the dragon trainer

a handbook for beginners

brian lloyd

the dragon trainer

a handbook for beginners

brian lloyd

First published 1983 by:
Sunshine Books
(An imprint of Scot Press Ltd.)
12–13 Little Newport Street,
London WC2R 3LD

ISBN 0 946408 09 2

Cover design by Graphic Design Ltd.
Illustration by Richard Dunn.
Typeset and printed in England by Commercial Colour Press, London E7.

This book is dedicated to my father for all his help, patience and encouragement

CONTENTS

Introduction

In compiling this book we assumed that the reader had little or no knowledge of computer programming. Each section has been tested by complete novices and re-written if proved to be too complicated. My thanks to the most ardent tester, Ray Hunt-Terry, who has read through the book word for word and tested each section from cover to cover.

The book is designed to be read from the first page right through to the last, *without* skipping any pages. Even skipping a single line could leave you totally confused and wondering how something works, so don't be in too much of a hurry to do things.

The book works through the commands in the order in which you need to know them so that you can get down to writing simple programs as soon as possible. As you read each section your understanding of the computer's commands and functions will grow, enabling you to write more complicated programs as you learn each new command.

BASIC is the main language of the Dragon 32 computer and you may be assured to know that BASIC stands for BEGINNER'S All-purpose Symbolic Instruction Code. The fact that the language is designed for beginners doesn't stop it from being a very powerful language giving you immense control over the computer.

If you read each section thoroughly and make sure you understand what you have read, the technique of computer programming will come to you quite quickly. It is best to make sure that you understand each section, even if you have to go through it several times. It saves a lot of time and frustration in the long run. Finally, never be afraid to experiment with any command, for experimenting is the best way to learn (you may even discover something that no-one else knows about).

1
Getting started

If you haven't used a computer before then you will probably be wondering what to do with this plastic box with keys on top. Well the first thing to do is to connect up your computer.

Plug the TV lead that came with the Dragon into your TV aerial socket and the other end into the socket marked TV on your computer.

Next the transformer. Plug the lead with a white wire and grey plug on the end into the socket marked TRANS SUPPLY at the rear of the computer. Plug in the power lead and switch on the computer by pressing the button marked ON/OFF which is also on the rear of the computer.

Switch on your television and tune it to approximately channel 36 until you see the message DRAGON DATA Ltd displayed clearly on the screen.

Now that you have your Dragon set up and working you will see a flashing dark green/black square. This square is the cursor. Its job is to show you where the next character that you type into the keyboard will appear on the screen.

If you try typing on the keyboard, the characters you type will appear on the screen with the cursor moving along in front of them.

Some of the keys have two characters on them. To obtain the top characters you will have to hold down one of the SHIFT keys while typing a key ie pressing SHIFT and 5 will result in the % sign appearing on the TV screen. Pressing SHIFT and 8 will produce the closed bracket etc.

If you now press the 0 key (the 0 key is a zero, not to be confused with the letter O) while holding down the SHIFT key and then typing on the keyboard (after releasing the SHIFT and 0 keys) the characters will change to green on black.

The green on black characters are known as INVERSE CHARACTERS and if you connect a printer to your Dragon these will appear as lower-case letters — abcd etc.

To return to upper-case characters you will have to press 0 and SHIFT again. All commands *must* be entered in upper-case. Try typing in some text using upper-case and lower-case characters, ie:

```
Hello, how are you Jill?
```

There are four keys with arrows pointing in different directions:

Pressing the arrow key pointing to the left makes the cursor move backwards, deleting the characters to its left as it goes. This key is used for correcting any mistakes you make while entering text. Try typing in some characters and then using the left-arrow key to delete them. As you will see it is much easier than writing on paper as mistakes can easily be corrected. Later we will demonstrate how to correct mistakes that you have made within a program without re-typing the whole line.

If you press SHIFT and the left-arrow at the same time the whole line will be deleted (but only one line, a maximum of 32 characters).

The up-arrow key produces an arrow pointing upwards when pressed. This key is used in calculations and its full use will be explained later.

Pressing the right-arrow key on its own produces nothing, but SHIFT right-arrow produces a closed square bracket.

The down-arrow key and SHIFT produces an open square bracket.

Pressing SHIFT and up-arrow produces an arrow pointing to the left.

The long bar at the bottom of the keyboard is the space bar, and as its name suggests, inserts a space each time it is pressed.

Try typing the above key combinations and see what effects they produce.

Another key on the keyboard is the CLEAR key. If you press this key the screen will clear and all the characters on it will be lost (unless they are part of a program). The cursor is also moved to the top left-hand corner of the screen ready for more text to be typed in.

Next to the CLEAR key is a key with the word ENTER printed on it. The use of this key will be explained later, as will the use of the key in the top right-hand corner the BREAK key, so do not worry too much about them at the moment as no harm can be done by pressing them accidentally.

2
Time to get working

If you have practised the last chapter you should have a good idea how the keyboard works, so now it's time to make your Dragon work for you. Let's see what it can do.

One of the things a computer can do is calculate, so let's try some calculations. To do calculations on the Dragon you will need to use one of the most common commands used in BASIC and that is PRINT.

There are several ways of using the PRINT command, but for the moment we will use just one. Remember to use the left-arrow key if you make any mistakes:

```
PRINT 5+19
```

You will need to hold the SHIFT key down to get the + sign, as you will with all the upper characters on the keys.

Now what has happened? Not much! This is because the Dragon will not carry out any command until the ENTER key is pressed. This tells the computer that you have finished entering your instructions and that you want them carried out. So let's press ENTER and see what has happened. If you have entered everything correctly then you should have the answer 24 on your screen.

If ?SN ERROR is on the screen instead you have made a mistake. Press CLEAR to give you a clean screen and try again.

What you have told your Dragon to do is PRINT (or display) on your screen the sum of 5 + 19. Now try the following:

```
PRINT 10+4
PRINT 6*4
PRINT 12-2
PRINT 24/6
```

Try some more calculations of your own, but just one thing before you do. To divide you must use the / sign and to multiply the * sign. This is common to all computers. The minus sign is situated beside the BREAK key.

If you have done some multiple calculations you may not have got the answer you expected. Try the following:

PRINT 6+9/3 (followed by ENTER)

The answer the computer should give is 9 and not 5 as you may have expected. This is because the Dragon as with most computers, carries out the division part of the calculation first. Now if you type:

PRINT (6 + 9)/3 (followed by ENTER)

The answer you will get this time will be 5, because putting brackets around part of the calculation tells the computer to work out that part first. The order in which the computer carries out calculations is: multiplication or division followed by addition or subtraction.

We can now go on to find out what the up-arrow is for. Try typing:

PRINT 2↑3 (again followed by ENTER)

You should have 8 printed up on your screen, because the up-arrow means 'to the power of'. The Dragon works out 'to the power of', or exponentiation as it is called, before multiplication or division.

If you want to calculate negative numbers all you have to do is put a minus sign in front of the number eg PRINT − 5 + 3.

3 PRINTing words or letters

The PRINT command is not only used to display the answers to calculations but can also be used to display words and messages. Try typing:

PRINT"HELLO, I AM YOUR DRAGON"

and then press ENTER.

Your Dragon will display the words enclosed in quotation marks ("') on the screen. The quotation marks tell the computer that what is enclosed within them is not a calculation and that is exactly what you want on the screen.

Try some text for yourself, but remember if you do not put quotation marks around the text you will receive an error message.

Anything enclosed in quotation marks be they letters, numbers or symbols will be PRINTed on the screen just as you entered them.

As we progress through this book we will be using the PRINT command to a greater extent. You will be amazed at what calculations together with PRINT can do.

To re-cap what we have learnt so far in this chapter, the PRINT command can be used to display and carry out calculations on your TV/ monitor screen.

It can also be used to display words etc, as long as they are enclosed in quotation marks.

You have to press ENTER before the computer will carry out an instruction.

As it is a lot better to have a clear screen to work with, remember the CLEAR key clears the screen and puts the cursor to the top left-hand corner of the screen. So if the screen starts to look a bit untidy or hard to follow CLEAR the screen before you start the next exercise.

Remember that if you press the CLEAR key all that is on the screen will be lost.

Now for a few tips before we leave this chapter. To save a lot of typing, instead of typing the word PRINT each time you can use the ? sign. For example ? 2+2 will give the same results as PRINT 2+2. As we use the PRINT command more and more this will save a lot of time, so it is worth remembering.

Try practising some of the things we have learnt in this chapter before moving on. It will save time later.

The only way to learn is to practise and experiment, so do try your own experiments as we go along and keep reading back over the chapter if you do not understand anything.

4
Line numbers

Until now your Dragon has carried out your instructions as soon as you have pressed the ENTER key. Obviously this is of no use whatsoever if you are writing a program as the computer forgets what you have told it to do as soon as it has carried out the instructions. So when we are writing programs we need to use *line numbers*.

Line numbers are used to make the computer remember what you are telling it to do. They also allow you to control the sequence in which the instructions are carried out.

Instructions which are entered using line numbers are not carried out immediately, but are saved in the computer's memory until you want them carried out.

Line numbers are usually in increments of ten, allowing you to insert lines in between ones that you have already entered. Try typing in this example program. You should type each line as it is printed below. After each line press ENTER:

```
10 CLS
20 PRINT"HELLO, I'M YOUR DRAGON 32"
30 PRINT"COMPUTER!"
40 PRINT"AND THIS IS A PROGRAM!!!"
```

When you have finished typing in this program check each line and if you see any mistakes you should retype the line that the mistake occurs in.

You will notice that nothing happened after you pressed ENTER at the end of each line except the cursor jumped down to the next line ready for you to enter another line or command. However, if you type RUN (and of course, press the ENTER key) the instructions will be carried out.

The screen should clear (the CLS command does the same as pressing the CLEAR key) and the words enclosed in quotation marks should appear on the screen.

If you get an error message then you will have made a typing mistake. In this case you should type LIST and then press the ENTER key. The program will re-appear on the screen.

The LIST command works at any time, not just if there is an error in the program. Any time you want to look at your program just type LIST.

Let's have a closer look at the program. The first thing that you will notice is the line numbers and the fact that they are in steps of ten.

Line 10 (the first line of the program) clears the screen in the same way as the CLEAR key. Lines 20, 30 and 40 PRINT the message on the screen. You will notice that each time a new PRINT command is used the text starts on a new line. If you now type the following you will see how to continue PRINTing text on the same line using two PRINT commands:

```
50 PRINT"9*9= ";
60 PRINT 9*9
```

When you have finished typing in these two lines type LIST and you will see that your Dragon has added two lines to the end of the program. If you now type RUN the program will be executed, complete with the two new lines you have entered.

Line 50 PRINTs the characters '9*9 = ' and line 60 PRINTs the answer to the calculation.

You will notice that the sum and the answer are PRINTed on the same line. This is due to the semi-colon (;) at the end of line 50. A semi-colon in a PRINT command tells the computer not to start the next lot of PRINTing on a new line but to carry on where it left off.

Try typing the following and then type LIST:

```
15 PRINT"A PROGRAM!!!"
```

You will see that the line you have just typed has been inserted between lines 10 and 20.

The Dragon automatically puts the lines in their numeric order, allowing you to insert lines anywhere in the program.

One thing about line numbers, however, is that you must not use the same line number twice. This is because if the same line number is used the Dragon will delete the first line with that number and replace it with the new one. To illustrate this type in:

```
15 PRINT"THIS LINE HAS CHANGED!"
```

and type LIST. As you can see the old line 15 has been replaced by the new one.

If you want to delete a line from a program you should just type its line number and press ENTER, eg if you type: 15 then line 15 will be deleted from the program. Type LIST to make sure that the line has been deleted.

