LEN(F$)+2,
150)
7050 PRINT @ 32*(P1+1)+P2,CHR$(150)+F$+C
HR$(150)
7060 PRINT @ 32*(P1+2)+P2,STRING$(LEN(F$
)+2,150)
7070 RETURN
```

A standard title-formatting module.

MODULE 6.1.3

```
1000 REM************************
1010 REM MENU
1020 REM************************
1030 CLS:LET F$="NAME AND NUMBER":LET P1
=0:GOSUB 7000
1040 PRINT:PRINT "COMMANDS AVAILABLE:"
1050 PRINT "    1)DISPLAY CURRENT LIST"
1060 PRINT "    2)INPUT TO CURRENT LIST"
1070 PRINT "    3)START FRESH LIST"
1080 PRINT "    4)DELETE FROM CURRENT LIS
T"
1090 PRINT "    5)EXTEND DICTIONARY"
1100 PRINT "    6)DISPLAY DICTIONARY"
1110 PRINT "    7)DATA FILES"
1120 PRINT "    8)INITIALISE"
1130 PRINT "    9)STOP"
1140 INPUT "WHICH DO YOU REQUIRE:";Z:CLS
1150 ON Z GOSUB 2000,2500,3000,5500,3500
,5000,6000,1500,1170
1160 CLS:GOTO 1000
1170 CLS:LET F$="NAME AND NUMBER":LET P1
=5 :GOSUB 7000
1180 LET F$="PROGRAM NOW STOPPED":LET P1
=10:GOSUB 7000
1190 END
```

A standard menu module.

MODULE 6.1.4

```
1500 REM************************
1510 REM VARIABLES
1520 REM************************
1530 CLEAR 10000:LET FLAG=0
1540 DIM A$(500,1),C(500),T$(50,1),T(50)
1550 LET CURR=0:LET ITEMS=2
1560 LET A$(0,0)="!":LET A$(1,0)="ZZZZZZ
ZZZZZZ"
1570 IF FLAG=1 THEN GOTO 6150
1580 PRINT:INPUT "NAME FOR ITEMS";NAME$
1590 PRINT: INPUT "NAME FOR ASSOCIATED U
NITS";QUANTITY$
1600 GOTO 1000
```

The purpose of this module, apart from initialising the variables to be used, is to allow the user to specify the type of item the program is to be applied to (e.g. FOOD) and a general name for the units of measurement (e.g. weight unit). The actual unit of measurement will be specified alongside the item as it is entered and can differ from item to item. Thus, in the case of food, some items may be measured in ounces, some in pints etc. You may like to note in passing that in line 1550 a variable is declared with a value of zero. This does not need to be done since the first time the Dragon encounters the variable name it will assume the value to be zero anyway. The question is, however, is a program easier to understand if all the major variables are listed in the initialisation module or is this a waste of space?

MODULE 6.1.5

```
3500 REM******************************
3510 REM EXTEND DICTIONARY
3520 REM******************************
3530 IF ITEMS=500 THEN PRINT "NO MORE RO
OM IN DICTIONARY.":FOR I=1 TO 5000:NEXT:
RETURN
3540 CLS:LET F$="NEW ITEMS FOR DICTIONAR
Y":LET P1=0:GOSUB 7000
3550 PRINT:PRINT NAME$;":";"<NAME OR 'ZZ
Z' TO QUIT>:":INPUT F$:IF F$="ZZZ" THEN
RETURN
3560 PRINT:PRINT QUANTITY$;":":INPUT G$
3570 PRINT: PRINT "QUANTITY PER ";G$:INP
UT NN
3580 PRINT:INPUT "ARE THESE CORRECT <Y/N
>:";Q$:IF Q$="N" THEN GOTO 3540
3590 CLS:GOSUB 4000:GOSUB 4500
3600 LET ITEMS=ITEMS+1:GOTO 3540
```

This module accepts inputs to the main dictionary of foods and calls up later modules to actually insert the items. In the case of the food example, the prompts for this module would be "FOOD:", "WEIGHT UNIT:" and QUANTITY PER (whatever is input under weight unit)".

Testing Module 6.1.5

You should now be able to input items under your specified prompts, though they will not be stored in the main arrays.

MODULE 6.1.6

```
4000 REM******************************
4010 REM BINARY SEARCH
4020 REM******************************
4030 LET POWER=(INT(LOG(ITEMS-1)/LOG(2))
)
4040 LET SEARCH=2^POWER
4050 FOR I=POWER-1 TO 0 STEP -1
4060 IF A$(SEARCH,0)<F$ THEN LET SEARCH=
SEARCH+2^I
4070 IF A$(SEARCH,0)>F$ THEN LET SEARCH=
SEARCH-2^I
4080 IF SEARCH<1 THEN LET SEARCH=1
4090 IF SEARCH>ITEMS-1 THEN LET SEARCH=I
TEMS-1
4100 NEXT I
4110 IF A$(SEARCH,0)<F$ THEN LET SEARCH=
SEARCH+1
4120 RETURN
```

A standard binary search module, less the section which actually inserts items into the file. This is held separately because the binary search module is used by several modules, not all of which require an item to be inserted.

MODULE 6.1.7

```
4500 REM*************************
4510 REM INSERT ITEM
4520 REM*************************
4530 FOR I=ITEMS TO INT(SEARCH)+1 STEP -
1:LET A$(I,0)=A$(I-1,0):LET A$(I,1)=A$(I
-1,1):LET C(I)=C(I-1):NEXT I
4540 LET A$(SEARCH,0)=F$:LET A$(SEARCH,1
)=G$:LET C(SEARCH)=NN
4550 RETURN
```

This module actually inserts the new items into the main dictionary.

Testing Module 6.1.7

You should now be able to insert items and find them stored in the main arrays.

MODULE 6.1.8

```
5000 REM*************************
5010 REM USER SEARCH
5020 REM*************************
5030 LET SEARCH=1
5040 CLS:PRINT:PRINT "ON APPEARANCE OF I
TEM, INPUT:"
5050 PRINT @15*32,"TOTAL ITEMS:";ITEMS-2
5060 PRINT @ 1*32,">'ENTER' FOR NEXT ITE
M"
5070 PRINT ">ITEM TO BE SEARCHED FOR"
5080 PRINT ">POSITIVE OR NEGATIVE NUMBER
,    PRECEDED BY '£' TO MOVE POINTER"
5090 PRINT ">'DDD' TO DELETE ITEM"
5100 PRINT ">'ZZZ' TO QUIT FUNCTION"
5110 PRINT STRING$(32,"£");
5120 PRINT "ENTRY NO:-";INT(SEARCH)
5130 PRINT NAME$;":";A$(SEARCH,0)
5140 PRINT QUANTITY$;":";A$(SEARCH,1)
5150 PRINT "QUANTITY PER ";A$(SEARCH,1);
":";C(SEARCH)
5160 INPUT "WHICH DO YOU REQUIRE:";F$
5170 IF F$="DDD" THEN FOR I=SEARCH TO IT
EMS-2:LET A$(I,0)=A$(I+1,0):LET A$(I,1)=
A$(I+1,1):LET C(I)=C(I+1):NEXT I:LET ITE
MS=ITEMS-1:GOTO 5040
5180 IF F$="ZZZ" THEN RETURN
5190 IF F$="" THEN LET SEARCH=SEARCH+1:I
F SEARCH>ITEMS-2 THEN RETURN ELSE GOTO 5
040
5200 IF LEFT$(F$,1)<>"£" THEN GOSUB 4000
:GOTO 5040
5210 LET SEARCH=SEARCH+VAL(MID$(F$,2))
5220 IF SEARCH>ITEMS-2 THEN LET SEARCH=I
TEMS-2
5230 IF SEARCH<1 THEN LET SEARCH=1
5240 GOTO 5040
```

A standard user search module with the one difference that an actual search for a specified item is carried out using the Binary Search module.

Testing Module 6.1.8

You should now be able to display data entered, to page through the dictionary, to jump backwards and forwards, to discover a named item and to delete items at will.

MODULE 6.1.9

```
2500 REM***************************
2510 REM EXTEND CURRENT LIST
2520 REM***************************
2530 IF CURR=50 THEN PRINT "CURRENT LIST
 FULL.":FOR I= 1 TO 5000:NEXT:RETURN
2540 LET F$="EXTEND CURRENT LIST":LET F1
=0:GOSUB 7000
2550 PRINT:PRINT NAME$;:INPUT F$
2560 GOSUB 4000:IF A$(SEARCH,0)<>F$ THEN
 PRINT @ 10*32,NAME$;" UNKNOWN--PLEASE C
HECK.":FOR I=1 TO 5000:NEXT I:RETURN
2570 PRINT:PRINT QUANTITY$;":";A$(SEARCH
,1)
2580 PRINT:INPUT "QUANTITY:";Q
2590 PRINT: INPUT "ARE THESE CORRECT (Y/
N):";Q$:CLS:IF Q$="N" THEN GOTO 2500
2600 :LET T$(CURR,0)=A$(SEARCH,0):LET T$
(CURR,1)=STR$(Q)+" "+A$(SEARCH,1):LET T(
CURR)=Q*C(SEARCH):LET CURR=CURR+1
2610 PRINT:INPUT "ANY MORE ITEMS (Y/N):"
;Q$:CLS:IF Q$="Y" THEN GOTO 2500
2620 RETURN
```

This module inputs items for inclusion in the current list, that is a working list of items contained in the main dictionary, which can be manipulated without corrupting the data contained in the main dictionary. In the case of the food example, the user would be required to supply the food type, the program would then supply the unit of measurement and the user would specify how many of those units were to be included.

MODULE 6.1.10

```
2000 REM***************************
2010 REM DISPLAY CURRENT LIST
2020 REM***************************
2030 IF CURR=0 THEN RETURN
2040 LET SUM=0:FOR I=0 TO CURR-1
2050 PRINT NAME$;":";T$(I,0)
2060 PRINT QUANTITY$;":";T$(I,1)
2070 PRINT "QUANTITY:";T(I)
2080 PRINT STRING$(32,"£");
2090 IF INKEY$="" THEN GOTO 2090
2100 LET SUM=SUM+T(I)
2110 NEXT I
2120 PRINT "TOTAL QUANTITY:";SUM
2130 PRINT:INPUT "PRESS enter TO CONTINU
E";Q$
2140 RETURN
```

This module displays the current list, item by item, and totals the quantities involved (calories in the case of the food example). The user is required to press any key to display the next item in order to prevent the list scrolling up the page too fast to be read.

Testing Module 6.1.10

You should now be able to load items from the main dictionary into the current list and to display that list.

MODULE 6.1.11

```
3000 REM***************************
3010 REM INITIALISE CURRENT LIST
```

```
3020 REM****************************
3030 FOR I=0 TO 50:LET T$(I,0)="":LET T$
(I,1)="":LET T(I)=0:NEXT I:LET CURR=0:RE
TURN
```

This module initialises the current list only.

MODULE 6.1.12