The next stage up from this simple program is a program using *multi-statement* lines.

Multi-statement lines are the same as the program lines that we have been using up till now except that more than one instruction is on each line.

Try retyping line 10, replacing it with this line:

```
10 CLS:PRINT"THIS IS A MULTI-STATEMENT LINE!"
```

As you can see we have two instructions on this line, the CLS command and the PRINT command, separated by a colon.

You can have as many commands on one line as you like (as long as the line is not over 255 characters long). The only thing you have to remember is to put a colon between each command.

To finish off this section let's type in the following program (don't worry about the lines not being in order, this is the whole idea of the program). But first type NEW to clear the computer's memory:

Line Sort

```
10 CLS
40 PRINT"ALREADY I HAVE SORTED THE LINES OF THIS PRO-
GRAM INTO NUMERIC ORDER."
60 PRINT"CLEVER LITTLE THING AREN'T I?";
50 PRINT"I CAN ALSO TELL YOU THAT";:
PRINT"876*564 = ";876*564
30 PRINT"COMPUTER AND I'M GOING TO
SHOW":PRINT"YOU WHAT I CAN DO."
20 PRINT"*************A*PROGRAM***********";:PRINT"HI
THERE! I'M YOUR DRAGON 32"
```

When you have entered the program type LIST and you will see that the Dragon has put the lines in their right order. Type RUN and the program will be carried out.

Try experimenting with this program and see what it does and how it works. You could even try writing your own program (remembering to type NEW first though).

5
Variables

So far in our programs we have had no need to use a value which may need to be changed, but very often we have to be able to change a value every few lines. For this reason we use variables.

A variable is a value which can be changed. Letters are used to represent these values, for instance, we can tell the computer to store the value 13 in the letter A. Try typing:

```
10 CLS:A=13
20 PRINT A
```

If you type RUN then your Dragon will PRINT the number 13 on the screen. You can make A represent any number of course.

Now add the following lines to your program:

```
30 B=99
40 PRINT B
50 PRINT"13+99=";A+B
```

Now RUN the program and you will see that the computer has also remembered that B = 99.

Line 50 PRINTs a calculation and its answer, replacing the numbers 13 and 99 with A and B respectively. Now add the next two lines to the program:

```
60 C=A+B
70 PRINT C
```

When you RUN the program this time you will see the computer has stored the value of A + B in C.

This is how the last part of the program works:

(I) LOOK TO SEE WHAT NUMBER 'A' REPRESENTS
(2) LOOK TO SEE WHAT NUMBER 'B' REPRESENTS
(3) ADD TOGETHER THE CONTENTS OF 'A' AND 'B' (112)
(4) REMEMBER THAT VARIABLE 'C' NOW HAS THE VALUE OF 'A' + 'B' (112)
(5) PRINT THE NUMBER THAT IS STORED IN 'C' ONTO THE SCREEN

A variable can be any letter from A to Z, or almost any combination of letters and numbers ie A, HELLO, ZH, AZ, A3, Z100 can all be used as variables.

IF, OR, PRINTER and TOP cannot be used as variables because they are either instructions or the first few letters of the variable are instructions.

Variables can be of any length but only the first two letters are recognised ie if the word HELLO is used as a variable only the letters HE are recognised and used as variables.

All variables have a value of zero before you use them, as it is perfectly alright to refer to a variable which has not yet been given a value (eg saying 'A= B' will set the variable 'A' to 0 if the variable 'B' has not been assigned a value).

One last thing. In this chapter we have been assigning values to variables simply by saying 'A= 1'. It is possible to say 'LET A = 1' but this takes up more program space, so if you ever see a program with a command 'LET A = 1' then you can just read it as 'A = 1'.

6
String variables

You may also need to use a variable to represent a letter or letters (string). In this case we use normal letters and add $ symbol to the end ie A$ can be used as a variable to represent a string of letters.

Type NEW to clear the Dragon's memory of the old program and type the following in:

```
10 CLS
20 A$ = "HELLO"
30 NAME$= "FRED"
40 PRINT A$;" ";NAME$
```

RUN this program and you will see that the Dragon has remembered that the variable A$ represents the word 'HELLO' and that the variable NAME$ represents the word 'FRED' in the same way as normal numeric variables.

Although these string variables, as they are known, can contain numbers, normal numeric variables cannot contain letters.

Here is an example of what you can't do with variables:

```
A = "HELLO"
OR = 3
OR$ = "HELLO"
B = A$
B = A$ + 90
```

It would be a good idea if you experimented some more with variables to make sure you understand how they are used.

In the next chapter, which is about the INPUT commands, we will use variables again and write a program to illustrate how to use both types of variables.

7 INPUT

Using the INPUT command it is possible to assign a value to a variable while a program is being executed.

When the INPUT command is used during a program the computer will stop the program, display a question mark and wait for you to enter a number or letters before carrying on with the program. When you answer the prompt what you actually do is assign a value to a variable.

If you are using the INPUT command to ask for a number and a word or series of letters are entered then the computer will display this message:

```
?REDO
```

and wait for you to enter a proper reply, in other words a number. The same thing applies if you answer with a numeric expression (eg 2+2).

If nothing is entered in reply to an INPUT command (ie if the ENTER key is pressed without first entering a reply) then the variable used will be set to zero (if a numeric variable) or emptied (if a string variable).

Input Demonstration

```
10 CLS
20 PRINT"PLEASE TYPE IN ANY NUMBER"
30 INPUT A
40 CLS
50 PRINT"YOU TYPED IN THE NUMBER";A
```

If you RUN this program the message PLEASE TYPE IN ANY NUMBER will be displayed on the screen and a question mark will appear with the cursor in front of it. Nothing further will happen until you type in a number. After pressing ENTER the screen will clear and the computer will tell you what number you have entered.

Make the following alterations to the program:

```
20 PRINT "WHAT IS YOUR NAME";
30 INPUT A$
50 PRINT"HELLO ";A$
```

When RUN, the program will this time wait for you to enter your name and then give you a nice friendly greeting.

Instead of using the PRINT command to ask a question before the INPUT command, it is possible to incorporate the message in the actual INPUT, eg INPUT"WHAT IS YOUR NAME";A$ asks you what your name is and then assigns your name to A$.

Next is a program illustrating all that we have so far learnt about BASIC programming, including the INPUT command (remember to type NEW before you enter the program).

Questionnaire

```
 10 CLS
 20 PRINT"**********QUESTIONAIRE**********";
 30 INPUT"WHAT IS YOUR NAME";NAME$
 40 INPUT"HOW OLD ARE YOU";AGE
 50 INPUT"ARE YOU INTERESTED IN COMPUTERS";
    COMPUTERS$
 60 INPUT"DO YOU LIKE WORK/SCHOOL";WS$
 70 INPUT"CAREER";CAREER$
 80 INPUT"WHAT YEAR IS IT";YEAR
 90 CLS
100 PRINT"NAME:";NAME$
110 PRINT"AGE:";AGE
120 PRINT"INTERESTED IN COMPUTERS?";COMPUTERS$
130 PRINT"HARD WORKING?";WS$
140 PRINT"IN THE YEAR  2000 YOU WILL BE";2000-YEAR+
          AGE
```

When you RUN the program you will be asked a series of questions which you must answer. When you have answered all the questions the screen will clear and you will be presented with a run down of what you are like. The program also tells you how old you will be in the year 2000.

If you list the program you will find that the program will not all fit on the screen at one time. If you want to see the first half of the program you should type LIST-80. When you have finished with the first half just type LIST to see the rest of the program.

As you can see from the program the computer INPUTs several questions and assigns the answers to different variables.

Line 80 then clears the screen before the following lines PRINT the answers to your questions.

Line 150 then calculates how old you will be in the year 2000.

Try adding to the program so that it asks more questions and PRINTs the answers at the end. You could even try writing your own program (in this case remember to type NEW first to clear the memory).

8
LIST

We have already used the LIST command to display our program on the screen, but there is a lot more to the LIST command than we have seen so far. The program in the last chapter was too long to fit onto the screen at one time so we used the command LIST −80. This means 'display all the current program as far as line 80 on the screen'. It is also possible to LIST all the program after line 80 by using LIST 80−. If you want to see a part of the middle of a program then you can use the command LIST 80−100 which tells the computer to display lines 80−100 on the screen (the line numbers we are using here are just examples and can of course be replaced with the lines that you want to see). The maximum number of lines on the screen at one time is 14. Here is a list of variations of the LIST command:

LIST 50−90

(display lines 50 to 90 on the screen).

LIST −50

(display all the program as far as line 50 on the screen).

LIST 120−

(display all the program after line 120 on the screen).

LIST

(display all the program on the screen, scrolling all but the last 14 lines).

If a program will not all fit on the screen at the same time then the computer 'scrolls' the screen, or moves everything on it up one line. To stop this scrolling at any point while LISTing you should press SHIFT and the @ key together. To continue with the LISTing press any other key (except BREAK).

9
RUN, NEW and CLS

RUN

As you know, the RUN command is used to tell the computer to carry out the program which is currently in memory, going through the lines in numeric order unless instructed to do otherwise. The RUN command usually starts executing the program from the first line, but it is possible to tell the computer to start the program from a different point eg RUN 50 tells the Dragon to start executing the program from line 50. The RUN command also clears all the variables before starting the program, so if you don't want the variable cleared, type GOTO and then the line of the program that you want to start at.

NEW

We have already used the NEW command so you should know that its purpose is to clear the Dragon's memory of anything that is in it. The NEW command will delete any program from memory, and once that has happened you can't get the program back, so make sure that you don't want the old program before NEWing it.

CLS

So far we have used the CLS command to clear the screen before we display any text on it, but the command can be used to clear the screen to any one of 9 colours. For example, use CLS 2 to clear the screen to yellow. The numbers for the different colours are:

0–Black	1–Green	2–Yellow	3–Blue
4–Red	5–Buff	6–Cyan	7–Magenta
8–Orange			

10
IF . . . THEN . . . ELSE

One of the most important aspects of a computer is its ability to compare one thing with another. For instance, it will search through a list of names until it finds the one it is looking for, simply by comparing each name in the list with the one it is looking for to see if they are the same.

The IF. . .THEN. . .ELSE structure is used to see if a condition is fulfilled and IF so goes on to carry out a further instruction. Here is an example to illustrate this:

IF A = 1 THEN B = 1

This simple line checks to see if A represents the number 1 and if so it goes on to assign the number 1 to the variable B.

Adding the ELSE command to the structure tells the computer that IF a condition is fulfilled THEN to carry out the next command(s), ELSE if the condition is not fulfilled then carry out a different set of commands.

Here are a few examples of how the IF. . .THEN. . .ELSE structure can be used:

IF A = 5 THEN Z= 10 ELSE Z = 0

(IF the value of A is 5 THEN assign the number 10 to Z otherwise (ELSE) assign the number 0 to Z).

IF Z< > THEN RUN

(IF the value of Z is anything apart from 1 THEN restart the program).

IF DD< 5 THEN DD = 5

(IF the value of DD is less than 5 THEN assign the number 5 to DD).

IF Y> 7 THEN CLS

(IF the value of Y is bigger than 7 THEN clear the screen).

IF S< = 90 THEN T =10

(IF the value of S is less than or equal to 90 THEN assign the number 10 to T).

IF ST> =19 THEN GT=1

(IF the value of ST is bigger than or equal to 19 THEN assign the number 1 to GT).

Here is a program demonstrating the IF...THEN...ELSE structure (remember to type NEW before you start the program).

Intelligence Test

```
 10 CLS
 20 INPUT"TYPE IN ANY NUMBER LESS THAN 10";N
 30 IF N<10 THEN PRINT"WELL DONE!" ELSE PRINT
    IDIOT!"
 40 INPUT"TYPE IN YOUR NAME";NAME$
 50 IF NAME$="JOHN" THEN PRINT"HELLO JOHN!" ELSE
    PRINT "I DON'T KNOW YOU!"
 60 IF N<10 THEN PRINT"YOU'RE QUITE CLEVER ";NAME$
    ELSE PRINT"YOU ARE PRETTY STUPID AREN'T YOU ";
    NAME$
 70 INPUT"TYPE IN THE ANSWER TO 12*12";T
 80 IF T=144 THEN PRINT"WELL DONE" ELSE PRINT
    "TWIT!"
 90 IF T=144 AND N<>10THEN PRINT"YOU'RE A GENIUS "
    ;NAME$
100 IF T=144 AND N>9 THEN PRINT"YOU'RE IMPROVING!"
110 IF T<>144 AND N<10 THEN PRINT"YOU'RE GETTING
    WORSE!"
120 IF T<>144 AND N>9 THEN PRINT"YOU'LL HAVE TO
    IMPROVE YOU KNOW!"
130 IF NAME$="JOHN" OR T=144 THEN PRINT"GLAD TO
    HAVE MET YOU ";NAME$
```

Here is an explanation of how Intelligence Test works:

10 CLEAR SCREEN

20 DISPLAY THE MESSAGE "TYPE IN A NUMBER LESS THAN 10", WAIT FOR A NUMBER AND THEN ASSIGN THAT NUMBER TO THE VARIABLE N

30 IF THE VALUE OF N IS LESS THAN 10 DISPLAY THE MESSAGE "WELL DONE!", OTHERWISE DISPLAY THE MESSAGE "IDIOT"

40 DISPLAY THE MESSAGE "TYPE IN YOUR NAME", WAIT FOR A NAME AND THEN ASSIGN THAT NAME TO THE VARIABLE NAME$

50 IF THE VARIABLE NAME$ REPRESENTS THE WORD "JOHN" THEN DISPLAY THE MESSAGE "HELLO JOHN!", OTHERWISE DISPLAY THE MESSAGE "I DON'T KNOW YOU!"

60 IF THE VALUE OF N IS LESS THAN 10 THEN DISPLAY THE MESSAGE "YOU'RE QUITE CLEVER" FOLLOWED BY THE STRING OF LETTERS REPRESENTED BY THE VARIABLE NAME$, OTHERWISE DISPLAY THE MESSAGE "YOU'RE PRETTY STUPID AREN'T YOU" FOLLOWED BY THE STRING OF LETTERS REPRESENTED BY NAME$

70 DISPLAY THE MESSAGE "TYPE IN THE ANSWER TO 12*12", WAIT FOR A NUMBER AND ASSIGN THAT NUMBER TO THE VARIABLE T

80 IF THE VALUE OF T IS 144 THEN DISPLAY THE MESSAGE "WELL DONE!", OTHERWISE DISPLAY THE MESSAGE "TWIT!"

90 IF THE VALUE OF T IS 144 AND THE VALUE OF N IS LESS THAN 10 THEN DISPLAY THE MESSAGE "YOU ARE A GENIUS" FOLLOWED BY THE STRING OF LETTERS REPRESENTED BY THE VARIABLE NAME$

100 IF THE VALUE OF T IS 144 AND THE VALUE OF N IS GREATER THAN 9 THEN DISPLAY THE MESSAGE "YOU ARE IMPROVING!"

110 IF THE VALUE OF T IS ANYTHING OTHER THAN 144 AND THE VALUE OF N IS LESS THAN 10 THEN DISPLAY THE MESSAGE "YOU'RE GETTING WORSE!"

120 IF THE VALUE OF T IS ANYTHING OTHER THAN 144 AND THE VALUEOF N IS GREATER THAN 9 THEN DISPLAY THE MESSAGE "YOU'LL HAVE TO IMPROVE YOU KNOW!"

130 IF THE VARIABLE NAME$ REPRESENTS THE WORD "JOHN" OR THE VALUE OF T IS 144 THEN DISPLAY THE MESSAGE "GLAD TO HAVE MET YOU" FOLLOWED BY THE STRING OF LETTERS REPRESENTED BY THE VARIABLE NAME$

You may have noticed that all the commands we have so far covered have been used in this program. You may also have noticed two new symbols creeping in: these are < which means 'less than' and > which means 'bigger than'. When these two symbols are put together they mean 'not equal to'.

The instructions AND and OR were also used in the IF. THEN. . .ELSE constructions. AND is used with IF to say that IF a first condition is fulfilled and a second condition is fulfilled THEN go on and carry out the following command(s). OR is used to say that IF either a first condition OR a second condition is fulfilled THEN go on and carry out the following command(s).

Brackets can be used in an IF. . . THEN. . . ELSE statement to allow a series of conditions to be treated as one, eg this line:

```
100 IF (A=1 AND B=1) OR (A=2 AND B=2) THEN PRINT
"HELLO"
```

tells the computer that IF either the value of A is 1 AND the value of B is 1, OR the value of A is 2 AND the value of B is 2 THEN display the word 'HELLO'.

11
FOR. . .NEXT Loops

Look at this program which PRINTs out the multiples of 12 up to 12*12:

```
 10 CLS
 20 PRINT 1*12
 30 PRINT 2*12
 40 PRINT 3*12
 50 PRINT 4*12
 60 PRINT 5*12
 70 PRINT 6*12
 80 PRINT 7*12
 90 PRINT 8*12
100 PRINT 9*12
110 PRINT 10*12
120 PRINT 11*12
130 PRINT 12*12
```

Very often you need the computer to carry out a series of instructions several times, so instead of typing out the instructions over and over again, as in the above program, we use FOR. . .NEXT loops.

A FOR. . .NEXT loop instructs the computer to carry out all the instructions between the FOR command and the NEXT command a set number of times.

Here is an example program which will PRINT all the multiples of 12 up to 12*12 using a FOR. . .NEXT loop:

```
10 FOR N = 1 TO 12
20 PRINT N;"*12=";N*12
30 NEXT N
```

Much shorter than the previous program, isn't it? Here's how it works:

Line 10 tells the computer to start repeating all the instructions between the FOR and NEXT command 12 times.

Each time the computer goes through the loop it adds 1 to the value of N.

Line 20 uses this fact to multiple the current value of N by 12.

Line 30 finishes off the loop (the variable N does not have to be added at the end, but it is a good practice to do so).

Trying changing line 10 to:

10 FOR N = 12 to 24

Then RUN the program. You will see the multiples of 12 from 12*12 to 12*24 because the value of N starts off at 12 and increases by one until it reaches 24.

Now change lines 10 and 20 to:

10 FOR N = 0 TO 144 STEP 12
20 PRINT N

When you RUN the program this time you will see the multiples of 12 PRINTed on the screen again, but this time the value of N is increasing in STEPs of 12.

The STEP section of the FOR...NEXT loop tells the computer to add more than one to the value of the variable being used (in this case the variable is N, although it could be any variable).

The STEP can be as much as you like and the size of the STEP is defined after the STEP command (STEP 12 in the above program). You can also have negative steps. For instance this program PRINTs the multiples of 12 in reverse order:

10 FOR N = 144 TO 0 STEP −12
20 PRINT N
30 NEXT N

Here is a program illustrating the use of the FOR...NEXT loop for a different purpose, as a delay:

Multiplication Tables

```
10 CLS 3
20 INPUT"WHICH MULTIPLICATION TABLE WOULD YOU
   LIKE";N
30 FOR M=1TO12
40 PRINT N;"*";M;"=";N*M
50 NEXT M
60 FOR Z=1 TO 4000:NEXT Z
70 RUN
```

When you RUN this program you will be asked which multiplication table you want and then all the multiples up to 12 of the number that you entered will be displayed on the screen.

To give you time to read the multiplication table line 60 goes through a FOR. . .NEXT loop 4000 times without doing anything.

The purpose of this is to cause a delay of almost 4 seconds before line 70 re-starts the program.

If you try changing the 4000 in line 60 you can increase and decrease the length of the delay as much as you like.

You could also take line 60 out to see the difference it makes. The computer clears the screen as soon as it has finished PRINTing the table, so fast that you won't even have time to see it.

This delay FOR. . .NEXT loop is used very often in programs to slow down the speed at which the program RUNs.

As you will notice this program keeps returning to line 10 after it has printed up a table. This is because line 70 keeps RUNning the program for you.

The only way to stop the program is either to switch off the computer or press the BREAK key (which is the correct way).

Pressing the BREAK key will stop any program or command that the computer is carrying out except a second routine.

The program is not lost by pressing BREAK as you will see if you type LIST.

Here's one last program using the FOR. . .NEXT loop. It illustrates delay loops and negative steps, as well as the different coloured screens which are possible on the Dragon:

```
10 FOR N=0 TO 8
20 CLS N
30 FOR M=0 TO 500:NEXT M
40 NEXT N
50 FOR N=8 TO 0 STEP -I
60 CLS N
70 FOR M=0 TO 500:NEXT M
80 NEXT N
90 RUN
```

Line 10 starts off the first FOR. . .NEXT loop which decides the screen colour. Line 20 then clears the screen to the colour which has the code which is the same as the number currently in N. Line 30 uses a FOR. . .NEXT loop for a delay, before line 40 finishes off the first loop. Lines 50–80 are the same as lines 10–40 except that the first FOR. . .NEXT loop works downwards from 8 to 0. Line 90 re-starts the program.

12
GOTO, GOSUB and RETURN

In the short section on the RUN command we said that a program is worked through in the numerical order of the line numbers *unless the computer is told to do otherwise.*

The GOTO instruction tells the computer to continue the program at a different line instead of carrying on as normal. Here is a short example program:

```
10 PRINT "*";
20 GOTO 10
```

When you RUN the program the computer will PRINT a star and then find that line 20 is telling it to go back to line 10 and carry on from there.

For this reason the computer will carry on PRINTing stars until you stop it by pressing the BREAK key.

The GOSUB command tells the computer to GO to the SUBroutine starting at a given line number.

A subroutine is a program within a program which is used several times during the main program.

The idea of having subroutines is to save you having to retype the routine each time you need it.

The computer will carry on working through the program from the specified line until a RETURN command is reached.

When a RETURN command is reached at the end of a subroutine the computer returns to the command directly after the last GOSUB command and continues with the program from that point.

Type this program in:

```
10 PRINT "*";
20 GOSUB 40
30 GOTO 10
40 PRINT "@";
50 RETURN
```

Here is a simple explanation of how the program works:

```
10 DISPLAY A STAR
20 GO TO THE SUBROUTINE STARTING AT LINE 40
```

30 GO BACK TO LINE 10 AND CARRY ON WITH THE PROGRAM FROM THERE
40 DISPLAY A @ SYMBOL
50 GO BACK TO THE NEXT COMMAND AFTER THE LAST GOSUB COMMAND (IN THIS CASE LINE 30)

Two things to remember with the GOTO and GOSUB command are:

(1) The line number after the command *cannot* be replaced with a variable.

(2) Any commands after a GOTO command on the same line will not be carried out.

The above program is just a simple demonstration and is not of any real use but by following the program explanation you could try writing a similar short program to perhaps display two separate messages by incorporating the CLS command.

The GOTO and GOSUB commands can, of course, be used with the IF. . . THEN. . . ELSE statement and this structure is one of the most important uses of the IF. . . THEN. . . ELSE statement.

13
Storing your programs on tape

By now our programs are beginning to get quite long, and they will get much longer before the end of this book. It would obviously be very boring and time-consuming to type in a program every time we needed it, and for this reason your Dragon has the ability to store your programs on normal cassette tapes.