```
5500 REM****************************
5510 REM CURRENT LIST DELETIONS
5520 REM****************************
5530 FOR I=0 TO CURR-1:PRINT T$(I,0)
5540 PRINT T$(I,1)
5550 INPUT "ddd=DELETE/enter=NEXT/zzz=QU
IT";Q$:IF Q$="ZZZ" THEN RETURN
5560 IF Q$="" THEN NEXT I:RETURN
5570 FOR J=I TO CURR-1
5580 LET T$(J,0)=T$(J+1,0)
5590 LET T$(J,1)=T$(J+1,0)
5600 LET T(J)=T(J+1)
5610 NEXT J
5620 LET CURR=CURR-1
5630 RETURN
```

This module accomplishes deletions from the current list.

Testing Module 6.1.12

You should now be able to delete items from the current list or to initialise it.

Summary

The program is yet another good example of the power of modular programming since most of the modules have been lifted, with very little modification, from other programs in this book.

From this you may draw the valuable lesson that, provided you clearly distinguish the functional units of a program, it is always methods that are more important to your progress as a programmer than the actual number of programs you have written. A good library of programs will stand you in good stead until a totally new application comes along. A good collection of methods, contained in clearly identifiable modules, will never let you down. So don't limit yourself to methods which you need for only present day applications. If you see an interesting way of doing things in a magazine or book, write a simple program to use it, just for the hell of it. Within a week or two you may well find that it is just what you are looking for, for that new program that is giving you so much trouble.....

6.2 TYPIST

I have to confess that I am inordinately fond of this program. Its presence here proves that a program doesn't have to be long to be useful — this one is short, neat and outstandingly good at what it does..... and what it does is help you to learn to touch type.

MODULE 6.2.1

```
1000 REM***************************
1010 REM PRINT KEY BOARD
1020 REM***************************
1030 CLS:PRINT @ 33, CHR$(138);STRING$(2
7,163);CHR$(133)
1040 LET A$="1234567890:-b":PRINT CHR$(1
42);CHR$(136);:FOR I=1 TO 13:PRINT CHR$(
175);MID$(A$,I,1);:NEXT I:PRINT CHR$(175
);CHR$(132);CHR$(141)
1050 LET A$="^QWERTYUIOP@"+CHR$(95)+CHR$
(95):PRINT CHR$(138);:FOR I=1 TO 14:PRIN
T CHR$(175);MID$(A$,I,1);:NEXT I:PRINT C
HR$(175);CHR$(132);CHR$(141);
1060 LET A$="^ASDFGHJKL;":PRINT CHR$(139
);CHR$(130);:FOR I=1 TO 11:PRINT CHR$(17
5);MID$(A$,I,1);:NEXT I: PRINT CHR$(175)
;"[illegible]";CHR$(175);"c";CHR$(175);CHR$(133)
1070 LET A$="ZXCVBNM,./":PRINT " ";CHR$(
138);CHR$(175);"ss";:FOR I=1 TO 10:PRINT
 CHR$(175);MID$(A$,I,1);:NEXT I:PRINT CH
R$(175);"ss";CHR$(175);CHR$(129);CHR$(13
1);CHR$(135);
1080 PRINT " ";CHR$(138);STRING$(27,172)
;CHR$(133)
```

All that this module does is to print a fairly crude-looking representation of the keyboard at the top of the screen. You will note that the down arrow and the right arrow are not represented properly because they are not available in the character set.

MODULE 6.2.2

```
2000 REM***************************
2010 REM ACCEPT INPUT
2020 REM***************************
2030 LET SUM=0:LET RIGHT=0:RESTORE
2040 READ A$:IF A$="STOP" THEN RESTORE:R
EAD A$
2050 IF LEN (A$)>31 THEN PRINT "STRING T
OO LONG":STOP
2060 PRINT @ 8*32,A$:PRINT @ 9*32,STRING
$(32," "):PRINT @ 9*32,"";
2070 FOR I=1 TO LEN(A$)
2080 LET T$=INKEY$:IF T$="" OR T$=CHR$(8
) THEN GOTO 2080
2090 LET SUM=SUM+1:SOUND 150,1
2100 IF T$=MID$(A$,I,1) THEN GOTO 2120 E
LSE PRINT T$;CHR$(95);
2110 LET T$=INKEY$:IF T$="" OR T$=CHR$(8
) THEN GOTO 2110 ELSE PRINT CHR$(8);CHR$
(8);:GOTO 2090
2120 LET RIGHT=RIGHT+1:IF T$<>" " THEN P
RINT T$; ELSE PRINT CHR$(159);
2130 NEXT I
2140 PRINT @ 12*32,INT(RIGHT/SUM*100);"%
"
2150 INPUT "MORE (Y/N):";Q$:IF Q$<>"N" T
HEN GOTO 2040
```

This module displays a line of text on the screen underneath the keyboard for the user to copy.

Commentary

Lines 2040–2050: Text to be copied is entered in the form of DATA statements after line 3000. Text must be no more than 32 characters long for any one unit.

Line 2060: The next line is cleared for the user's input.

Lines 2070–2180: The INKEY$ function is used to obtain any character which the user inputs. The character is displayed on the screen underneath the line to be copied. If it corresponds to the next character to be typed in the text to be copied, then the program moves onto the next position. If the input is incorrect an arrow is placed next to it and the program stays with that position until the correct character is entered. Correct keystrokes are recorded along with the total number of keystrokes, and the percentage success rate is displayed at the end of the line.

MODULE 6.2.3

```
3000 REM***************************
3010 REM DATA FOR TESTS
3020 REM***************************
3030 DATA "THIS IS A DRAGON TYPING TEST"
3040 DATA "JUST TYPE WHAT YOU SEE"
3050 DATA "DON'T LOOK DOWN AT THE KEYBOA
RD"
3060 DATA "STOP"
```

This is included as an example of what may be entered. Note that the text must end with a DATA statement with STOP in it — this will redirect the program to the beginning of the material again.

Going Further

1) What about including some reference to the Dragon's timing function, so that an assessment of speed can be made.
2) Perhaps when a wrong key is depressed it could flash, thus giving the user a better indication of where he or she is going wrong.
3) Far better than random sentences would be to enter some exercises from a good typing tutor.

6.3 TEXTED

Another useful and none too lengthy contribution in the text field, this one thinks that it's a word processor. Indeed it is capable of apeing many of the abilities of more expensive systems. Built into it are some features of special interest to those who own, or hope to own, a printer to go with their Dragon.

MODULE 6.3.1

```
1000 REM***************************
1010 REM INITIALISE
1020 REM***************************
1030 CLS
1040 PCLEAR 1:CLEAR 20000
1050 DIM TEXT$(500)
1060 LET LL=1:LET PLACE=1
1070 LET TEXT$(0)=STRING$(32,CHR$(118))
1080 LET TEXT$(1)=STRING$(32,CHR$(126))
1090 GOSUB 2120
```

This module initialises the main array TEXT$, which is used to hold the text input by the user. The two strings stored in positions zero and one are simply visual markers of the beginning and end of the text.

MODULE 6.3.2

```
1500 REM**************************
1510 REM EDIT LINE
1520 REM**************************
1530 LET A$=" "
1540 LET P=0:PRINT @ 12*32,A$
1550 LET T$=INKEY$:IF T$="" THEN POKE 10
24+12*32+P,175:POKE 1024+12*32+P,ASC(MID
$(A$,P+1)):GOTO 1550
1560 IF LEN(A$)=65 OR T$=CHR$(13) THEN G
OSUB 2000
1570 IF T$=CHR$(94) THEN GOSUB 2500
1580 IF P<LEN(A$)-1 AND T$=CHR$(12) THEN
 LET A$=LEFT$(A$,P)+MID$(A$,P+2):GOTO 16
10
1590 IF T$<>CHR$(8) AND T$<>CHR$(9) AND
(ASC(T$)<32 OR ASC(T$)>90) THEN GOTO 154
0
1600 IF  T$<>"" AND T$<>CHR$(8) AND T$<>
CHR$(9) THEN LET A$=LEFT$(A$,P)+T$+MID$(
A$,P+1):LET P=P+1
1610 PRINT @ 12*32,A$
1620 IF T$=CHR$(9) AND P<LEN(A$)-1 THEN
LET P=P+1
1630 IF T$=CHR$(8) AND P>0 THEN LET P=P-
1
1640 GOTO 1550
```

The function of this module is to allow the user to build up two lines of text at the bottom of the screen, including the ability to edit them, before they are placed into the main body of text at a specified point.

Commentary

Line 1550: The purpose of this line is to flash a cursor over the letter of the string A$ pointed to by the variable P.

Line 1560: A string is entered into the main body of text either when the user presses ENTER or when the length of the string reaches two lines of display.

Line 1570: Pressing the up arrow key calls up another part of the editing mode which will be discussed later.

Line 1580: Pressing the CLEAR key results in the deletion of the letter over which the cursor is currently flashing.

Line 1600: If an input falls into the group of normal text characters, then it is added to the string being built up.

Lines 1620–1630: The left and right arrowed keys move the flashing cursor along the line in the desired direction.

Testing Module 6.3.2

Entering a temporary line 2120 RETURN should allow you to build up two lines of text in the lower part of the display. You should be able to move the cursor backwards and forwards over the string, to delete letters or words and to insert letters or words, either at the end or into the middle of the string.

MODULE 6.3.3

```
2000 REM**************************
2010 REM INSERT LINE
2020 REM *************************
2030 IF LEN(A$)>33 THEN LET X=2 ELSE LET
 X=1
2040 FOR I=LL+X TO PLACE+X STEP -1:LET T
EXT$(I)=TEXT$(I-X):NEXT I
2050 IF LEN(A$)>33 THEN LET TEXT$(PLACE)
=LEFT$(A$,32):LET TEXT$(PLACE+1)=MID$(A$
,33,LEN(A$)-33):ELSE LET TEXT$(PLACE)=LE
FT$(A$,LEN(A$))
2060 FOR I=0 TO X-1
2070 IF RIGHT$(TEXT$(PLACE+I),1)=" " THE
N LET TEXT$(PLACE+I)=LEFT$(TEXT$(PLACE+I
),LEN(TEXT$(PLACE+I))-1):GOTO 2070
2080 NEXT I
2090 LET A$=" ":LET P=0:PRINT @ 13*32,""
:PRINT @ 12*32,A$
2100 PRINT @ 14*32," "
2110 LET LL=LL+X:LET PLACE=PLACE+X
2120 IF PLACE<5 THEN LET START=0 ELSE LE
T START=PLACE-5
2130 CLS:FOR I=START TO START+9:PRINT TE
XT$(I);:IF LEN(TEXT$(I))<32 THEN PRINT
2140 IF I=PLACE-1 THEN PRINT CHR$(62)
2150 NEXT I:PRINT STRING$(32,CHR$(175))
2160 RETURN
```

This module inserts the line of text built up in the lower half of the screen into the main body of the text and prints a part of the main body of text in the top half of the screen.

Commentary

Lines 2030–2050: Depending on whether one or two lines of text were entered onto the lower part of the screen, a space is made for it (them) in the main array at the point indicated by the variable LL.

Lines 2060–2080: The newly entered lines are stripped of any trailing spaces which take up memory unnecessarily.

Lines 2090–2100: A$ is reset equal to one space (note that here and in line 1530 A$ is not an empty string — it is made up of one space). And both of the lines used in the lower half of the screen are cleared.

Lines 2120–2150: The portion of text around the newly inserted lines is reprinted on the upper half of the screen to include them.

Testing Module 6.3.3

You should now be able to enter your text onto the upper half of the screen by pressing ENTER when you have built up a satisfactory string at the bottom of the screen.

MODULE 6.3.4