You should set up the cassette recorder as explained in the Additional Information leaflet included with your Dragon.

Set the volume on the cassette recorder at roughly half volume and insert a tape (any tape can be used as long as it is a normal bias tape). Now type in the following program (don't worry about how it works, it will be fully explained later):

```
10 OPEN "O", # -1, "FILE"
20 FOR N=0 TO 2000
30 PRINT # -1,65;
40 NEXT
50 CLOSE # -1
```

Press the PLAY and RECORD buttons on the tape recorder. Don't worry about the tape not starting, it's not supposed to. (If you don't have a remote control socket on your tape recorder the tape will start).

Type RUN and the tape will start, allowing the computer to store 2001 number 65s on the tape. When the program stops rewind the tape, type NEW and type this in:

```
10 OPEN "I", # -1, "FILE"
20 FOR N=0 TO 2000
30 INPUT # -1, A
40 PRINT CHR$(A);
50 NEXT
60 CLOSE # -1
```

Press the PLAY button on the cassette recorder and type RUN.

You should see a long string of A's appearing on the screen. If nothing happens turn up the volume. If you get an I/O error rewind the tape and reRUN the program.

Once you have the volume level right type NEW and enter this program:

```
10 CLS
20 PRINT "A PROGRAM"
30 PRINT "SAVED ON TAPE"
40 PRINT "AND LOADED BACK AGAIN!"
50 FOR N=0 TO 3000:NEXT
60 GOTO 10
```

Erase everything on the tape and then rewind it. Press the PLAY and RECORD buttons and then type:

```
CSAVE "PROGRAM"
```

The tape will start and after a short pause will stop again.

Rewind the tape, type NEW and then press the PLAY button. When you have done this type in:

```
CLOAD "PROGRAM"
```

The tape will again start and the screen will clear. A letter S should appear in the top left-hand corner of the screen, after a short pause, followed by an inverse letter F and the word PROGRAM.

When the OK prompt returns stop the tape and rewind it. Type LIST and you will see that your program is back again.

If the program is not there or you receive an I/O error rewind the tape and try the LOADing process again (make sure that you haven't recorded the program over the leader at the start of the tape).

The CSAVE "program name" command tells the computer to store a copy of the program currently in memory on tape.

The CLOAD "program name" command tells the computer to look for the program whose name you have specified on the tape and then transfer it into the memory.

In case you were wondering what a program recorded on tape sounds like try rewinding the tape and taking out the earphone plug.

When you play the tape you will hear a long, dull note followed by a series of high and low pitched squeaks. Nonsense to you, but easily interpreted by your Dragon.

If you do not know where a program ends on a tape and need to record another program after it you can use the SKIPF "program name" command to stop the tape at the end of the program.

Try rewinding the tape with PROGRAM saved on it and press the play button. Now type:

SKIPF "PROGRAM"

The tape will start, the screen will clear and all the messages you usually receive when loading programs will appear in the top left-hand corner of the screen.

When the computer finds the end of the program it will stop the tape.

The SKIPF command, however, does not load a program into memory. It just finds where the program ends on the tape.

It is possible to reroute the sound from the tape recorder through the television's speaker.

Type AUDIO ON, rewind the tape and then type MOTOR ON.

You will hear the program noise being played over the television.

Type MOTOR OFF and the tape will stop.

The AUDIO ON command tells the computer to reroute the sound from the tape through the television, and the AUDIO OFF command turns it off.

The MOTOR ON command tells the Dragon to start the tape recorder motor, and the MOTOR OFF command turns if off again.

If you have more than one program on the tape then you will have to carry out the SKIPF command for each program.

14
EDITing your programs

Until now you have had to retype a program line if you have made a mistake in it. Fortunately, your Dragon is equipped with an EDITOR to help you correct lines without retyping them.

Type the following program line exactly as it is:

10 PRRNT "THEIR AR A LOTT OF MISSTAKES INN THISS LINEE!"

You can easily spot all the mistakes in the line and if you had made any of these mistakes before now you would have had to retype the whole line.

To use the EDITOR to correct the line you must first type in:

EDIT 10

The Dragon will print the number 10 on the screen followed by a space. Press the space bar twice and you will see the letters P and R appear with the cursor moving along in front.

The first correction to make is to change the R to I so press the C key (for Change) and then press I. The letter I will appear after the R with cursor in front.

Carry on pressing the space bar until the E in THEIR appears. Now type C followed by R, then C followed by E. The word THERE has now been corrected.

The next stage is to add an E after the AR, so carry on pressing space until the R of AR appears. Now type I (for Insert) followed by E.

Hold the SHIFT key down and press the up-arrow key to leave the Insert mode.

Type 6 followed by the space bar and the next six characters will appear. We now need to get rid of the extra T so press D (for Delete) and one of the Ts will be erased.

Type 6 followed by the space bar again and you will be ready to delete the extra S, again by pressing the D key.

See if you can delete the extra letters in INN, THISS and LINEE on your own using the method shown here.

When you have made all the corrections, or if you want to see how you are getting on with the corrections type L. The whole line will be displayed with the line number underneath ready for you to make any more corrections.

When you have finished with the line press the ENTER key and the corrected line will be displayed before the computer goes back to normal command mode.

Here is a full list of the EDITOR commands together with what they do:

SPACE	Move cursor along the line
C character	Change the next character for the one specified
n C character	Change the next n characters (where n is any number) for the ones specified
I	Insert all the following characters after the last character
D	Delete the next character
n D	Delete the next n characters
H	Hack (or cut) off the rest of the line and then enter Insert mode
X	Go to the end of the line and enter Insert mode
S character	Search for the specified character and move cursor to that position
nS character	Search for the nth occurrence of the specified character and move the cursor to that position
K	Delete all the lines from the cursor position
n K	Delete the next n characters after the cursor
L	Display line and return to EDIT mode
←	Move cursor backwards along the line
n ←	Move the cursor back n spaces
SHIFT ↑	Leave Insert or Change mode
ENTER	Leave EDITOR mode

We will finish this chapter with a Maths Test program which will ask you multiplication sums. The program is followed by a list of alterations that you can make to change it to an addition test.

You can use the EDITOR to make these alterations (by the way, don't worry about the commands on lines 30 and 40, they will be explained in the next chapter).

Maths Test

```
10 CLS
20 PRINT"***********MATHS TEST***********"
30 A=RND(20)
40 B=RND(20)
50 PRINT"WHAT IS";A;"*";B;
60 INPUT C
```

```
 70 IF A*B=C THEN PRINT"WELL DONE!!":RIGHT=RIGHT+1
 80 IF A*B<>C THEN PRINT"WRONG! THE ANSWER IS";A*B:
   WRONG=WRONG+1
 90 INPUT"ANOTHER SUM";A$
100 IF A$="Y" THEN GOTO 10
110 IF A$<>"N" THEN GOTO 90
120 PRINT"YOU GOT";RIGHT;"OUT OF";RIGHT+WRONG;"!"
```

Here is a list of alterations to make the program into an addition test:

50 PRINT "WHAT IS";A;" + ";B;
70 IF A+B=C THEN PRINT "WELL DONE!!":RIGHT=RIGHT +1
80 IF A+B<>C THEN PRINT "WRONG! THE ANSWER IS";A+B:WRONG=WRONG+1

By the time you have made the above alterations to the program you should be quite efficient at EDITing your programs.

If there is something that you don't understand then try rereading this section and experimenting with that particular EDITing command.

15
RND

In the program at the end of the last chapter we used a new command, RND.

RND is a *function*, that is it takes one or more numbers and uses them to give you back a result. In this case we give RND a number and it will give us back a random number between 1 and the number that we gave it. Try typing this in:

```
10 FOR N = 0 TO 50
20 PRINT RND(20)
30 NEXT
```

This short program will PRINT 51 random numbers between 1 and 20. The number in brackets after the RND function tells the computer that we want the random number to be less than 20.

Try changing the number to 50 (using your newly acquired knowledge of the EDIT command) and reRUN the program. You will now see random numbers between 1 and 50 being PRINTed on the screen.

It is possible to store the random number in a variable, as in the Maths Test program. For example, A = RND(10) will store a random number between 1 and 10 in the variable A.

If you want a random number between 0 and 1 then you should put the number 0 in the brackets after the RND command. Try altering the program to do this.

The RND function is used in almost every program for one thing or another. It can be used to make the computer carry out a command randomly. For example, if you wanted a 50:50 chance that a monster appears in a game you might have this line:

```
100 IF RND(2)= 1 THEN PRINT "LOOK OUT! A MONSTER!"
```

16
PRINT@

In the first chapter we said that there are many variations of the PRINT command and this is one of them. The PRINT @ command does exactly the same thing as the normal PRINT command except that you can decide where you want the characters to be displayed.

Imagine that the picture on the screen is divided up into 512 squares and that each one has a number like the numbers of a house. The first square, which is in the top left-hand corner of the screen, has the number 0 and the last square, in the bottom right-hand corner of the screen, has the number 511.

You can tell the computer to start PRINTing in any of these squares. For example PRINT@10,"HELLO" tells the computer to PRINT the word HELLO with the letter H in the 10th square, the letter E in the 11th square and so on.

On pages 28 and 140 of the Dragon manual are PRINT@ grids which show how the screen is numbered.

Here is a short program using the PRINT@ command:

Random Blocks

```
10 CLS0
20 FOR N=0 TO 600
30 A=RND(512)-1
40 PRINT@A," ";
50 NEXT
```

This program clears the screen to a black background and then PRINTs green blocks all over it. Line 30 chooses a random number between 1 and 512 and then subtracts one (this is because the screen locations go from 0 to 511). Line 40 then PRINTs a space at the Ath square on the screen.

It is possible to display the value of variables using the PRINT@ command, not only strings contained in quotation marks. For example, if we wanted to display the value of the variable D in the centre of the screen we would use a command like this:

PRINT@239,D

Maybe we would like to PRINT the title of our program in the centre of the screen. In this case we might use a line similar to this:

```
120 PRINT@235, "INVADERS"
130 PRINT@266, "FROM SPACE"
```

This would PRINT the word INVADERS in the centre of the screen with the words FROM SPACE directly underneath.

A useful formula to remember is X + Y*32. Using this formula you can control the position of a moving object on the screen by storing its position in two variables (in this case X and Y). The variable X (or whichever one you are using) should contain the object's horizontal position, and Y should contain the object's vertical position. You can then PRINT the object using PRINT@ X + Y*32.

Any text or graphics characters can be PRINTed on any part of the screen in this way, allowing you to produce quite good visual effects.

Try PRINTing a title and then by altering the PRINT@ positions move it around the screen to get the best effect.

The PRINT@ command has many uses, especially in games. The program on the previous page for example could be used to display a maze.

17
INKEY$

In many programs we need to be able to check to see if a key is being pressed or not. Obviously, the INPUT command would not be of much use as we have to press the ENTER key every time, so we use the INKEY$ command instead. The following part of a program demonstrates how we would use the INKEY$ command instead of INPUT:

```
100 PRINT "DO YOU WANT TO PLAY AGAIN?"
110 A$=INKEY$
120 IF A$="Y" THEN RUN
130 IF A$="N" THEN END
140 GOTO 110
```

INKEY$ 'scans' the keyboard to see if you are pressing a key. The INKEY$ command does not stop the program or require you to press the ENTER key, unlike the INPUT command.

There are many uses for the INKEY$ command. You could use it to test the keyboard and move a bat left or right, for instance if the L key is being pressed the bat would move left and if the R key is being pressed then the bat would move right. Another use would be to scan the keyboard to see whether or not you are pressing the F key, firing a rocket if you are. Any key may be used in this way, including the SHIFTed characters.

The following program shows how the INKEY$ command can be used to scan the keyboard:

```
10 FOR N=0 TO 500
20 A$=INKEY$
30 PRINT A$
40 FOR M =0 TO 20:NEXT M
50 NEXT N
```

The only thing that will happen when you RUN the program on the previous page is the screen will slowly scroll. However, if you press any key (except for BREAK and CLEAR) you will see the character on that key appear on the screen.

What is happening? Line 20 stores the character of the key which is currently being pressed in the variable A$. If none of the keys are being pressed then A$ stays empty.

Line 30 then PRINTs the character in A$ before line 40 causes a delay.

INKEY$ is a function which takes a value from the keyboard and uses it to tell you which key, if any, is being pressed.

A very common use for INKEY$ is as a delay line. For example, if you are displaying the instructions for a program you could use INKEY$ to wait for you to press a key before continuing with the instructions in this way:

```
100 PRINT "PRESS ANY KEY TO CONTINUE"
110 IF INKEY$ = "" THEN 110
120 CLS
```

In the next chapter we will use INKEY$, along with all the other commands we have so far used, in a short game. Meanwhile, experiment with the INKEY$ command to make sure you know how to use it properly.

18
CHR$

Every character on the Dragon keyboard has its own code number. The letter A for example has the code 65. It is possible to display these characters on the screen by using their code numbers.

To do this we use the CHR$ command followed by a number in brackets. This tells the computer that we want it to display the character with the code number that we have specified.

Like RND and INKEY$, CHR$ is a function and so we have to tell it which character we want displayed. Try typing in:

PRINT CHR$(65)

This line tells the computer to display the character which has the code 65 (the letter A) on the screen.

It is also possible to display special coloured blocks called *graphics symbols* by using the CHR$ command. For example if you type:

PRINT CHR$(175)

a blue square will be displayed.

Here is a short program which displays every character that the Dragon is capable of displaying on the screen and then displays the characters one by one with their codes:

Character Set

```
10 CLS
20 FOR N=32 TO 255
30 PRINT CHR$(N);
40 NEXT
50 FOR N=32 TO 255
60 PRINT@333,CHR$(N);"  =";N
70 FOR M=0 TO 300:NEXT M
80 NEXT N
```

Let's first look at lines 10 to 40 and see how they work. We already know that the CLS command clears the screen and how the FOR. . . NEXT loop works so you can see that line 20 starts off the loop with the value of N set at 32.

Line 30 then PRINTs the character with the code number N. Line 40 sends the computer back to line 20 which adds one to the value of the variable N. The program continues round and round the loop until the value of N equals 255.

The second part of the program is very similar to the first part, except that the characters are PRINTed one by one together with their codes.

Line 60 handles the PRINTing of the characters, using the PRINT@ command explained earlier. Line 70 causes the delay.

A full list of the codes for the graphics symbols is on page 138 of the Dragon manual.

Here is that program I promised you at the end of the last chapter:

Invader

```
 10 CLS
 20 BASE=431
 30 SHIP=RND(32)+64
 40 LIFE=3
 50 PRINT@BASE," ";CHR$(159);" ";
 60 PRINT@BASE+31," ";CHR$(159);CHR$(159);
   CHR$(159);" ";
 70 PRINT@SHIP,"Y";
 80 IF LIFE=0 THEN PRINT@233,"YOU'RE DEAD!!":END
 90 A$=INKEY$
100 IF A$=CHR$(9) THEN BASE=BASE+1
110 IF A$=CHR$(8) THEN BASE=BASE-1
120 IF BASE>445 THEN BASE=445
130 IF BASE<416 THEN BASE=416
140 IF A$="F" AND MISSILE=0 THEN MISSILE=1:MM=
   BASE+1
150 IF MISSILE=1 THEN MM=MM-32
160 IF MM>0 THEN PRINT@MM,"↑";:PRINT@MM+32," ";
170 IF MM=SHIP THEN FOR N=143 TO 155 STEP 16:
   PRINT@MM,CHR$(N);:FOR N=0 TO 30:NEXT M:NEXT N:
   HITS=HITS+1:SHIP=RND(32)+64
180 IF MM<96 AND MM>0 THEN PRINT@MM," ";:MM=0:
   MISSILE=0
190 G=G+1:IF G<10 THEN GOTO 240
200 G=0
210 PRINT@SHIP," ";
220 Z=RND(2):IF Z=1 THEN SHIP=SHIP+1
230 IF Z=2 THEN SHIP=SHIP-1
240 IF SHIP<63 THEN SHIP=63
250 IF SHIP>94 THEN SHIP=93
260 IF RND(5)=1 AND BOMB=0 THEN BOMB=SHIP
270 IF BOMB>0 THEN BOMB=BOMB+32
280 IF BOMB>0 THEN PRINT@BOMB,"*";:PRINT@BOMB-32,
   " ";
290 IF BOMB=BASE+1 OR BOMB=BASE+32 OR BOMB=BASE+32
```

```
   THEN FOR N=0 TO 7:CLS(N):FOR M=0 TO 50:NEXT M:
   NEXT N:LIFE=LIFE-1:CLS
300 IF BOMB>479 THEN PRINT@BOMB," ";:BOMB=0
310 PRINT@0,"SCORE:";HITS;"   LIVES:";LIFE
320 GOTO 50
```

This program is a kind of simple Space Invaders with only one invader. You move your base left and right with the left and right arrow keys. To fire press the F key.

You have three lives and an infinite number of aliens.

If you read through the program you should be able to understand how all the lines work, but you probably won't be able to understand how each line contributes to the program. For this reason we have split the program up into routines on the next pages to show you what each section does:

Lines 20–40 set up the variables. The variable BASE determines where your base is on the screen. SHIP determines where the Invader is and LIFE keeps a record of how many lives you have left.

Lines 50–60 display your base. The spaces left on either side of the yellow blocks (CHR$(159) is a yellow block) make sure that no trails are left behind the base when it moves.

Line 70 displays the Invader.

Line 80 checks to see if you are dead yet (the END command stops the program).

Lines 90–110 check to see if you are pressing a key and take appropriate action. CHR$(9) is the code for the right-arrow key and CHR$(8) is the code for the left-arrow key. Adding one to the variable BASE or subtracting one has the effect of moving the base right and left one square respectively.

Lines 120–130 make sure that your base hasn't gone off either edge of the screen.

Line 140 checks to see if you are pressing the F key and if you are (and there isn't a missile on the screen already) assigns the number 1 to the variable MISSILE. This is so we can check to see if there is a missile on the screen and assign the value of the variable BASE plus one to the variable MM. This variable determines the position of your missile.

Line 150 — if there is a missile on the screen then move it up one line.

Line 160 PRINTs the missile on the screen and rubs out the missile behind it.

Line 170 checks to see if you have hit the Invader. If you have every dif ferent coloured block is PRINTed over the Invader in an explosion effect.

Line 180 checks to see if your missile has gone off the screen, deleting it if it has.

Lines 190–230 — once in every eleven times round the program the Invader moves. This routine moves it left or right randomly.

Lines 240–250 makes sure that the Invader doesn't go off the screen.

Lines 260–300 — this section controls the Invader's bombs using the same sort of routine as the one controlling your missiles, except that they come down instead of going up.

Line 310 displays your score and how many lives you have left.

Line 320 — this line goes back to line 50 to carry on with the program.

It would be advisable to save this program on tape as we will be coming back to it from time to time to add more commands.

19 PRINT TAB and STRING$

The PRINT TAB command is used to tell the computer which column you want it to start PRINTing in. This is useful for displaying tables. Try the following program:

```
10 CLS
20 INPUT "WHICH COLUMN (0–31)";COLUMN
30 PRINT TAB(COLUMN);"X"
40 FOR N=0 TO 1000:NEXT N
50 GOTO 10
```

This program will ask you which column you want to start PRINTing in and then display an X in that column before pausing and starting again. If you look at line 30 you will see that we have to enclose the column number in brackets after the TAB command and then put a semi-colon before whatever we want to PRINT, in this case a letter X. One thing to remember, though. If you PRINT TAB something, and then try to PRINT TAB something *behind* the original piece of PRINTing, the second piece of PRINTing will appear on the next line down.

STRING$

The STRING$ command is used to display a line of characters. For example, if we wanted to display a line of 20 orange squares we would use this command:

```
PRINT STRING$(20,255)
```

As you can see there are two numbers in brackets after the STRING$ command. The first one (20) tells the computer how many characters you want it to print. The second one (255) is the CHR$ code of the character. The following program displays a whole line of each of the different coloured squares which are available:

```
0 CLS
20 FOR N=143 TO 255 STEP 16
30 PRINT STRING$(32,N);
40 NEXT N
```

20
DELeting

If you have been experimenting with writing your own programs as we have progressed through the book, you have possibly had to delete some of your program lines. Up until now we have been deleting program lines by typing in the line number and pressing ENTER. That's fine for the odd line or two but what about ten or maybe twenty lines? Well the Dragon provides a way of deleting whole blocks of lines in one command – DEL. The command DEL is not used in a program but is used while writing a program. When you use the DEL command, check to make sure that the line numbers you have stated are the ones you want deleted because once you press the ENTER key they will be lost for good.

There are several ways to use the DEL command ie if you type in DEL 20–80 followed by ENTER lines 20 to 80 inclusive will be deleted. You can DELete the whole program by typing DEL – . Before pressing the ENTER key double check that you have the right line numbers entered. If you have made a mistake you can cancel the command by pressing the BREAK key and start again. Below are a list of DEL commands and what they do:

DEL 10	Will delete line 10 only
DEL – 50	Will delete all lines from the start of the program up to and including line 50
DEL60–	Will delete all lines from 60 to the end of the program including line 60
DEL 40–100	Will delete all lines between line 40 and 100 including lines 40 and 100
DEL –	Will delete the whole program

The line numbers we have given are of course just examples. Any line numbers can be used as long as they are in the program. Although this is a useful command it does not pay to be careless so do recheck before you press the ENTER key.

21
RENUMbering

RENUM is another useful command that can be used when writing a program. Remember in an earlier chapter we said the idea of numbering lines in increments of 10 was to allow you to enter lines in between those already entered. Well what happens if you have no more room between those lines set at increments of ten? This is where RENUM comes in — you just RENUMber the lines.

To carry out the RENUM command you must tell the computer what you want renumbering and in what increments. If you just type RENUM followed by ENTER the whole programm will be renumbered starting from the first line, in increments of 10.

When the program is renumbered the Dragon also renumbers all the GOTOs and GOSUBs so that the program flow is not altered. You may RENUMber a program as many times as you wish.

Set out below is the way you can use the RENUM command:

RENUM	Will renumber the whole program in increments of 10
RENUM100,20,5	Will renumber the program from the old line 20 replacing it with 100 and increasing by increments of 5 ie 100, 105, 100 etc
RENUM50,,2	Will renumber the whole program, the first line numbered 50 and then increasing by increments of 2
RENUM,,30	Will renumber the whole program, the first line starting with the number 30 and the rest by increments of 30 ie 30, 60, 90 etc

22 SOUND

We have had two chapters on useful commands when writing a program, now we can return to a useful command to use in your program.

Any good program can be improved with the careful addition of some SOUND.

The Dragon provides two ways of entering sound to your program, but at this stage we will look at one and that is the command SOUND.

There are two instructions you must give when entering a SOUND command, the pitch and the duration. Try typing in:

SOUND 10,3 followed by ENTER.

With the pitch instruction the numbers range from 1, the lowest note to 255, the highest note, 89 being middle C on a piano. The duration also ranges from 1, the shortest duration, to 255 for the longest.