```
2500 REM****************************
2510 REM MOVE EDIT LINE
2520 REM****************************
2530 IF PLACE<5 THEN LET P2=PLACE ELSE L
ET P2=5
2540 LET T1$=INKEY$:IF T1$="" THEN PRINT
 @ P2*32," ":PRINT @ P2*32,">":GOTO 2540
2550 IF PLACE>1 AND T1$=CHR$(94) THEN LE
T PLACE=PLACE-1:GOSUB 2120
2560 IF PLACE>10 AND T1$="Q" THEN LET PL
ACE=PLACE-10:GOSUB 2120
2570 IF PLACE<LL AND T1$=CHR$(10) THEN L
ET PLACE=PLACE+1:GOSUB 2120
2580 IF PLACE<LL-9 AND T1$="A" THEN LET
PLACE=PLACE+10:GOSUB 2120
2590 IF T1$=CHR$(13) THEN RETURN
2600 IF PLACE<LL AND T1$="D" THEN LET LL
=LL-1:FOR I=PLACE TO LL:LET TEXT$(I)=TEX
T$(I+1):NEXT I:LET TEXT$(LL+1)="":GOSUB
2120
2610 IF PLACE<LL AND T1$="C" THEN FOR I=
32 TO 1 STEP -1:IF MID$(TEXT$(PLACE),I,1
)=" " THEN NEXT I:ELSE LET A$=LEFT$(TEXT
$(PLACE),I)+" ":RETURN
2620 IF T1$="F" THEN GOSUB 3000
2630 IF T1$="P" THEN GOSUB 3500:CLOSE£-2
2640 IF T1$="S" THEN GOSUB 6000
2650 GOTO 2530
```

Apart from calling up three more edit functions which have not yet been entered, the purpose of this module is to move a flashing > cursor up or down through the main body of text. The position of the cursor indicates either where new lines are to be inserted or where other edit functions are to be carried out.

The module is called by entering the up arrow in the course of Module 2.

Commentary

Lines 2550–2580: The cursor can be moved by means of the up or down arrows (one space) or the Q and A keys (10 spaces).

Line 2600: Input of D results in the deletion of the line immediately below the cursor.

Line 2610: Input of C results in the line immediately below the cursor being recalled to the bottom of the screen, although it is not deleted from the main body of text.

Line 2620: Input of F calls up the text formatting module.

Line 2630: Input of P results in the current text being printed on a printer if connected.

Line 2640: Input of S calls up the data file module.

Testing Module 6.3.4

Having entered some text you should now be in a position to move the flashing cursor, to delete lines and to recopy to the bottom of the screen lines previously entered.

MODULE 6.3.5

```
3000 REM****************************
3010 REM FORMAT LINE
3020 REM****************************
3030 FOR I=1 TO LL-2
3040 IF LEFT$(TEXT$(I),1)=CHR$(126)THEN
GOTO 3120
3050 IF TEXT$(I)="" OR TEXT$(I+1)="" THE
N GOTO 3110
3060 LET SPACE=32-LEN(TEXT$(I))
3080 LET ITEM=INSTR(TEXT$(I+1)," ")
3090 IF SPACE>=ITEM AND ITEM>0 THEN LET
TEXT$(I)=TEXT$(I)+" "+LEFT$(TEXT$(I+1),I
TEM-1):LET TEXT$(I+1)=MID$(TEXT$(I+1),IT
EM+1):GOTO 3060
3100 IF LEN(TEXT$(I+1))<SPACE THEN LET T
EXT$(I)=TEXT$(I)+" "+TEXT$(I+1):FOR J=I+
1 TO LL:LET TEXT$(J)=TEXT$(J+1):NEXT J:L
ET LL=LL-1:LET PLACE=PLACE-1:GOTO3060
3110 NEXT I
3120 GOSUB 2120
3130 RETURN
```

The effect of this module is to examine each line in the main body of text, together with its following line, and to determine whether the transfer of the first word from the following line would make the first line longer than 32 characters. If it is possible to transfer the first word, then this is done. If it is possible to transfer the whole of the following line into the first line without exceeding 32 characters, then this is done. Only in the case of an empty line will the program not attempt to run together the two lines of text. Empty lines are therefore used to separate paragraphs and the like.

Although not perfectly right-justified, text which has been processed with this module will be quite tidy. The module therefore allows the user to enter text without too much regard for appearances, knowing that the module can be used to tidy up the text later.

Testing Module 6.3.5

You should now be able to input a series of individual words on separate lines, call up this module and see them formatted into a single line. Single words on separate lines can be inserted into the main body of text or lines modified to include a word overlapping onto the next line and this module used to reformat the text. No formatting should take place for the lines immediately before or after an empty line entered into the array.

MODULE 6.3.6

```
3500 REM*************************
3510 REM OUTPUT TO PRINTER
3520 REM*************************
3530 OPEN "O",£-2,"TEXTED"
3540 LET X=1
3550 IF X=LL THEN PRINT £-2,"":RETURN
3560 IF TEXT$(X)="" THEN PRINT £-2,"":LE
T X=X+1:GOTO 3550:ELSE PRINT £-2,TEXT$(X
);"  ";
3570 IF X+1=LL THEN PRINT £-2,"":RETURN
3580 IF TEXT$(X+1)="" THEN PRINT £-2,"":
PRINT £-2,"":ELSE PRINT £-2,TEXT$(X+1)
3590 LET X=X+2:GOTO 3550
3600 PRINT £-2,""
3610 RETURN
```

Input of P from Module 4 will call up this module, which will in turn print the main body of text on a printer. Note that since the width of text on a printer is usually at least double that of the Dragon screen's 32 characters, two array lines are run together for each line printed (except where one of the two is a blank line).

MODULE 6.3.7

```
6000 REM*************************
6010 REM DATA FILES
6020 REM*************************
6030 CLS:MOTOR ON:AUDIO ON:INPUT "POSITI
ON TAPE (MOTOR IS ON)      THEN PRESS ent
er:";Q$
6040 MOTOR OFF:AUDIO OFF:INPUT "PLACE RE
CORDER IN CORRECT MODE  THEN PRESS enter
";Q$
6050 INPUT "1)SAVE~~~~2)RECALL: ";Q:ON Q
GOTO6070,6130
6060 GOTO 6050
6070 MOTOR ON:FOR I=1 TO 10000:NEXT I:OP
EN "O",£-1,"TEXTED"
6080 PRINT £-1,PLACE,LL
6090 FOR I=1 TO LL-1:PRINT£-1,TEXT$(I):N
EXT I
6100 CLOSE £-1
6110 GOSUB 2130
6120 RETURN
6130 OPEN "I",£-1,"TEXTED"
6140 INPUT £-1,PLACE,LL
6150 FOR I=1 TO LL-1:INPUT £-1,TEXT$(I):
NEXT I
6160 GOSUB 2120
6170 CLOSE £-1
6180 LET TEXT$(LL)=STRING$(32,CHR$(126))
6190 RETURN
```

A standard data-file module.

Summary

This program is a tribute to the speed of the Dragon, without which many of the processes involved, including editing on the screen, with its resultant constant reprinting of the string, would be painfully slow. The techniques of on-screen editing bear some study, since altering something while you look at it is by far the easiest way of making changes to almost anything and could be incorporated into a wide variety of programs where strings have to be changed — including most of the programs in this book.

TEXTED: Summary of one-key instructions:

With flashing cursor at bottom of screen:
Text characters may be entered at position of flashing cursor.
Left and right arrows move cursor over string.
Up arrow calls up remaining editing command modules.
ENTER places current string into main body of text at position indicated by >.
CLEAR key erases letter immediately under cursor.

With flashing > at top of screen:

↑, ↓, Q and A	move cursor up and down through main body of text.
D	deletes line immediately below cursor.
C	copies line below cursor to lower part of screen (original is not deleted).
F	formats text.
P	prints text if printer is connected.
S	calls up data-file module.
ENTER	returns control to lower screen cursor.

6.4 MUSIC

All work and no play etc., so this program does just that — plays. Like the DRAW function that we have used in several places, Dragon Data have built a flexible string driven music command called PLAY. In this program we take advantage of the Dragon's ability to play about with strings, to literally edit music.

Before entering this program it would be a good idea to go back to the Dragon manual to check up on the basics of the PLAY. You will find the details in Chapter 9 of the manual.

MODULE 6.4.1

```
6000 REM*************************
6010 REM DATA FILES
6020 REM*************************
6030 MOTOR ON:AUDIO ON:CLS:INPUT "POSTIO
N TAPE THEN PRESS enter    <MOTOR IS ON>:
";Q$
6040 MOTOR OFF:INPUT "PLACE RECORDER IN
CORRECT MODE, THEN PRESS enter:";Q$
6050 PRINT:PRINT "1>SAVE",,"2> RECALL":I
NPUT "WHICH DO YOU REQUIRE:";Q
6060 ON Q GOTO 6080,6160
6070 GOTO 6050
6080 MOTOR ON:FOR I=1 TO 10000:NEXT I
6090 OPEN "O",£-1,"MUSIC"
6100 PRINT £-1,N
6110 FOR I=0 TO N
6120 PRINT £-1,TUNE$(I)
6130 NEXT I
6140 CLOSE £-1
6150 RETURN
6160 OPEN "I",£-1,"MUSIC"
6170 INPUT £-1,N
6180 FOR I=0 TO N
6190 INPUT £-1,TUNE$(I)
6200 NEXT I
6210 CLOSE £-1
6220 RETURN
```

A standard data file module.

MODULE 6.4.2

```
7000 REM***************************
7010 REM HELP
7020 REM***************************
7030 CLS:PRINT "THE FOLLOWING FUNCTIONS
ARE AVAILABLE:"
7040 PRINT:PRINT "THE CURSOR MAY BE MOVE
D BY MEANSOF THE 4 CURSOR ARROWS. THE
  NOTES DISPLAYED CAN BE MOVED 10 FORWAR
DS OR BACKWARDS BY USE OF THE a AND q KE
YS."
7050 PRINT:PRINT "WHEN THE CURSOR IS OPP
OSITE THE REQUIRED NOTE IT CAN ALSO MOVE
D TO THE RELEVANT PART OF THE NOTEBY USE
 OF THE KEYS l,o,v AND n."
7060 PRINT:INPUT "PRESS enter FOR MORE:"
;Q$
7070 CLS:PRINT "A Pause CAN BE INSERTED
BY      PRESSING p WHEN OPPOSITE THE
RELEVANT ENTRY. HAVING ENTERED APAUSE IT
 CANNOT BE CHANGED BACK BUT MUST BE DELE
TED."
7080 PRINT:PRINT "PRESSING d WILL RESULT
 IN THE   DELETION OF THE NOTE NEXT TO T
HECURSOR."
7090 PRINT:PRINT "PRESSING i WILL ALLOW
A NEW NOTETO BE INSERTED IMMEDIATELY
  BEFORE THE CURRENT NOTE."
7100 PRINT:INPUT "PRESS ENTER FOR MORE:"
;Q$
7110 CLS:PRINT "TO CHANGE THE DURATION O
F A NOTETO A DOTTED VALUE, PRESS THE
FULL STOP.   TO CHANGE BACK TO ANORMAL V
ALUE PRESS THE SEMI-    COLON KEY."
7120 PRINT:PRINT "TO DETERMINE THE TEMPO
, PRESS t AND ENTER THE DESIRED VALUE WH
ENPROMPTED."
7130 PRINT:PRINT "TO PLAY THE TUNE PRESS
 m THEN   SPECIFY THE START AND FINISH
  POINTS."
7140 PRINT:INPUT "PRESS enter FOR MORE:"
;Q$
7150 CLS:PRINT:PRINT "IF THE TUNE HAS NO
T BEEN         FINISHED, IT CAN BE SAVED
 IN ITSDEVELOPING FORM BY PRESSING    s.
 THIS SAME KEY WILL ALSO ALLOWTHE RECALL
 OF A PREVIOUS TUNE  WHICH HAS BEEN STO
RED BY THIS   METHOD."
7160 PRINT:PRINT "A FINISHED TUNE CAN BE
 STORED   ON TAPE BY PRESSING c."
7170 PRINT:INPUT "PRESS enter FOR MORE:"
;Q$
7180 CLS:PRINT "REMEMBER THAT YOU DO NOT
 HAVE TOENTER A VALUE FOR EVERY ITEM
THAT MAKES UP THE NOTE. IF YOU  DO NOT S
PECIFY A VALUE, THE ONE GOVERNING THE PR
EVIOUS NOTE WILLBE INSERTED. PRESSING en
ter     RESULTS IN ANY CHANGES BEING REC
ORDED."
7190 PRINT:PRINT "ONCE THE CURSOR IS MOV
ED OVER   THE NOTE, IT CAN ONLY BE MOVED
  OFF AGAIN BY THE USE OF enter."
7200 PRINT:INPUT "PRESS enter TO RETURN
TO MAIN PROGRAM:";Q$
7210 RETURN
```

This is the only program in this book to contain such a module as this, known as a Help function. Its aim is to present the rules of the program on command. It is included here partly because this number has more one-key commands than any other in the book, and partly because I felt you should have an example of such a module anyway. Some people have no trouble with one-key commands, after a couple of tries of the program with the instructions in front of them they never look back. Others do not find them so easy — in which case all they have to do is to press H during the main program section and they can page through these instructions. Such a module could easily be added to many of the programs in this book where memory space is not the prime consideration.

MODULE 6.4.3

```
1000 REM*************************
1010 REM INITIALISE
1020 REM*************************
1030 CLEAR 5000
1040 DIM TUNE$(500)
1050 LET N=1
1060 LET TUNE$(1)="L000;00;V00;00"
1070 DIM LAST$(4):LET LAST$(1)="004;":LE
T LAST$(2)="2":LET LAST$(3)="15":LET LAS
T$(4)="01"
1080 DIM TEMP$(3):DIM STORE$(20)
```

Initialises the program. The arrays will be discussed later.

MODULE 6.4.4

```
1500 REM*************************
1510 REM PRINT NOTES FOR EDITING
1520 REM*************************
1530 IF NN>N THEN LET NN=N
1540 IF NN<1 THEN LET NN=1
1550 LET P=0:LET P2=0
1560 LET N2=13:IF N2+NN>N THEN LET N2=N-
NN
1570 CLS:FOR I=0 TO N2:PRINT USING "££££
";NN+I;:PRINT">";TUNE$(NN+I):NEXT I
1580 LET T$=INKEY$:IF T$="" THEN LET T=P
EEK(1024+P):POKE 1024+P,175:POKE 1024+P,
T:GOTO 1580
1590 IF T$=CHR$(13) THEN GOSUB 2000:LET
P=P2*32:GOTO 1560
1600 IF T$=CHR$(94) AND P2>0 THEN LET P=
P-32
1610 IF T$=CHR$(10) AND P2<N2 THEN LET P
=P+32
1620 IF T$=CHR$(9) AND P-32*P2<18 THEN L
ET P=P+1
1630 IF T$=CHR$(8) AND P-32*P2<>5 THEN L
ET P=P-1
1640 IF T$="A" AND P-32*P2=0 THEN LET NN
=NN+10:GOTO 1530
1650 IF T$="Q" AND P-32*P2=0 THEN LET NN
=NN-10:GOTO 1530
1660 IF T$="I" THEN FOR I=N TO NN+P2 STE
P-1:LET TUNE$(I+1)=TUNE$(I):NEXT I:LET T
UNE$(NN+P2)="L000;00;V00;00":LET N=N+1:G
OSUB 2000:GOTO 1560
1670 IF T$="D" THEN FOR I=NN+P2 TO N:LET
 TUNE$(I)=TUNE$(I+1):NEXT I:LET N=N-1:GO
TO 1560
1680 IF PEEK(1024+P)>111 AND PEEK (1024+
P)<122 AND T$>"/" AND T$<":" THEN MID$(T
UNE$(NN+P2),P-32*P2-4)=T$:PRINT @ P2*32+
5,TUNE$(NN+P2):IF P-32*P2<18 THEN LET P=
P+1
1690 IF T$="L" THEN LET P=32*P2+6
1700 IF T$="O" THEN LET P=32*P2+11
1710 IF T$="V" THEN LET P=32*P2+14
1720 IF T$="N" THEN LET P=32*P2+17
1730 IF T$="P" THEN MID$(TUNE$(NN+P2),1)
="P":LET TUNE$(NN+P2)=LEFT$(TUNE$(NN+P2)
,5):PRINT @ 32*P2+5,TUNE$(NN+P2):LET P=3
2*P2+6
1740 IF T$="M" THEN GOSUB 2500:GOTO 1570
1750 IF T$<>"T" THEN GOTO 1770:ELSE PRIN
T @ 14*32,"";:INPUT "TEMPO:";TEMPO:IF TE
MPO<1 OR TEMPO>255 TNEN GOTO 1750
1760 LET TUNE$(0)="T"+STR$(TEMPO)
1770 IF T$="." THEN MID$(TUNE$(NN+P2),5)
=T$:GOTO 1570
1780 IF T$=";" THEN MID$(TUNE$(NN+P2),5)
=T$:GOTO 1570
1790 IF T$="C" THEN GOSUB 3000:GOTO 1560
1800 IF T$="S" THEN GOSUB 6000:GOTO 1550
1810 IF T$="H" THEN GOSUB 7000:GOTO 1560
1820 LET P2=INT(P/32)
1830 GOTO 1580
```

Believe it or not, there is hardly anything to this module. All it really consists of is a cursor move module with a series of simple operations revolving around moving the cursor or editing the contents of the screen.

Commentary

Line 1530: N is the number of notes so far entered. NN is the position of the cursor in the list of notes.

Line 1550: P represents the position of the cursor on the line. P2 represents the position of the pointer down the page.

Line 1560–1570: N2 is the end of the display — if N2 is 50 then the 14 notes up to note 50 will be displayed.

Line 1580: The same type of flashing cursor as found in Texted.

Line 1590: Pressing ENTER results in a note being registered — you cannot yet do this.

Lines 1600–1610: Up and down arrows move cursor.

Lines 1620–1630: Left and right arrows move cursor.

Lines 1640–1650: As in Texted, the A and Q keys are used to specify a move of 10 places for the cursor.

Line 1660: Pressing I results in a new note being inserted immediately before the note the cursor is currently opposite.

Line 1670: Pressing D deletes the note the cursor is currently opposite.

Line 1680: If the input is in the range 0-9, it replaces the character under the flashing cursor.

Lines 1690–1720: The format of the notes when they are displayed is shown in line 1060. The first four characters will store the length, the next two the octave, the next three the volume and the final two the note — the notation is the one using figures rather than letters to represent notes. Input of L,O,V or N moves the cursor immediately to the first figure (not letter) of the corresponding note. This saves a great deal of button pushing.

Line 1730: Inputting P results in the note being transformed into a pause. The length of the pause can be edited but it cannot be edited back to a note.

Line 1740: Inputting M calls up the module which plays the tune as developed so far.

Lines 1750–1760: Inputting T allows the user to specify the tempo.

Line 1770: Pressing . results in the note length becoming a dotted value.

Line 1780: Inputting ; removes a dotted value, if present.

Line 1790: Inputting C (for compile) leads to the tune being processed into the form of string under which it will eventually be saved and the user given the option of saving it.

Line 1800: Inputting S allows the current tune to be saved in its multi-line form or a tune saved in this form to be picked up and worked upon. Tunes saved in the C format cannot be re-edited by this program.

Line 1820: Inputting H calls up the help function.

Testing Module 6.4.4

Sorry, but you can't yet.

MODULE 6.4.5

```
2000 REM****************************
2010 REM INSERT DEFAULT VALUES
2020 REM****************************
2030 IF LEFT$(TUNE$(P2+NN),1)="P" THEN L
ET TUNE$(P2+NN)=LEFT$(TUNE$(P2+NN),5):GO
TO 2080
2040 RESTORE:FOR I=1 TO 4
2050 READ PL,LL
2060 IF VAL(MID$(TUNE$(P2+NN),PL,LL))=0
THEN MID$(TUNE$(P2+NN),PL)=LAST$(I) ELSE
 LET LAST$(I)=MID$(TUNE$(P2+NN),PL,LL)
2070 NEXT I
2080 IF P2+NN=N THEN LET N=N+1:LET TUNE$
(N)="L000:00:V00:00":LET P2=P2+1
2090 RETURN
2100 DATA 2,4,7,1,10,2,13,2
```

If the thought of entering all the data necessary to fill the format shown in line 1060 has you daunted, then don't worry because for most tunes this module will do the work for you. What it does is to allow you to specify only those values that are different from the previous not entered. Any that are not specified are given a default value equal to their value in the previous note. Default values are set initially by line 1070. Values are inserted by reading the positions of the various sections of each note from the DATA statement.

Testing Module 6.4.5

You should now be able to perform all of the functions described so far except for those of playing the tune and compiling it for permanent storage.

MODULE 6.4.6

```
2500 REM***************************
2510 REM PLAY TUNE
2520 REM***************************
2530 PRINT @ 32*14,"";:INPUT "START POIN
T:";START
2540 INPUT "FINISH POINT:";FINISH:IF FIN
ISH>N-1 THEN LET FINISH=N-1
2550 FOR I=START TO FINISH
2560 INPUT "DISPLAY NOTES (Y/N):";Q$
2570 FOR I=START TO FINISH
2580 IF Q$="Y" THEN PRINT USING "££££";I
;:PRINT ")";TUNE$(I)
2590 PLAY TUNE$(I)
2600 NEXT I
2610 RETURN
```

A straightforward module which plays lines of the array TUNE$ between points specified by the user. Displaying notes as they are played makes the music sound slightly stacatto.

Testing Module 6.4.6

You should now be able to play part of any tune you have entered.

MODULE 6.4.7

```
3000 REM***************************
3010 REM STORE TUNE
3020 REM***************************
3030 LET TEMP$(0)=LEFT$(TUNE$(1),5)
3040 LET TEMP$(1)=MID$(TUNE$(1),6,3)
3050 LET TEMP$(2)=MID$(TUNE$(1),9,4)
3060 CLS:INPUT "NAME FOR THIS TUNE:";Q$:
LET STORE$(0)=Q$
3070 LET ROWS=1
3080 FOR I=1 TO 20: LET STORE$(I)="":NEX
T
3090 LET STORE$(ROWS)=TUNE$(0)+","+TUNE$
(1)+";"
3100 FOR I=2 TO N-1
3110 IF LEFT$(TUNE$(I),5)<>TEMP$(0) THEN
 LET STORE$(ROWS)=STORE$(ROWS)+LEFT$(TUN
ES$(I),5):LET TEMP$(0)=LEFT$(TUNE$(I),5)
3120 IF MID$(TUNE$(I),6,3)<>TEMP$(1) THE
N LET STORE$(ROWS)=STORE$(ROWS)+MID$(TUN
E$(I),6,3):LET TEMP$(1)=MID$(TUNE$(I),6,
3)
3130 IF MID$(TUNE$(I),9,4)<>TEMP$(2) THE
N LET STORE$(ROWS)=STORE$(ROWS)+MID$(TUN
E$(I),9,4):LET TEMP$(2)=MID$(TUNE$(I),9,
4)
3140 LET STORE$(ROWS)=STORE$(ROWS)+MID$(
TUNE$(I),13,2)+";"
3150 IF LEN(STORE$(ROWS))>200 THEN LET R
OWS=ROWS+1
3160 NEXT I
3170 FOR I=1 TO ROWS:PLAY STORE$(I):NEXT
 I
3180 CLS:INPUT "DO YOU WANT TO SAVE (Y/N
):";Q$:IF Q$<>"Y" THEN RETURN
3190 CLS:MOTOR ON:AUDIO ON:INPUT"POSITIO
N TAPE THEN PRESS enter (MOTOR IS ON)'"
;Q$
3200 MOTOR OFF:AUDIO OFF:INPUT "PLACE RE
CORDER IN record MODE  THEN PRESS enter
:";Q$
```