For the full range of tones type in the following little program:

FOR N = 1 TO 255:SOUND N,1:NEXT

This gives the whole range of sounds available with the SOUND command. To try the different combinations of pitch and duration type in the next program:

Sound Demonstration

```
10 CLS
20 INPUT"SELECT PITCH (1 TO 255)";P
30 INPUT"SELECT DURATION (1 TO 255)";D
40 SOUND P,D
50 GOTO 10
```

This little program, when RUN, will ask you to select the pitch, before storing that number in the variable P. It will then ask you for the duration and store the number entered in D.

Line 40, uses the variables P,D to play the note for the duration asked for.

Line 50 then returns you to the start of the program so that you may try some more combinations.

If you have saved the program we used in the section on CHR$ you can reload it to add some SOUND commands. If not perhaps you would like to retype it in and this time save it on tape as we will be coming back to it from time to time to add to it.

Now that we have learn about the SOUND command we can add some sound to your INVADER program.

You should change the following lines (you can use the EDITOR to make these changes):

```
140 IF A$="F" AND MISSILE=0 THEN MISSILE=1:
   MM=BASE+1:SOUND 100,1
160 IF MM>0 THEN PRINT@MM,"↑";:PRINT@MM+32," ";:
   SOUND 200,1
170 IF MM=SHIP THEN FOR N=143 TO 255 STEP 16:
   PRINT@MM,CHR$(N);:SOUND N,1:FOR M=0 TO 30:
   NEXT M:NEXT N:HITS=HITS+1:SHIP=RND(32)+64
280 IF BOMB>0 THEN PRINT@BOMB,"*";:PRINT@BOMB--32,
   " ";:SOUND 255,1
290 IF BOMB=BASE+1 OR BOMB=BASE+32 OR BOMB=BASE+32
   THEN FOR N=0 TO 7:CLS (N):SOUND N+1,1:FOR M=0 TO
   50:NEXT M:NEXT N:LIFE=LIFE-1:CLS
```

The SOUND commands which we have added to the above lines makes the following changes to the program:

Line 140 produces the sound when you fire your missiles.

Line 160 makes the noise as the missile goes up.

Line 170 produces the sound effects when the Invader is hit.

Line 280 makes the noise for the Invaders bomb.

Line 290 makes the noises when you get blown up.

When you have made these adjustments please reSAVE the program as we will be coming back to it again later on and this will save you having to type it in again.

The SOUND command can liven up your programs and make them a lot more interesting to use.

However, you shouldn't use too much sound as it slows the program down, as you can see from the Invader program.

23
DIM and ARRAY variables

By now you should have mastered variables and fully understand how to use them. At least you should understand how to use simple variables. So far we have only used simple variables, but there is another type of variable that we haven't met yet, the *array variable.*

An array variable is exactly the same as a simple variable apart from the fact that the variable is always followed by a number in brackets.

For example, A, B, HELLO and ZZ are all simple variables, but A(1), B(18), HELLO(54) and ZZ(3) are all array variables.

The number in brackets after the array variable is the index number and tells the computer which part of the variable you want to use.

For example, the variable B(18) is referring to the 18th number in the variable B, and the variable B(12) is referring to the 12th number in the variable B. You may store as many numbers as you like in a single array variable.

Unfortunately, array variables take up a lot more of your Dragon's memory than simple variables so you have to tell the computer to save some extra memory space to store them in.

To do this we must DIM (for DIMension) the array variable before we use it and tell the computer how many numbers you will be storing in it. For example, if you wanted to store 12 numbers in an array called BB then you would use a line like this:

```
10 DIM BB(11)
```

This command tells the computer to save enough memory to store 12 numbers in the array variable BB. You may think that we have made a printing mistake here by using an 11 instead of a 12, but the first index number that we can use with an array variable is 0. We are telling the computer that the index numbers will range from 0 to 11, a total of 12 index numbers.

Type in the following lines in addition to line 10:

```
20 FOR N = 0 TO 11
30 INPUT "TYPE IN ANY NUMBER";BB(N)
40 NEXT N
```

```
50 CLS
60 FOR N=0 TO 11
70 PRINT BB(N)
80 NEXT N
```

When you RUN this program you will be asked to type in 12 numbers, one by one.

When you have done this the screen will clear and the 12 numbers which you typed in will be displayed. The program works in this way:

10 RESERVE ENOUGH MEMORY SPACE TO STORE 12 NUMBERS IN THE ARRAY VARIABLE 'BB'
20 START REPEATING ALL COMMANDS BETWEEN THIS LINE AND THE 'NEXT' COMMAND ON LINE 40 TWELVE TIMES
30 DISPLAY THE MESSAGE "TYPE IN ANY NUMBER", WAIT FOR A NUMBER TO BE ENTERED AND THEN STORE THAT NUMBER IN THE Nth PART OF THE ARRAY VARIABLE 'BB'
40 MARKS THE END OF THE 'FOR...NEXT' LOOP
50 CLEAR THE SCREEN
60 START REPEATING ALL COMMANDS BETWEEN THIS LINE AND THE 'NEXT' COMMAND ON LINE 40 TWELVE TIMES
70 DISPLAY THE Nth VALUE OF THE ARRAY VARIABLE 'BB'
80 MARKS THE END OF THE 'FOR...NEXT' LOOP

This program uses *one-dimensional* array variables to store a list of numbers and use any one of those numbers whenever you want to.

As you can see, you can use one dimensional array variables to store a list of numbers and use any one of those numbers whenever you want to.

However, imagine you wanted to store a whole table of numbers in a variable. In this case you would use a two-dimensional array variable.

This kind of variable has two index numbers,the first one is the row index and the second one is the column index. So, imagine you had a table of how many goals a number of football teams scored in each of four games. The table might look like this:

Team	**Match 1**	**Match2**	**Match3**	**Match4**
LIVERPOOL	2	1	3	0
IPSWICH	1	2	2	1
ARSENAL	0	3	1	1
SPURS	2	2	1	2
NORWICH	1	0	2	2

If, for example, you wanted to find out how many goals Ipswich scored in their third game you would first find Ipswich, which is in the second row, and then look across to Match 3 which is in the third row, to discover that they scored 2 goals in this particular game.

Now let's put this into a program:

```
 10 CLS
 20 DIM SCORE(4,3)
 30 SCORE(0,0)=2:SCORE(0,1)=1:SCORE(0,2)=3:
    SCORE(0,3)=0
 40 SCORE(1,0)=1:SCORE(1,1)=2:SCORE(1,2)=2:
    SCORE(1,3)=1
 50 SCORE(2,0)=0:SCORE(2,1)=3:SCORE(2,2)=1:
    SCORE(2,3)=1
 60 SCORE(3,0)=2:SCORE(3,1)=2:SCORE(3,2)=1:
    SCORE(3,3)=2
 70 SCORE(4,0)=1:SCORE(4,1)=0:SCORE(4,2)=2:
    SCORE(4,3)=2
 80 INPUT"NAME OF TEAM";TEAM$
 90 IF TEAM$="LIVERPOOL" THEN N=0:GOTO 150
100 IF TEAM$="IPSWICH" THEN N=1:GOTO 150
110 IF TEAM$="ARSENAL" THEN N=2:GOTO 150
120 IF TEAM$="SPURS" THEN N=3:GOTO 150
130 IF TEAM$="NORWICH" THEN N=4:GOTO 150
140 PRINT"I DON'T KNOW THAT TEAM":PRINT"PLEASE TRY
    AGAIN":GOTO 70
150 INPUT"WHICH MATCH";MATCH
160 MATCH=MATCH-1
170 IF MATCH>3 OR MATCH<0 THEN PRINT"TRY AGAIN":
    GOTO 140
180 PRINT"THE SCORE WAS";:PRINT SCORE(N,MATCH)
```

This program, when RUN, will clear the screen and ask you for the name of the team. It will then ask you which match you want the score of and then tell you how many goals that team scored in that match. The program works like this:

10 CLEAR THE SCREEN
20 RESERVE ENOUGH MEMORY FOR THE TWO DIMENSIONAL ARRAY 'SCORE' FOR 5 ROWS AND 4 COLUMNS
30 STORE THE NUMBER 2 IN THE 1st ROW AND THE 1st COLUMN OF THE VARIABLE 'SCORE' etc.
40 STORE THE NUMBER 1 IN THE 2nd ROW AND THE 1st COLUMN OF THE VARIABLE 'SCORE' etc.
50 STORE THE NUMBER 0 IN THE 3rd ROW AND THE 1st COLUMN OF THE VARIABLE 'SCORE' etc.
60 STORE THE NUMBER 2 IN THE 4th ROW AND THE 1st COLUMN OF THE VARIABLE 'SCORE' etc.
70 STORE THE NUMBER 1 IN THE 5th ROW AND THE 1st COLUMN OF THE VARIABLE 'SCORE' etc.

```
 80 DISPLAY THE MESSAGE "NAME OF TEAM" AND THEN
    WAIT FOR AN ANSWER BEFORE STORING THAT ANSWER
    IN THE VARIABLE 'TEAM$'
 90 IF THE VARIABLE 'TEAM$' REPRESENTS THE WORD "LIV-
    ERPOOL" THEN STORE THE NUMBER 0 IN THE VARIABLE
    'N' AND THEN GO TO LINE 150
100 IF THE VARIABLE 'TEAM$' REPRESENTS THE WORD
    "IPSWICH" THEN STORE THE NUMBER I IN THE VARIA-
    BLE 'N' AND THEN GO TO LINE 150
110 IF THE VARIABLE 'TEAM$' REPRESENTS THE WORD
    "ARSENAL" THEN STORE THE NUMBER 2 IN THE VARIA-
    BLE 'N' AND THEN GO TO LINE 150
120 IF THE VARIABLE 'TEAM$' REPRESENTS THE WORD
    "SPURS" THEN STORE THE NUMBER 3 IN THE VARIABLE
    'N' AND THEN GO TO LINE 150
130 IF THE VARIABLE 'TEAM$' REPRESENTS THE WORD
    "NORWICH" THEN STORE THE NUMBER 4 IN THE VARIA-
    BLE 'N' AND THEN GO TO LINE 150
140 DISPLAY THE MESSAGES "I DON'T KNOW THAT TEAM"
    AND "PLEASE TRY AGAIN" BEFORE GOING TO LINE 80
    AND CARRYING ON FROM THERE
150 DISPLAY THE MESSAGE "MATCH" AND THEN WAIT FOR A
    NUMBER TO BE ENTERED
160 SUBTRACT ONE FROM THE VALUE OF THE VARIABLE
    'MATCH'
170 IF THE VARIABLE 'MATCH' CONTAINS A NUMBER
    GREATER THAN 4 OR LESS THAN 0 THEN DISPLAY THE
    MESSAGE "PLEASE TRY AGAIN" BEFORE GOING TO LINE
    150 AND CARRYING ON FROM THERE
180 DISPLAY THE MESSAGE "THE SCORE WAS" AND THEN
    DISPLAY THE NUMBER IN THE Nth ROW AND THE
    'MATCH'th COLUMN OF THE VARIABLE 'SCORE'
```

It is possible to have one dimensional and two-dimensional string arrays by just putting the index numbers in brackets after the string variable ie A$(1,3) is a two-dimensional string array.

These string arrays are used in exactly the same way as numeric arrays except, of course, you can store letters and other characters in them as with simple string variables.