```
3210 MOTOR ON:FOR I=1 TO 10000:NEXT I
3220 OPEN "O",£-1,"PLAY"
3230 PRINT £-1,ROWS
3240 FOR I=0 TO ROWS
3250 PRINT £-1,STORE$(I)
3260 NEXT I
3270 CLOSE £-1
3280 RETURN
```

This module compiles the tunes into an economical format for storage on tape.

Commentary

Lines 3030–3050: The three lines of TEMP$ are used to store the last values of length, octave and volume (in this case those of the first note).

Line 3090: The name of the tune and the first note are stored in STORE$.

Lines 3100–3160: The contents of TUNE$ are added to STORE$. Note values are always added, but other details of a note are only added if they differ from the last value specified, since PLAY works on a default basis — the last value for length, volume and octave governing all following notes until another value is specified.

Lines 3170–3280: The resultant string is PLAYed and the user is given the option of saving it if it is satisfactory.

Testing Module 6.4.7

The module should, after a pause, play the desired tune and give the option of saving it.

MODULE 6.4.8

```
8000 REM**************************
8010 REM PICK UP TUNE
8020 REM**************************
8030 CLEAR 3000:OPEN"I",£-1,"PLAY"
8040 INPUT £-1,ROWS
8050 FOR I=0 TO ROWS
8060 IF EOF(-1) THEN GOTO 8090
8070 INPUT £-1,STORE$(I)
8080 NEXT I
8090 CLOSE £-1
8100 FOR I=1TOROWS:PLAY STORE$(I):NEXT I
```

This module serves no purpose whatsoever in this program but is placed here to give you an example of the kind of module that would be needed to pick a tune from tape and play it. If you have compiled and saved a tune, you can test this module and the last by stopping the program, re-RUNning it to initialise the variables, stopping it again and then entering GOTO 8000 (don't forget to position the tape). The original tune should be played again when loading has finished.

Summary

This program well illustrates the world of difference between a screen-editing approach to data and one relying on response to prompts. Imagine the length of this program if each possibility had to be spelt out in a menu at various stages.

Going Further

1) This program will only really come into its own when, like Artist, it is integrated into your program library as the supplier of material for other programs to work on. Most programs could benefit from the addition of a bit of sound now and then and the memory cost should not be high.

MUSIC: Summary of single-key commands:

Up and Down arrows, A and Q move cursor up and down.

I	places new line before line indicated by cursor.
D	deletes line indicated by cursor.
0-9	changes value of any number cursor is placed over.
L,O, V or N	move cursor immediately to relevant section of note.
P	changes note to pause — it cannot be edited back, only deleted.
M	plays all or part of tune.
T	allows setting of tempo.
. and ;	change to dotted note value and back.
C	compiles tune and gives option of saving.
S	saves tune in form such that it can be reloaded by this program.
H	calls up help function.
ENTER	registers note on the same line as cursor. N.B. when cursor has been moved over a note it can only be removed by pressing ENTER.

6.5 GRAPH

Earlier on, in the chapter on high resolution text, you were promised an example of a program using such text in an efficient manner. This is it.

Apart from that not inconsiderable feature the program is a graph drawing tool, enabling the user to draw line graphs of a variety of data, specifying the units and the set-up of the axes.

MODULE 6.5.1

```
6000 REM*************************
6010 REM DATA FILES
6020 REM*************************
6030 MOTOR ON:AUDIO ON:INPUT "POSITION T
APE THEN PRESS enter  (MOTOR IS ON):";Q$
:MOTOR OFF
6040 PRINT:INPUT "PLACE RECORDER IN CORR
ECT MODE  THEN PRESS enter";Q$
```

```
6050 PRINT:PRINT "FUNCTIONS AVAILABLE:",
"1)SAVE DATA",,"2)LOAD DATA",,"3)LOAD CH
ARACTER SET":INPUT "WHICH DO YOU REQUIRE
:";Q:ON Q GOTO 6070,6200,6270
6060 GOTO 1000
6070 MOTOR ON:FOR I=1 TO 10000:NEXT I
6080 OPEN "O",£-1,"GRAPHS"
6090 PRINT £-1,HOR,VER,LH,LV,HM,VM,HO$,V
E$,BASE,LIMIT,VV
6100 FOR I=0 TO HOR-1
6110 PRINT £-1,G(I)
6120 NEXT I
6130 CLOSE £-1:OPEN "O",£-1,"CHARSET"
6140 PRINT £-1,CI
6150 FOR I=0 TO CI-1
6160 PRINT £-1,CHAR$(I)
6170 NEXT I
6180 CLOSE £-1
6190 GOTO 1000
6200 PCLEAR4:CLEAR 10000:PCLS:PMODE4,1
6210 OPEN "I",£-1,"GRAPHS"
6220 INPUT £-1,HOR,VER,LH,LV,HM,VM,HO$,V
E$,BASE,LIMIT,VV:DIM G(HOR-1):DIM CHAR$(
39)
6230 FOR I=0 TO HOR-1
6240 INPUT £-1,G(I)
6250 NEXT I
6260 CLOSE £-1
6270 OPEN "I",£-1,"CHARSET"
6280 INPUT £-1,CI
6290 FOR I=0 TO CI-1
6300 INPUT £-1,CHAR$(I)
6310 NEXT I
6320 CLOSE £-1
6330 GOTO 1000
```

A standard data file module with the slight addition that it is also capable of loading a character set created by the Dictionary program. Having loaded that character set it is then saved and loaded with any data that is stored on tape subsequently.

MODULE 6.5.2

```
7000 REM**************************
7010 REM FORMAT TITLES
7020 REM**************************
7030 LET P2=14-INT(LEN(F$)/2)
7040 PRINT @ 32*P1+P2,STRING$(LEN(F$)+2,
CHR$(185))
7050 PRINT @ 32*(P1+1)+P2,CHR$(185)+F$+C
HR$(185)
7060 PRINT @ 32*(P1+2)+P2,STRING$(LEN(F$
)+2,CHR$(185))
7070 RETURN
```

A standard title formatting module.

MODULE 6.5.3

```
1000 REM**************************
1010 REM MENU
1020 REM**************************
1030 CLS:LET F$="GRAPH":LET P1=1:GOSUB 7
000
1040 PRINT "COMMANDS AVAILABLE:"
1050 PRINT "     1)SET UP FRAMEWORK
1060 PRINT "     2)INPUT VALUES"
1070 PRINT "     3)DRAW GRAPH"
1080 PRINT "     4)DATA FILES"
1090 PRINT "     5)STOP"
1100 PRINT:INPUT "WHICH DO YOU REQUIRE:"
;Z:CLS
1110 ON Z GOSUB 2000,3000,5000,6000,1140
1130 GOTO 1000
1140 CLS:LET F$="GRAPH":LET P1=1:GOSUB 7
000
```

```
1150 LET F$="PROGRAM TERMINATED":LET P1=
9:GOSUB 7000
1160 END
```

A standard menu module.

MODULE 6.5.4

```
2000 REM****************************
2010 REM SET UP AXES
2020 REM****************************
2030 PCLEAR 4:PMODE 4,1:PCLS:CLEAR 10000
:LET VM=999:LET HM=999:DIM CHAR$(39)
2040 CLS:INPUT "HOW MANY INTERVALS ON TH
E       HORIZONTAL AXIS:";HOR:LET LH=INT
(236/HOR)
2050 INPUT "HOW MANY INTERVALS ON THE
    VERTICAL AXIS:";VER:LET LV=INT(168/V
ER)
2060 PRINT:INPUT "PRESS enter TO VIEW AX
ES, THEN  enter TO RETURN:";Q$
2070 GOSUB 4000
2080 IF INKEY$="" THEN GOTO 2080
2090 CLS:PRINT "MARKERS AT REGULAR INTER
VALS AREHIGHLIGHTED:"
2100 INPUT "PLEASE INPUT GAP FOR HORIZON
TAL HIGHLIGHTING:",HM
2110 INPUT "PLEASE INPUT GAP FOR VERTICA
L HIGHLIGHTING:";VM
2120 PRINT:INPUT "PRESS enter TO VIEW AX
ES, THEN  enter TO RETURN:";Q$
2130 GOSUB 4000
2140 IF INKEY$="" THEN GOTO 2140
2150 CLS:INPUT "IS THAT SATISFACTORY (Y/
N):";Q$:IF Q$="N" THEN GOTO 2000
2160 INPUT "NAME FOR UNITS ON HORIZONTAL
    AXIS:";HO$
2170 PRINT:INPUT "NAME FOR UNITS ON VERT
ICAL       AXIS:";VE$
2180 PRINT:INPUT "MINIMUM VALUE ON THE V
ERTICAL    AXIS:";BASE
2190 PRINT:PRINT "MAXIMUM VALUE REPRESEN
TED BY",VER; INPUT " UNITS VERTICALLY:";
LIMIT
2200 LET VV=(LIMIT-BASE)/VER
2210 DIM G(HOR-1):FOR I=0 TO HOR-1:LET G
(I)=-9999.9:NEXT I:GOTO 1000
```

The function of this module is to allow the user to specify the kind of axes desired and the units to be represented.

Commentary

Line 2030: The two variables VM and HM will ultimately store the pixel interval between highlighting marks to be placed on the axes. They are set to 999 so that the marks are not printed initially.

Lines 2040 – 2050: LV and LH are the pixel lengths of the units on the axes.

Line 2200: VV will be used to plot the vertical position on the axis of any data later entered.

Line 2210: The array G is filled with –9999.9 simply because it is a value unlikely to be in much demand in the entry of data — unlike zero.

MODULE 6.5.5

```
4000 REM**************************
4010 REM DRAW AXES
4020 REM**************************
4030 PCLS:SCREEN 1,1
4050 FOR I=LH TO 236 STEP LH:DRAW "BM"+S
TR$(13+I)+",176;D2;L"+STR$(LH)+";D1;R"+S
TR$(LH)
4060 IF (I/LH)/HM=INT((I/LH)/HM) THEN DR
AW "U6"
4070 NEXT I
4080 FOR I=LV TO 172 STEP LV:DRAW "BM15,
"+STR$(177-I)+";L2;D"+STR$(LV+2)+"L1;U"+
STR$(LV+2)
4090 IF (I/LV)/VM=INT((I/LV)/VM) THEN DR
AW "R6"
4100 NEXT I
4110 RETURN
```

This simple module draws the axes with divisions and highlighting marks specified.

Testing Module 6.5.5

You should now be able to specify the form of the axes and see them displayed.

MODULE 6.5.6

```
3000 REM**************************
3010 REM INPUT DATA
3020 REM**************************
3030 CLS:PRINT "POSITION IN ";HO$;:INPUT
 H1:IF H1>HOR THEN PRINT:PRINT "VALUE OU
TSIDE RANGE SPECIFIED   FOR HORIZONTAL A
XIS.":FOR I=1 TO 5000:NEXT:GOTO 3000
3040 PRINT:PRINT "QUANTITY IN ";VE$;:INP
UT V1:IF V1>LIMIT THEN PRINT:PRINT "VALU
E OUTSIDE RANGE SPECIFIED   FOR VERTICAL
 AXIS.":FOR I=1 TO 5000:NEXT:GOTO 3000
3050 PRINT:INPUT "ARE THESE CORRECT:";Q$
:IF Q$="N" THEN GOTO 3000
3060 IF G(H1-1)<>-9999.9 THEN PRINT:PRIN
T "THAT POSITION IS ALREADY FILLED BY TH
E VALUE";G(H1-1)
3070 IF G(H1-1)<>-9999.9 THEN PRINT:PRIN
T "DO YOU WISH TO REPLACE";G(H1-1);:INPU
T Q$:IF Q$="N" THEN GOTO 3000
3080 LET G(H1-1)=V1
3090 CLS:PRINT @ 7*32,"";:INPUT "ANOTHER
 VALUE:";Q$:IF Q$="Y" THEN GOTO 3000
3100 RETURN
```

This module accepts data under the headings supplied by the user relating to the vertical and horizontal axes. If the data input would overwrite an existing item of data, the user is informed and has the option to cancel the input.

MODULE 6.5.7

```
5000 REM**************************
5010 REM DRAW GRAPH
5020 REM**************************
5030 IF CHAR$(0)="" THEN CLS:PRINT @ 7*3
2,"CHARACTER SET NOT LOADED.":FOR J=1 TO
 5000:NEXT:RETURN
5040 GOSUB 4000
5050 FOR L=0 TO HOR-1:IF G(L)=-9999.9 TH
EN NEXT L:CLS:PRINT @ 7*32,"************
```

```
*NO DATA***************":FOR I=1 TO 5000:NE
XT:RETURN
5060 LET T$=HO$:LET P1=80:LET P2=182:GOS
UB 8000
5070 LET T$=VE$+"     (UNIT"+STR$(INT(VV)
)+")":LET P1=22:LET P2=1:GOSUB 8000
5080 DRAW "BM"+STR$(13+(L+1)*LH)+","+STR
$(177-(G(L)-BASE)/VV*LV)
5090 FOR J=L TO HOR-1
5100 IF G(J)<>-9999.9 THEN DRAW "M"+STR$
(INT(13+(J+1)*LH))+","+STR$(INT(177-(G(J
)-BASE)/VV*LV))
5110 NEXT J
5120 IF INKEY$="" THEN GOTO 5120
5130 RETURN
```

Based on the data contained in the array G and the scaling calculated in Module4, this module draws a simple line graph onto the axes specified by the user.

Testing Module 6.5.7

Lines 5030, 5060 and 5070 should be edited so that REM is inserted at the beginning of each. After this you should be able to input specifications and data and then see a graph drawn onto the axes.

MODULE 6.5.8

```
8000 REM*************************
8010 REM HIGH RESOLUTION TEXT
8020 REM*************************
8030 LET GR$="ABCDEFGHIJKLMNOPQRSTUVWXYZ
 1234567890()"
8040 FOR I=1 TO LEN (T$)
8050 FOR J=1 TO LEN(GR$)
8060 IF MID$(T$,I,1)=MID$(GR$,J,1)THEN G
OTO 8090
8070 NEXT J
8080 CLS:PRINT "GRAPHICS SYMBOL NOT CATE
RED FOR:";MID$(T$,I,1):FOR J=1 TO 5000:N
EXT:STOP
8090 DRAW "BM"+STR$(P1+6*(I-1))+","+STR$
(P2)+CHAR$(J-1)
8100 NEXT I
8110 RETURN
```

Apart from the extra data file requirement, this module is all that is necessary for practical handling of text in high resolution graphics modes. The printing of the text is not as fast as normal printing but it is acceptable for labelling and other limited text purposes. The quality of the lettering will depend on the quality of what you have created with the high resolution text programs.

Commentary

Line 8030: This string is a list of text characters in the same order as they are to be found in the character set contained in the array CHAR$.

Line 8040: T$ is the name of a string which is declared when the module is called up and which is to be printed.

Lines 8050–8070: This loop compares letters in T$ with those in GR$ and, when they are discovered in GR$, executes the DRAWing of the corresponding character from CHAR$. It does not actually matter that the characters in GR$ are the same as the characters in CHAR$, as long as the user knows what the GR$ characters are meant to indicate e.g. if the first character in CHAR$ were a * then specifying A in T$ would result in the printing of an asterisk.

Line 8090: PI and P2 are the X and Y co-ordinates to start drawing.

Testing Module 6.5.8

All that should be necessary to unveil high resolution text on your Dragon is to remove the temporary REMs from the beginning of lines 5030, 5060 and 5070. Note that you must first enter your data and then call up the character set from tape. In a program with a separate initialisation module, this would not be necessary, it is just that in this program the variables are all reset when a new framework is specified for a graph.

Summary

One day there will no doubt be a version of the Dragon which will not need to go to these ridiculous lengths to provide such a desirable facility as high resolution text. Even when it comes, however, it will lack something else that you would have liked to see. Perhaps with applications like this one behind you, you will be emboldened to believe that if Dragon haven't provided it there's no reason why you shouldn't do it yourself!

CHAPTER 7
Fun and Games

You will, perhaps, already have gathered from the overall form of this book that I do not consider games the be-all and end-all of home computing. My suspicion is that games are often the fall-back of those who have discovered the fascination of computing but not yet explored the ways in which the power of the micro can enhance their daily living.

Nevertheless, games have their place, depending on the games themselves. Too many magazines and books contain examples of extremely boring games which no one would ever think of playing for pleasure were it not for the fact that they have now been put onto a computer. Personally, I like computer games that are irritatingly difficult and that never let you leave the machine with the feeling that you have absolutely conquered them. Here are three of my favourites.

7.1 TRACKER

This game is infuriating. It will have you questioning the correct functioning of either the program or of your Dragon in next to no time. So sure am I of this that I have even included a line in the game which gives away the answer, so that you can play a couple of times and prove that the whole thing is working properly!

MODULE 7.1.1

```
8000 REM****************************
8010 REM INSTRUCTIONS
8020 REM****************************
8030 CLS:PRINT @ 10,"instructions"
8040 PRINT:PRINT "THIS IS A HUNTING GAME
."
8050 PRINT:PRINT "THE HUNTING GROUND IS 
A 12 BY","30 GRID.":PRINT:PRINT "THE QUA
RRY IS INVISIBLE."
8060 PRINT:PRINT "EACH TURN,THE QUARRY M
AKES A     SECRET MOVE. THIS MOVE DOES NO
T CHANGE DURING A PARTICULAR HUNT."
8070 PRINT "THE MOVE CAN BE UP TO SIX SP
ACESUP OR DOWN AND SIX SPACES LEFT  OR R
IGHT.";:INPUT "     <enter FOR MORE>";Q$
8080 CLS:PRINT "EACH TURN CONSISTS OF:"
8090 PRINT:PRINT "1> AN INVITATION TO IN
PUT YOUR  ESTIMATE OF THE POSITION OF TH
E QUARRY."
8100 PRINT:PRINT "2> AN 'O' WILL APPEAR 
IN YOUR   CHOSEN SQUARE. A '+' IN AN   
  ADJACENT SQUARE WILL INDICATE   THE DI
RECTION OF THE QUARRY."
8110 PRINT:PRINT "3> THE QUARRY WILL MOV
E."
8120 PRINT:INPUT "enter FOR MORE:";Q$
8130 CLS:PRINT "AT THE START OF EACH TUR
```

```
N YOU   HAVE THE OPPORTUNITY TO REVIEW
THE HUNT SO FAR."
8140 PRINT:PRINT "THIS IS DONE BY ENTERI
NG ZERO   WHEN THE DOWN CO-ORDINATE IS
  CALLED FOR."
8150 PRINT:PRINT "YOU CAN START THE REVI
EW AT ANY PREVIOUS MOVE  BUT YOU ARE
  LIMITED TO REVIEWING 20 MOVES INANY ON
E HUNT."
8160 PRINT:PRINT "THE 20 REVIEWS CAN BE
TAKEN ALL AT ONCE OR IN BATCHES.":INPUT
 "  (enter)";Q$
8170 RETURN
```

Instructions for the game.

MODULE 7.1.2

```
9000 REM***************************
9010 REM SET DIFFICULTY
9020 REM***************************
9030 PRINT "THERE IS A DIFFICULTY FACTOR
    BUILT INTO THE GAME."
9040 PRINT:PRINT "THIS CONSISTS OF A RAN
DOM MOVE  OF UP TO 6 DOWN AND 6 RIGHT
  EVERY SO OFTEN. YOU ARE NOTIFIEDWHEN A
 RANDOM MOVE TAKES PLACE."
9050 PRINT:PRINT "THE DIFFICULTY FACTOR
RANGES     0 TO 10."
9060 PRINT:PRINT "'0' MEANS NO RANDOM MO
VES."
9070 PRINT:INPUT "PLEASE INPUT YOUR DESI
RED DIFFICULTY FACTOR:";E
9080 LET E=(11-E)*2+2-100*(E=0):RETURN
```

This module sets a difficulty factor as explained in the module itself.

MODULE 7.1 .3

```
1000 REM****************************
1010 REM INITIALISE
1020 REM****************************
1030 CLS:PRINT @ 8*32+12,"tracker"
1040 PRINT:INPUT "DO YOU WANT INSTRUCTIO
NS (Y/N):";Q$:IF Q$="Y" THEN GOSUB 8000
1050 CLS:GOSUB 9000
1060 DIM M(99,3)
1070 LET R1=7-RND(13)
1080 LET R2=7-RND(13)
1090 LET P1=RND(12)
1100 LET P2=RND(30)
1110 LET B$="":LET T$=CHR$(159)+CHR$(191
):FOR I=1 TO 15:LET B$=B$+T$:NEXT
1120 LET C$="":LET T$=CHR$(191)+CHR$(159
):FOR I=1 TO 15:LET C$=C$+T$:NEXT
1130 LET T=0:LET C1=0
```

This module sets up the program variables.

Commentary

Lines 1070–1080: Rl and R2 are the vertical and horizontal components of the quarry's secret move each turn.

Lines 1090–11 00: The vertical and horizontal co-ordinates of the quarry's initial position.

Lines 1110–1120: This sets up two lines of red and yellow chequerboard to speed later printing.

MODULE 7.1.4

```
2000 REM***************************
2010 REM INITIAL BOARD
2020 REM***************************
2030 CLS
2040 LET A$="1234567890123456789012345678
890":PRINT "  ";A$
2050 FOR I=1 TO 12:PRINT @ I*32,MID$(A$,
I,1):NEXT:GOSUB 4000:IF A=1 THEN RETURN
```

This module simply prints a grid of numbers on the edge of the chequerboard (which has not yet been printed).

MODULE 7.1.5

```
4000 REM***************************
4010 REM PRINT BOARD AND MOVE
4020 REM***************************
4030 FOR I=1 TO 6:PRINT @ (I*2-1)*32+1,B
$:PRINT @ I*2*32+1,C$:NEXT I
4040 FOR I=13 TO 15:PRINT @ I*32,STRING$
(31," ");:NEXT
4050 IF T=0 THEN RETURN
4060 POKE 1024+32*M1+M2,ASC("O"):POKE 10
24+32*(M1-(P1>M1)+(P1<M1))+(M2-(P2>M2)+(
P2<M2)),ASC("+")
4070 PRINT @15*32,P1;"   ";P2;
4080 IF A<>1 THEN RETURN
4090 RETURN
```

This module prints the board itself.

Commentary

Line 4060: This line pokes the 0 into the position guessed by the player and, using the value of conditions, a + in the direction of the quarry. Note the way the ASC value of a character is poked directly to the screen though since the chip which controls the display works on a slightly different character set than the Dragon's Basic the + is inverted.

Line 4070: This line actually tells the player where the quarry was when the 0+ clue was formulated. It should be removed when you begin to play seriously.

MODULE 7.1.6

```
5000 REM***************************
5010 REM MOVE INCREMENT
5020 REM***************************
5030 LET P1=P1+R1:LET P2=P2+R2
5040 IF T/E=INT(T/E) THEN LET P1=P1+RND(
6):LET P2=P2+RND(6):PRINT @ 15*32,"rando
m move";
5050 LET P1=P1-12*(P1<1)+12*(P1>12)
5060 LET P2=P2-30*(P2<1)+30*(P2>30)
5070 RETURN
```

This module adds the secret move and calculates whether the quarry has moved off one side of the board or the other. Depending on the difficulty factor, the module also assesses whether it is time for a random move.

MODULE 7.1.7