Try the following example:

Race Positions

```
10 CLS
20 DIM A$(3,3)
30 A$(1,1)="JOHN":A$(1,2)="PETER":A$(1,3)="PAUL"
```

```
 40 A$(2,1)="PETER":A$(2,2)="PAUL":A$(2,3)="JOHN"
 50 A$(3,1)="PAUL":A$(3,2)="JOHN":A$(3,3)="PETER"
 60 INPUT"WHICH RACE";R
 70 IF R>3 OR R<1 THEN PRINT"PLEASE TRY AGAIN":
   GOTO 60
 80 INPUT"WHICH POSITION";P
 90 IF P>3 OR P<1 THEN PRINT"PLEASE TRY AGAIN":
   GOTO 80
100 PRINT A$(R,P)
110 GOTO 60
```

This program sets up a two-dimensional string array and then stores the names of the people who came first, second and third in each of the three races in the variable A$. The array, if written out on a piece of paper, would look like this:

	1st PLACE	2nd PLACE	3rd PLACE
1st RACE	John	Peter	Paul
2nd RACE	Peter	Paul	John
3rd RACE	Paul	John	Peter

From this table we can work out that the first index number used with the variable A$ refers to the race and the second number refers to the position. Therefore if we want to find out who came second in the first race then we would look in the variable A$(1,2) and find the name Peter stored there.

If you need to DIMension more than one array variable in a program then you need not use a new DIM statement for each variable. Instead you may simply use one DIM statement and separate each variable with a comma eg

10 DIM A$(20),B(12,14)

It is possible to have arrays in up to five dimensions on the Dragon (ie A(1,4,3,5,2) is a five dimensional array). However, arrays with more than three dimensions are not often used, partly because they use a lot of memory, and partly because there are not many uses for four or five dimensional tables.

One last thing about array variables. If you only want to store 10 (or less) numbers or letters in an array then you do not need to DIMension the variable. You do, however, need to DIMension the variable if you are going to store more than 10 numbers or letters in it.

24
REM, END and STOP

Quite often when writing a long program you forget what part of your program does and then have to spend ages scratching your head and wondering what it's supposed to do.

Fortunately your Dragon has a command to help you remember what a routine does, the REM command. Here is an example of how the REM command is used in a program:

```
270 REM FIRING ROUTINE
```

This line will be totally ignored by the computer when the program is RUN and its only purpose is to remind you what that particular routine does. You may type anything you like after the REM command.

There is an abbreviated form of the REM command and this is the apostrophe ('). For example the line means exactly the same as the above line:

```
270 ' FIRING ROUTINE
```

Remember back in the INVADER program we used the END command to stop the program when you have been killed? Well, the END command tells the computer that it has come to the end of the program and that it must stop here, even if the END command is not on the last line of the program. The STOP command is very similar to the END command except that you are told which line the program stopped at. There is also one more difference which will be explained on the next page, about the CONT command.

25 CONT, TRON and TROFF

The CONT (short for CONTinue) command tells the computer to restart the program after a STOP command has been carried out. Type in this example:

```
10 PRINT "TYPE CONT TO CARRY ON WITH THIS PROGRAM"
20 STOP
30 PRINT "THIS PROGRAM HAS RE-STARTED!!!!"
```

When you RUN this program line 10 will be carried out and then line 20 will stop the program. If you type CONT line 30 will be carried out. The CONT command can also be used to restart a program after you have stopped the program by pressing the BREAK key.

When you have a mistake in a program which does not cause an error but nevertheless stops the program from operating properly it is usually very difficult to find the error. To help you with this error trapping the Dragon has a TRACE which constantly displays the line which it is currently working on. To turn the trace on you should use the TRON command, and to turn it off you should use the TROFF command. To see how the trace works type TRON and then RUN the program shown above. You will see this displayed:

```
[10] PRESS CONT TO CARRY ON WITH THIS PROGRAM
[20]
BREAK IN 20
```

If you now type CONT this will appear:

```
[30] THIS PROGRAM HAS RESTARTED!!!!
```

26
ON. . .GOTO. . .and ON. . .GOSUB

The ON. . .GOTO and ON. . .GOSUB commands are really combinations of the IF. . .THEN and GOTO or GOSUB commands. To explain what these commands are it is useful if you first know what they look like. Here is an example of the ON. . .GOTO command:

```
230 ON X GOTO 100,150,200,250,300
```

This line first looks to see what number the variable X represents.

If X is 1 (or a number between 1 and 2) then the computer goes to line 100 and carries on with the program from there.

If X is 2 (or a number between 2 and 3) then the computer goes to line 150 and carries on from there, and so on.

However if the value of X is 6 or more, or the value of X is less than 1 then an error occurs.

The ON. . .GOSUB command works in exactly the same way as the ON. . .GOTO command but goes to a subroutine which it can RETURN from (as with the normal GOSUB command).

Here is an example of how the ON. . .GOSUB command can be used:

```
10 CLS
20 N = RND(6)
30 ON N GOSUB 60,70,80,90,100,110
40 IF RND(10) = 1 THEN FOR N = 0 TO 2000:NEXT
50 FOR M = 0 TO 50:NEXT:GOTO 10
60 PRINT:PRINT"*":RETURN
70 PRINT"*":PRINT:PRINT"  *":RETURN
80 PRINT"*":PRINT" *":PRINT"  *":RETURN
90 PRINT"* *":PRINT:PRINT"* *":RETURN
100 PRINT"* *":PRINT" *":PRINT"* *":RETURN
110 PRINT"* *":PRINT"* *":PRINT"* *":RETURN
```

When you RUN this program you will see a dice rolling in the top left-hand corner of the screen. Every now and then the dice stops.

Here is a simple explanation of how the program works:

```
10      CLEAR THE SCREEN
20      SELECT A RANDOM NUMBER BETWEEN 1 AND 6 AND
        STORE THAT NUMBER IN THE VARIABLE N
```

30	IF THE VALUE OF N IS 1 THEN GO TO THE SUBROUTINE STARTING AT LINE 60, IF THE VALUE OF N IS 2 THEN GO TO THE SUBROUTINE STARTING AT LINE 70 ETC
40	SELECT A RANDOM NUMBER BETWEEN 1 AND 10 AND IF THAT NUMBER IS I THEN PAUSE FOR A FEW SECONDS
50	PAUSE FOR A SHORT WHILE THEN GOTO LINE I0 AND CARRY ON WITH THE PROGRAM FROM THERE
60	MOVE THE CURSOR DOWN ONE LINE AND THEN DISPLAY A SPACE FOLLOWED BY A STAR BEFORE RETURNING TO THE NEXT COMMAND AFTER THE GOSUB COMMAND (LINE 40)
70–110	SAME AS LINE 60 EXCEPT DIFFERENT VARIATIONS OF STARS AND SPACES

The ON...GOTO and ON...GOSUB commands are very useful commands and save a lot of typing and memory. Imagine having to have separate lines each using an IF...THEN command to tell the computer which line to go to!

You must, however, remember to make sure that the variable you are using is not bigger than the number of line numbers that you have after the ON...GOTO or ON...GOSUB command otherwise you will receive an error.

27
String handling

You probably understand by now how to use string variables properly, but there are a lot of ways in which you can manipulate strings that we haven't yet met. These are explained on this and the following pages.

LEFT$

In our programs so far we have sometimes had lines similar to these:

```
160 INPUT"ANOTHER GAME";A$
170 IF A$ = "Y" THEN RUN
180 END
```

As you can see, in this particular routine we have to type Y if we want another go. Some people like to type YES if they want another go so we could replace line 170 with this line:

```
170 IF LEFT$(A$,1) = "Y" THEN RUN
```

This line introduces a new command, LEFT$. In this particular example we are telling the computer to look at the first character in the string variable A$ and if it is the letter Y then to restart the program. The number in the brackets following the LEFT$ command tells the computer that you want the first character, and the A$ is the variable that you are using.

The LEFT$ command has a wide range of uses. For example you could use it to allow only people with FR as the first two letters of their name to play a game. This kind of routine could be used in this case:

```
10 INPUT"WHAT IS YOUR NAME";NAME$
20 IF LEFT$(NAME$,2)< > "FR" THEN END
```

RIGHT$

RIGHT$ is very similar to LEFT$ except that it is used to find the LAST instead of the first characters of a string variable. Here is an example:

```
10 INPUT"TYPE IN SOME CHARACTERS";A$
20 PRINT"THE LAST 3 CHARACTERS THAT YOU TYPED IN
WERE";RIGHT$(A$,3)
```

The RIGHT$ command at the end of line 20 displays the last 3 characters of the string variable 'A$'.

MID$

We now know how to find the first and last characters of a string. How do we find the middle characters of a string. We could, of course, use a routine like this:

```
100 A$= "SUPERCALIFRAGILISTICEXPIALIDOTIOUS"
110 B$=RIGHT$(A$,9)
120 C$=LEFT$(B$,5)
130 PRINT C$
```

This routine stores the last nine characters of the string A$ in the string B$, and then stores the first five characters of the string B$ in the string C$. This has the effect of finding the 26th character in the string A$ and the next four characters after it. Luckily, we do not have to use this complicated routine to find characters somewhere in the middle of a string. Instead we use MID$. We can replace lines 110 and 120 with this single line using the MID$ command:

```
110 C$=MID$(A$,26,5)
```

This line finds five characters in the middle of A$ starting with the 26th character and stores them in the variable C$.

The MID$ command is followed by a string variable and two numbers, all in brackets. The string variable refers to the variable that you are using, and the first number tells the computer where in the string you want to start. The second number then tells the computer how many characters you want. Here is an example of the MID$ command:

```
70 Z$=MID$(D$,3,5)
```

This means that we want the third character and the four characters following it of the string variable D$. When the computer has found these characters it stores them in the variable Z$.

We can also change part of a string variable using the MID$ command, eg the command:

```
MID$(A$,4,5)="HELLO"
```

will change the fourth to eighth characters of the variable A$ to HELLO.

LEN

The LEN command is not really a string handling function, but it is a very useful command to use when manipulating strings. This command is used to find out how many characters a string variable contains and is used in this way:

```
10 INPUT"TYPE IN SOME CHARACTERS";A$
20 PRINT"YOU TYPED IN";LEN(A$);"CHARACTERS!"
```

The LEN command in line 20 is followed by the string variable in brackets to tell the Dragon which string you are referring to. The LEN command can be used to find the length of ANY string variable, just as long as you specify which variable you want to use.

By now you should understand how to find the middle, left and right parts of strings properly, as well as find out how many characters are in the string. If you don't understand these properly then you should go back and reread this section.

28
DEF FN

The DEF FN command is used to define your own functions, in other words you can make up your own function and use it whenever you like in your programs. For example, you might need to work out the areas of circles several times during your program. Instead of typing out the calculation to work out the area each time you need it you could define your own function to do this:

```
10 DEF FNA(R)=3.1415926*R↑2
```

This line tells the computer to define a function so that whenever you refer to the variable FNA(R) it will take the value of the variable R, multiply that number by 3.1415926 and then store the result in the variable FNA.

The R in brackets is the *argument* of the function, that is R is the number on which the calculations in the function are carried out on. When you call the function R may be replaced by any number, eg if you typed in PRINT FNA(2) you would receive the answer 12.5663704, which is 2 squared multiplied by 3.1415926.

The A in FNA is the variable part of the function, so you may define as many functions as you like, simply by replacing the A with any other variable. Meanwhile, let's add to line 10 to make a small program:

```
20 CLS
30 INPUT"WHAT IS THE RADIUS OF THE CIRCLE";R
40 PRINT"THE AREA OF THE CIRCLE IS";FNA(R)
```

When you RUN the program the screen will clear and you will be asked for the radius of the circle. Line40 then displays the area of the circle, using the function which was defined in line 10.