```
3000 REM****************************
3010 REM INPUT AND DIRECTIONS
3020 REM****************************
3030 LET T=T+1:IF T>100 THEN CLS:PRINT @
 7*32,"SORRY-CAN'T TAKE ANY MORE MOVES."
,,"YOU'RE JUST SO BAD IT'S PAINFUL!":END
3040 LET Q$=""
3050 LET M$="MOVE"+STR$(T):FOR I=1 TO LE
N(M$):PRINT @ (I+2)*32+31,MID$(M$,I,1);:
NEXT
3060 PRINT @ 15*32+15,"<'0' FOR REVIEW>"
;
3070 PRINT @ 13*32,"";:INPUT "DOWN:";M1
3080 IF M1>12 OR M1<0 THEN PRINT">OUT OF
 RANGE":PRINT @ 13*32,"":GOTO 3050
3090 IF M1=0 THEN GOSUB 7000:GOTO 3050
3100 PRINT @ 15*32,STRING$(31," ");
3110 PRINT @ 14*32,"";:INPUT "ACROSS:";M
2
3120 IF M2>30 OR M2<1 THEN PRINT ">OUT O
F RANGE";:PRINT @ 14*32,"":GOTO 3110
3130 PRINT @ 15*32,STRING$(31," ");
3140 IF M1=P1 AND M2=P2 THEN GOTO 6000
3150 LET M(T-1,0)=M1:LET M(T-1,1)=M2
3160 LET M(T-1,2)=P1:LET M(T-1,3)=P2
3170 GOSUB 4000:GOSUB 5000:GOTO 3030
```

This module accepts the player's guess as to the current position of the quarry and stores it, together with the quarry's co-ordinates, in the array M. It obviously also checks to determine whether the player has actually caught the quarry.

MODULE 7.1.8

```
6000 REM****************************
6010 REM SUCCESS AT LAST
6020 REM****************************
6030 PRINT @ 7*32,STRING$(31," ");:PRINT
 @ 7*32+10,"GOT IT!":FOR I=1 TO 5000:NEX
T
6040 PRINT @ 7*32+6,"";:INPUT "ANOTHER G
AME (Y/N)";Q$:IF Q$="Y" THEN GOTO 1070
6050 END
```

This module informs the player that the game is won and allows a restart if desired.

MODULE 7.1.9

```
7000 REM****************************
7010 REM REVIEW OF GAME
7020 REM****************************
7030 LET A=1:IF C1>20 THEN GOTO 7110
7040 CLS:PRINT @ 4*32+10,"review"
7050 PRINT:PRINT "REVIEW ALLOWANCE 20 MO
VES."
7060 PRINT:PRINT "YOU HAVE USED";C1
7070 PRINT:PRINT "LAST MOVE WAS NO.";MID
$(STR$(T),2)
7080 PRINT:INPUT "INPUT FIRST MOVE FOR R
EVIEW:";T1
7090 CLS
7100 FOR J=T1-1 TO T-2:LET C1=C1+1
7110 IF C1>20 THEN CLS:PRINT @ 10*32+4,"
```

```
review rights exhausted":FOR I=1 TO 3000
 :NEXT:GOTO 7190
7120 LET M1=M(J,0):LET M2=M(J,1):LET P1=
M(J,2):LET P2=M(J,3)
7130 GOSUB 2000
7140 PRINT @ 13*32,"REVIEW OF MOVE";J+1
7150 IF (J+1)/E=INT((J+1)/E) THEN PRINT
@ 15*32,"random move followed";
7160 PRINT @ 14*32,"";:INPUT "enter=NEXT
 MOVE/'0'=QUIT:";Q$
7170 FOR I=13 TO 15:PRINT @ I*32,STRING$
(31," ");:NEXT
7180 IF Q$<>"0" THEN NEXT J
7190 LET M1=M(T-2,0):LET M2=M(T-2,1):LET
 P1=M(T-2,2):LET P2=M(T-2,3)
7200 GOSUB 2000:LET A=0:RETURN
```

This is the module which allows the player to review previous moves. It uses previous modules to draw the board but sets the indicator variable A to zero so that moves will not be input.

Going Further

1) One definite improvement would be a facility which, at the end of the game — either successful or otherwise, allowed the player not only to review the moves made but also to see the actual position of the quarry. Since this information is stored in the array M, there should be little difficulty in adding such a module.

7.2 HEADLONG

If you thought that one was bad enough, this one is absolutely impossible — at least on the higher levels of difficulty. The object of the game is to steer a moving dot around a cluttered screen without crashing into anything, including the trail left by the dot itself. As an added incentive there is a large white block which rushes across the screen mindlessly and if it collides with you then that is the end of the game. The basis of the game is the Doodle program you were given earlier and two versions are given — one for joysticks (which is the better of the two) and one using only the arrowed keys.

MODULE 7.2.1

```
1000 REM***************************
1010 REM INITIALISE
1020 REM***************************
1030 CLS:INPUT "PLEASE INPUT DIFFICULTY
LEVEL    FROM 1 TO 10:";DIFF
1040 LET P1=100:LET P2=100
1050 PMODE 0,1:PCLS3:SCREEN 1,1
1060 DIM G(15,15),H(15,15)
1070 GET (0,0)-(15,15),G,G
1080 PCLS
1090 DRAW "BM0,0;R255;D191;L255;U191"
1100 DRAW "BM20,0;D100"
1110 DRAW "BM50,191;U75"
1120 DRAW "BM0,60;R15"
1130 DRAW "BM0,120;R40"
1140 DRAW "BM255,97;L100"
1150 DRAW "BM75,150;E100"
1160 DRAW "BM125,191;U75"
1170 DRAW "BM30,40;R120"
1180 DRAW "BM200,0;D80"
1190 DRAW "BM 220,191;U80"
1200 DPAW "BM140,150;R65"
1210 LET SC=0
```

Sets up program variables and draws obstacles on the screen. The moving block is taken, using GET, from the empty screen in line 1070 and stored in the array G.

MODULE 7.2.2 (Joystick Version)

```
2000 REM**************************
2010 REM EXECUTE DRAWING
2020 REM**************************
2030 LET X=2:LET Y=2
2040 FOR I=0 TO 3:LET J(I)=JOYSTK(I):NEX
T
2050 LET X1=X:LET Y1=Y
2060 LET X=X-2*(J(2)>63-DIFF*3)+2*(J(2)<
0+DIFF*3)
2070 LET Y=Y-2*(J(3)>63-DIFF*3)+2*(J(3)<
0+DIFF*3)
2080 IF P1>240 THEN LET P1=0
2090 IF P2>176 THEN LET P2=0
2100 GET (P1,P2)-(P1+15,P2+15),H,G:PUT(P
1,P2)-(P1+15,P2+15),G,PSET
2110 REMFOR I=1 TO 5:NEXT
2120 IF X1<>X OR Y1<>Y THEN IF PPOINT(X,
Y)<>0 THEN GOTO 2230
2130 IF X1<>X OR Y1<>Y THEN LET SC=SC+1
2140 PUT (P1,P2)-(P1+15,P2+15),H,PSET
2150 LET P1=P1+8:LET P2=P2+8
2160 PSET(X,Y,3)
2190 GOTO 2040
```

Most of the module will be familiar from the Doodle program.

Commentary

Line 2050: These variables are used to determine whether the user's dot has moved in this pass through the module (keeping the joystick straight allows the dot to be stationary). If the dot has not moved then no attempt is made to see whether there is an obstacle where the dot is about to be printed.

Lines 2080–2110: These lines move the white block across the screen diagonally.

Line 2130: If the screen point where the dot is about to be printed is set then the game ends.

MODULE 7.2.3

```
2200 REM**************************
2210 REM GAME ENDS
2220 REM**************************
2230 CLS:PRINT @ 7*32+10,"SCORE=";SC:PRI
NT @ 8*32+10,"DIFFICULTY=";DIFF:END
```

Confirms the end of the game and gives the score.

ALTERNATIVE MODULE 7.2.2 (Non-Joystick Version)

```
2000 REM**************************
2010 REM EXECUTE DRAWING
2020 REM**************************
2030 LET X=2:LET Y=2:LET D2=50-DIFF*5
2040 LET T$=INKEY$:IF T$<>"" THEN LET D$
=T$
```

```
2060 LET X=X-2*(D$=CHR$(9))+2*(D$=CHR$(8
))
2070 LET Y=Y-2*(D$=CHR$(10))+2*(D$=CHR$(
94))
2075 FOR I=1 TO D2:NEXT
2080 IF P1>240 THEN LET P1=0
2090 IF P2>176 THEN LET P2=0
2100 GET (P1,P2)-(P1+15,P2+15),H,G:PUT(P
1,P2)-(P1+15,P2+15),G,PSET
2120 IF PPOINT(X,Y)<>0 THEN GOTO 2230
2130 LET SC=SC+1
2140 PUT (P1,P2)-(P1+15,P2+15),H,PSET
2150 LET P1=P1+8:LET P2=P2+8
2160 PSET(X,Y,3)
2190 GOTO 2040
```

Joystick functions are here replaced with INKEY$. The dot is never stationary but continues moving until a new direction is input at one of the arrowed keys.

Going Further

Enjoyable though it certainly is, the simple structure of this game gives ample scope for all kinds of added features — not the least of which might be a two player version.

7.3 QUOITS

A game of judgment and reactions and also a program that well displays some of the strengths and weaknesses of the GET and PUT commands.

MODULE 7.3.1

```
9000 REM*************************
9010 REM INSTRUCTIONS
9020 REM*************************
9030 CLS:PRINT "YOU ARE INVITED TO JOIN
A GAME  OF THREE DIMENSIONAL QUOITS."
9040 PRINT:PRINT "IT'S CALLED THREE DIME
NSIONAL   BECAUSE THE QUOITS HAVE TO BE
  THROWN FROM A HEIGHT ONTO        PEDEST
ALS THAT VARY IN HEIGHT   FROM ONE TO FI
VE FEET."
9050 PRINT:PRINT "YOU HAVE FOURTY QUOITS
 AND YOUR OBJECT IS TO GET ONTO AS MANY
OFTHE LOW PEDESTALS AS POSSIBLE."
9060 PRINT:INPUT "<enter> TO CONTINUE:";
Q$
9070 CLS:PRINT "THE HEIGHT OF THE PEDEST
ALS IS  INDICATED BY A NUMBER AS ON THE
FACE OF A DICE."
9080 PRINT:PRINT "EACH TIME YOU LAND ON
A PEDESTALYOU SCORE SIX MINUS THE HEIGHT
--PROVIDED THAT YOU HAVEN'T HIT   THAT O
NE BEFORE."
9090 PRINT:PRINT "THE GAME IS ENTIRELY W
ITHOUT    TEXT. YOUR ONLY AIDS ARE TWO
  INSTRUMENT DISPLAYS."
9100 PRINT:INPUT "<enter> TO CONTINUE:";
Q$
9110 CLS:PRINT "THE FIRST INSTRUMENT IS
A LINE  WHICH RACES ROUND THE SQUARE ON
WHICH YOU PLAY. PRESSING A KEY  WILL STO
P IT AND WHEREVER IT    STOPS, THAT IS T
HE DIRECTION IN WHICH YOUR QUOIT WILL TR
AVEL    YOUR PLATFORM IN THE CENTRE OF
GRID."
9120 PRINT:PRINT "THE SECOND INSTRUMENT
IS A POWERINDICATOR. IT IS ALSO A LINE B
UTACROSS THE TOP OF THE SCREEN.   THE FU
RTHER YOU LET IT GO. THE  HARDER YOUR TH
ROW."
9130 INPUT "<enter> TO CONTINUE:";Q$
9140 CLS: PRINT "AFTER YOUR HAVE THROWN
```

```
THE QUOITIT STARTS TO FALL. YOU CAN TRAC
KITS FALL ON THE HEIGHT LINE AT  THE RIG
HT OF THE SCREEN--IT ALSOINDICATES THE H
EIGHTS OF THE     FIVE TYPES OF PEDESTAL.
"
9150 PRINT:PRINT "IF IT'S TOO LOW WHEN I
T GETS TO A PEDESTAL, THE QUOIT IS WASTE
D."
9160 PRINT:INPUT "<enter> TO CONTINUE:";
Q$
9170 CLS:PRINT "ONE FINAL PROBLEM. YOUR
PLATFORMIS CONSTANTLY DESCENDING, MAKING
IT INCREASINGLY DIFFICULT TO HITTHE LOWE
R PEDESTALS.               GOOD LUCK"
9180 PRINT:INPUT "<enter> TO START:";Q$
9190 RETURN
```

Instructions for the game.

MODULE 7.3.2

```
2000 REM***************************
2010 REM INITIALISE
2020 REM***************************
2030 CLEAR:PCLEAR4
2040 DIM A(8,10)
2050 FOR I=0 TO 8:FOR J=0 TO 10
2060 LET A(I,J)=RND(5)
2070 NEXT J,I
2080 LET X2=6:LET Y2=5
2090 LET SCORE=0:LET A(4,5)=0
2100 GOTO 1060
```

This module sets up the program variables, especially the values in the array A which determine the heights of the pedestals.

MODULE 7.3.3

```
6000 REM***************************
6010 REM CHIMNEY SQUARES
6020 REM***************************
6030 PCLS:PMODE4
6040 DIM B1(15,15):DIM B2(15,15):DIM B3(
15,15):DIM B4(15,15):DIM B5(15,15)
6050 DIM B$(3):LET A$="BM1,1;R15;D15;L15
;U15;"
6060 LET B$(1)=A$+"BR7;BD7;R1;D1;L1"
6070 LET B$(2)=A$+"BR3;BD3;R1;D1;L1;BD7;
BR8;R1;D1;L1"
6080 LET B$(3)=A$+"BR11;BD3;R1;D1;L1;BD7
;BL8;R1;D1;L1"
6090 DRAW B$(1):GET (1,1)-(16,16),B1,G
6100 DRAW B$(2):GET (1,1)-(16,16),B3,G
6110 DRAW B$(3):GET (1,1)-(16,16),B5,G
6120 PCLS:DRAW B$(2):GET (1,1)-(16,16),B
2,G
6130 DRAW B$(3):GET (1,1)-(16,16),B4,G
6140 RETURN
```

Using DRAW instructions which provide a 1,2 and 3 this module DRAWs the five dice faces and then GETs them into the 5 arrays B1-B5. Unfortunately a three-dimensional array such as B(4,15,15) cannot be used since GET and PUT will not work with such an array.

MODULE 7.3.4

```
7000 REM***************************
7010 REM PRINT CHIMNEYS
7020 REM***************************
7030 PMODE4:PCLS:SCREEN 1,1
```

```
7040 FOR I=24 TO 152 STEP 16:FOR J=32 TO
 192 STEP 16
7050 IF I=88 AND J=88 THEN GOTO 7090
7060 LET I1=(I-24)/16:LET J1=(J-32)/16
7070 ON ABS(A(I1,J1)) GOSUB 7140,7150,71
60,7170,7180
7080 IF A(I1,J1)<0 THEN PUT(J,I)-(J+15,I
+15),B1,NOT
7090 NEXT J,I
7100 DRAW "BM250,191":FORI=1 TO 6:DRAW "
R5;U1;L5;BM+0,-10":NEXT I
7110 DRAW "BM5,186":FOR I=1 TO (H-100)/5
+3-G:DRAW "D2;R2;U2;L2BM+5,+0":NEXT I
7120 DRAW "BM8,0;D5;R1;U5;BM240,0;D5;R1;
U5"
7130 RETURN
7140 PUT(J,I)-(J+15,I+15),B1,PSET:RETURN
7150 PUT(J,I)-(J+15,I+15),B2,PSET:RETURN
7160 PUT(J,I)-(J+15,I+15),B3,PSET:RETURN
7170 PUT(J,I)-(J+15,I+15),B4,PSET:RETURN
7180 PUT(J,I)-(J+15,I+15),B5,PSET:RETURN
```

This module DRAWs the display on which the game is played.

Commentary

Lines 7040–7090: These loops work through the array A, using the values contained in it to call up the one line sub-routines at line 7140–7180.
Line 7100: This draws the marks which indicate the height of the pedestals on the right of the screen.

Line 7110: The remaining quoits are displayed visually at the bottom of the screen.

Testing Module 7.3.4

Temporarily removing line 1060 and line 6140 should result in a program which will create the display described, before giving an error RETURN WITHOUT GOSUB.

MODULE 7.3.5

```
1000 REM*************************
1010 REM MAIN PROGRAMME
1020 REM*************************
1030 CLS:PRINT @ 2*32+12,"quoits"
1040 PRINT:INPUT "DO YOU WANT INSTRUCTIO
NS (Y/N):";Q$:IF Q$="Y" THEN GOSUB 9000
1050 CLS:GOTO 2000
1060 GOSUB 6000
1070 FOR H=300 TO 100 STEP -10:FOR G=1 T
O 2
1080 GOSUB 7000:GOSUB 3000
1090 IF INKEY$="" THEN GOTO 1090
1100 CLS:NEXT G,H
1110 CLS:PRINT @ 2*32+12,"QUOITS"
1120 PRINT:PRINT "YOUR SCORE WAS ";SCORE
:END
```

The main loop of the program, which allocates work among the other modules.

MODULE 7.3.6

```
3000 REM*************************
3010 REM DIRECTION
3020 REM*************************
3030 PMODE 4:SCREEN1,1
```

```
3040 LET S1=119:LET S2=96
3050 LET X=16:FOR Y=175 TO 17 STEP -1
3060 PSET (X,Y)
3070 IF INKEY$<>"" THEN GOTO 4000
3080 NEXT Y
3090 FOR X=16 TO 222
3100 PSET (X,Y)
3110 IF INKEY$<>"" THEN GOTO 4000
3120 NEXT X
3130 FOR Y=16 TO 174
3140 PSET (X,Y)
3150 IF INKEY$<>"" THEN GOTO 4000
3160 NEXT Y
3170 FOR X=223 TO 16 STEP -1
3180 PSET (X,Y)
3190 IF INKEY$<>"" THEN GOTO 4000
3200 NEXT X
```

This module draws the direction indicator line around the screen. Stopping it supplies two co-ordinates, X and Y to the next module.

MODULE 7.3.7

```
4000 REM**************************
4010 REM ANGLE AND VELOCITY
4020 REM**************************
4030 FOR I=1 TO 500:NEXT I
4040 H1=X-S1:V1=Y-S2
4050 IF ABS (V1)>=ABS(H1) THEN LET V2=SG
N(V1):LET H2=ABS(H1/V1)*SGN(H1)
4060 IF ABS(H1)>ABS(V1) THEN LET H2=SGN(
H1):LET V2=ABS(V1/H1)*SGN(V1)
4070 FOR V=8 TO 240
4080 PSET (V,3):IF INKEY$<>"" THEN GOTO
5000
4090 NEXT V
4100 RETURN
```

Based on the co-ordinates at which the direction indicator stopped, this module calculates a direction for the throw from the centre of the grid.

Commentary

Lines 4050–4060: These two lines serve to ensure that the path of the quoit, when it is plotted on the grid, will be a continuous line rather than spaced pixels. It does this by determining which is the greater, the horizontal or the vertical component of the direction and then using that component as the basis for the line, with adjustments up, down, left or right for the other component.

Lines 4070–4090: This draws the strength indicator.

MODULE 7.3.8

```
5000 REM**************************
5010 REM PLOT COURSE
5020 REM**************************
5030 LET T=(255-V)/1000
5040 FOR I=1 TO 100
5050 LET X=INT(S1+INT(I*H2)):LET Y=INT(S
2+INT(I*V2))
5060 IF Y<0 OR Y>191 THEN RETURN
5070 IF PPOINT(X,Y)<>0 THEN PRESET(X,Y)
ELSE PSET(X,Y)
5080 GOSUB 8000
5090 LET H3=H-5*(T*I)^2:IF H3<=0 THEN RE
TURN
5100 IF X1=X2 AND Y1=Y2 AND X1<>999 AND
Y1<>999 THEN IF H3<ABS(A(Y1,X1))*20 AND
```

```
A(Y1,X1)>0 THEN LET SCORE=SCORE+6-A(Y1,X
1):LET A(Y1,X1)=A(Y1,X1)*-1
5110 IF X1<>999 AND Y1<>999 THEN IF H3<A
BS(A(Y1,X1))*20 THEN RETURN
5120 LET Y2=Y1:LET X2=X1
5130 IF H3/2<=191 THEN LINE(252,0)-(252,
191-(H3/2)),PSET
5140 NEXT I:RETURN
```

This module calculates the track of the quoit across the grid and the height of the quoit as it falls.

Commentary

Line 5070: The pixel over which the quoit is currently passing is reversed, no matter whether set or not.

Line 5090: This formula describes a falling trajectory.

Line 5100: This line ensures that if the quoit enters a square for which it does not have sufficient height, it is not registered as a landing, nor if the square has been landed on before — only if it is a fresh square and it is entered from above is a landing registered.

Line 5130: This line draws the height indicator — a downward line, on the right of the screen.

MODULE 7.3.9

```
8000 REM****************************
8010 REM LANDING?
8020 REM****************************
8030 LET X1=INT((X-31)/16)
8040 LET Y1=INT((Y-23)/16)
8050 IF X1>10 OR X1<0 THEN LET X1=999
8060 IF Y1>8 OR Y1<0 THEN LET Y1=999
8070 RETURN
```

This module transforms the pixel co-ordinates of the quoit into co-ordinates on the 11*9 grid. If the quoit has passed beyond the grid the value of the relevant co-ordinate is set to 999 as an indicator for the previous module.

POSTWORD

I hope that it has not escaped your notice, now that you have come to the end of this book, that you are the possessor of a library of programs. True, it is not the most extensive library in the history of computing, but it contains the tools to tackle a variety of tasks if you are prepared to adapt the programs to your own specific needs. In addition the collection as a whole may have given you a glimpse of what the Dragon is waiting to achieve with and for you.

Like the simplest camera, the meanest computer is far better than its programmers, always capable of more than has yet been done with it. The Dragon is a highly powered tool and it waits only for you to take programs like these and make them your own, cannibalise them for spares, discard them on the path to better things.

In other words the Dragon waits only to be put to work.

The Working Dragon 32 is based on a collection of solid, sophisticated programs in areas such as data storage, finance, graphics, household management, education and games of skill.

Some of the more advanced programs include a Text Editor, which can perform many of the functions of a word processor, and a Music Editor, which will let you write long music programs without endlessly repeating similar routines.

Each of the programs is explained in detail, line by line. And each of the programs is built up out of general purpose subroutines and modules which, once understood, can form the basis of any other programs you need to write.

Advanced programming skills spring out of the discussion explaining each subroutine. The collection also leaves you with a wide range of practical applications programs which might otherwise only be available on cassette.

The author, David Lawrence, has also written *The Working Spectrum* and is a regular contributor to *Popular Computing Weekly.*

Sunshine Books

£5.95 net

ISBN 0 946408 01 7

The Working Dragon 32 David Lawrence Sunshine Books