29
READ, DATA and RESTORE

In many programs you will need to use a long list of numbers and letters, or DATA, as it is known. You could, of course, use variables to store all these numbers and letters, but this takes up a lot of memory space (and a lot of time). Instead we store all the data in DATA statements. Here is an example of a DATA statement:

```
230 DATA 1,43,52,7,17,9,10,3985
```

As you can see the DATA statement is followed by a long list of numbers each one separated by a comma. Letters can be stored in a similar way:

```
230 DATA "HELLO", "GOODBYE", "FRED", "DOG", "CAT"
```

Each word is enclosed in quotation marks and, like the numbers, is separated from the next by a comma.

Now we know how to store a list of numbers and letters, but how do we use them? Well, when we want to use any number or series of letters we READ them into a variable. Type in this short routine:

```
 10 CLS
 20 DIM A(12)
 30 FOR N = 0 TO 12
 40 READ A(N)
 50 PRINT A(N)
 60 NEXT
 70 END
100 DATA I,1,3,5,8,13,21,34,55,89,144,233,377
```

This routine READs in all the 13 numbers in the DATA statement and stores them in the variables A(0) to A(12). You should notice that the DATA statement is after the end of the program. If you trace the flow of the program (using the TRON command) you will see that the computer never actually goes to line 100. It just knows where all the data is stored and goes straight to the DATA statement rather than the line number.

If you now change line 70 to:

70 GOTO 30

and RUN the program again you will see the 13 numbers PRINTed on the screen again, but when the computer tries to start READing more numbers the second time round it finds that it has run out of data and gives you an ?OD error (out of data error). To cure this we need to put in an extra line:

65 RESTORE

This line tells the computer to go back to the start of the data and carry on READing in numbers (or letters) from the start of the list.

So far our program doesn't do anything in particular except PRINT a list of numbers. To make the program do something we can delete lines 65 and 70 and add the following lines:

45 T = T + A(N)
70 PRINT "THE AVERAGE OF THESE NUMBERS IS";T/13
80 GOTO 80

The program now READs in all the numbers and adds them up as it goes, storing the total in the variable T. When all the numbers have been read in line 70 works out the average. Line 80 forms an endless loop to stop the screen from scrolling up when the program ends. If you delete this line then the top number will disappear when the program ends.

On the next page is a program using the READ, DATA and RESTORE statements.

Address Book

```
 10 REM 'NUMBER' MUST EQUAL THE NUMBER OF NAMES IN
   THE LIST
 20 NUMBER=2
 30 CLS
 40 INPUT"PLEASE ENTER PERSON'S NAME";NAME$
 50 FOR N=1 TO NUMBER
 60 READ A$
 70 IF LEFT$(A$,LEN(NAME$))=NAME$ THEN PRINT A$:
   PRINT:Q=1
 80 NEXT N
 90 IF Q=0 THEN PRINT"SORRY, I CAN'T FIND THAT
   NAME"
100 Q=0
110 PRINT"ANY MORE (Y/N)";
120 A$=INKEY$
130 IF A$="Y" THEN RESTORE:GOTO 30
140 IF A$="N" THEN END
150 GOTO 120
160 REM ENTER PERSON'S NAME AND DETAILS HERE
170 DATA"JOHN SMITH, 12 THE ROAD,       TOWNSVILLE"
180 DATA"JACK JONES, TEL:123456"
```

This program allows you to store the names and addresses of all your friends as DATA statements at the end of the program. The names may be added from line 160 onwards, and the variable NUMBER at the start of the program must be set to the number of names in the list.

The main part of the program is the FOR...NEXT loop from lines 50–80. This READs in each name in turn and checks to see if the first few letters correspond to the name which you are trying to find. If a name is found then it is PRINTed on the screen and the variable 'Q' is set to I so that the computer knows that it has found a name.

When the FOR...NEXT loop is completed line 90 checks to see if any names were found, telling you that it doesn't know the name that you wish to find.

30 Graphics

Until now our programs have not been as interesting as they might be partly because of a lack of colour, and partly because the only characters we have been able to use are the normal keyboard characters and the graphics characters.

However, your Dragon can produce quite good effects using what is known as *low resolution graphics.*

The word graphics is just another word for drawing, and low resolution refers to the thickness of the lines that we can draw. Low resolution drawings use thick lines, and high resolution drawings use thin lines. In the next few pages we will be using low resolution graphics and learning the special commands which we use to make our drawings.

SET

We already know that the screen is split up into squares, each one with its own number. Normally there are 32 squares across the screen and 16 down, but when we use low resolution graphics the screen size increases to 64 squares across and 32 down.

It is possible to light up any of these squares in any of 8 different colours, and to do this we use the SET command. For a simple example of this type in the program below:

Stardust

```
10 CLS 0
20 FOR I=0 TO 2000
30 N=RND(64)-1:M=RND(32)-1
40 C=RND(8)
50 SET(N,M,C)
60 NEXT I
```

This demonstration program works in this way:

10 CLEAR THE SCREEN TO A BLACK BACKGROUND
20 START REPEATING EVERYTHING BETWEEN THIS LINE AND THE 'NEXT' COMMAND ON LINE 60, 2001 TIMES
30 CHOOSE A RANDOM NUMBER BETWEEN I AND 64,

```
    SUBTRACT ONE AND STORE THIS NUMBER IN THE VARI-
    ABLE 'N'. CHOOSE A RANDOM NUMBER BETWEEN I AND
    32, SUBTRACT ONE AND STORE THIS NUMBER IN THE
    VARIABLE 'M'
40  CHOOSE A RANDOM NUMBER BETWEEN I AND 8 AND
    STORE THIS NUMBER IN THE VARIABLE 'C'
50  LIGHT UP THE POINT 'N' SQUARES ACROSS AND 'M'
    SQUARES DOWN FROM THE TOP LEFT-HAND CORNER OF
    THE SCREEN IN THE COLOUR 'C'
60  FINISH OFF THE 'FOR...NEXT' LOOP
```

You should be able to see from this program that the SET command needs three numbers to work. The first number is the number of squares across from the left of the screen the point should be, and the second number is how many squares down from the top of the screen the square should be. The third number tells the computer what colour you want the square to be.

Unfortunately you can't always have two squares side by side in different colours. For instance, you can have a red square at the point three squares across with a yellow square to the right of it, but you cannot have a red square at the point four squares across with a yellow square to the right of it.

The reason for this is that the low resolution graphics being used are really different graphics symbols being displayed on the screen.

If you look at page 138 in the Dragon manual you will see the different graphics symbols. Each graphics symbol is really a space split up into four sections, each of which may be lit up.

However, no graphics character can be more than one colour. For this reason you can only have two different coloured blocks side by side in low resolution graphics if they are in different character squares.

RESET

We now know how to light up different low resolution squares, but how do we delete them? Well, the answer is we RESET the square.

The RESET command resets a square back to the background colour of the screen. A RESET command is followed by two numbers in brackets. These numbers are the position of the squares to be RESET, just like the numbers in the SET command.

However, we do not tell the computer which colour we want the square RESET to because it works out for itself what the background colour is.

Add the following lines to the program from the previous section (Stardust):

```
70 FOR I = 0 TO 2000
```

```
 80 N = RND(64) – 1:M = RND(32) – 1
 90 RESET (N,M)
100 NEXT I
```

When you RUN the program this time you will see the screen fill up with different colour blocks as before, but then the computer starts to RESET the blocks back to the background colour (in this case black).

Line 80 chooses the coordinates of the square to be RESET. In the same way line 30 chooses the one to be SET, and line 90 RESETs the block.

On the next page is a program using the SET and RESET commands to make a ball bounce about on the screen:

Bouncing Ball

```
 10 CLS 0
 20 BX=31:BY=15:C=1
 30 X=1:Y=1
 40 FOR N=0 TO 63
 50 SET (N,0,4):SET (N,31,4)
 60 NEXT N
 70 FOR N=0 TO 31
 80 SET (0,N,4):SET (63,N,4)
 90 NEXT N
100 RESET (BX,BY)
110 BX=BX+X:BY=BY+Y
120 IF BX=61 OR BX=2 THEN X=-X
130 IF BY=29 OR BY=2 THEN Y=-Y
140 C=C+1:IF C=9 THEN C=1
150 SET(BX,BY,C)
160 FOR N=0 TO 30:NEXT
170 GOTO 100
```

Lines 20 and 30 set up the variables that control where the ball is, the direction it is moving in and what colour it is.

The variables BX and BY are the position of the ball, the variables X and Y control its direction and C is the colour.

Lines 40–90 draw the border in red around the screen.

Line 100 RESETs the ball (this stops it from leaving a trail behind it).

Line 11 0changes the position of the ball by adding X and Y to the variables BX and BY.

If X has the value 1 then the ball moves right, if it has the value – I then the ball moves left.

If Y has the value 1 then the ball moves down, if it has the value − 1 then the ball moves up.

Lines 120 and 130 make sure the ball does not go off the screen. If the ball is likely to go off the screen then the variable X or Y is changed in this way: if X is 1 then it becomes − 1, if X is − 1 then it becomes 1. The same goes for the variable Y. This has the effect of reversing the direction of the ball.

Line 140 changes the value of the variable C which controls the colour of the ball.

Line 150 draws the ball again, and line 160 causes a pause before the whole ball moving routine starts again.

If you make the following alterations to the program some quite startling effects can be produced:

100 SET(BX,BY,C)
120 IF BX>62 or BX< 1 THEN X = − X
130 IF BY>30 OR BY< THEN Y = − Y
DELETE LINE 150

These alterations allow the ball to leave a trail behind it and also let it touch the border. This produces a pattern with ever-changing colours.

POINT

Quite often in programs which use graphics we need to see if a square is lit up or not, and if it is what colour it is. To do this we use the POINT command. Try entering the following program:

Searcher

```
 10  CLS 0
 20  X=RND(64)-1
 30  Y=RND(32)-1
 40  IF X=63 THEN X=62
 50  SET(X,Y,4):SET(X+1,Y,4)
 60  FOR X=0 TO 63
 70  FOR Y=0 TO 31
 80  IF POINT(X,Y)=4 THEN PRINT@0,"I'VE FOUND
    IT!";:SET(X,Y,7):SET(X+1,Y,7):FORN=0TO2000:
    NEXT N:RUN
 90  SET (X,Y,5)
100  NEXT Y:NEXT X
```

When you RUN this program you will see a red line appear somewhere on the screen then the screen will gradually turn white. When the white reaches the red line the message "I'VE FOUND IT!" is displayed in the top left-hand corner of the screen.

There is a short pause and then the program starts again. Try to work out how this program works.

Most of the commands will be familiar to you, apart from line 80 that is.

The POINT command looks to see what colour the square at X squares across and Y squares down is.

Line 80 checks to see if the square is red (the number 4 is the code for red) and if it is the rest of line 8 0 is carried out.

As with the RESET command, the numbers in brackets are the coordinates of the square that you want to check. The colour codes are the same as for the CLS and SET commands, with 0 being black.

The code for text is − 1, so if you wanted to check to see if there was some text in the top left-hand corner of the screen you would use IF POINT(0,0) = − I THEN. . . .

We have now reached the stage where we can produce quite a good game, complete with colour graphics and sound. The following program is a breakout game.

The idea of the game is to knock down as much of a multi-coloured wall at the top of the screen as you can by hitting the ball against it. You have a bat which you must use to hit the ball back up as it falls down.

The bat is moved by the Q and W keys, Q to move left and W to move right. To stop the bat from moving you should repress the key that corresponds with the direction the bat is moving.

If you want the bat to move faster you can use the left and right arrow keys instead.

You have only three balls so if you miss the ball and it hits the ground you lose it. To serve each new ball just press any key.

Breakout

```
 10  B=3
 20  CLS 6
 30  IF B=0 THEN FOR N=255 TO 200 STEP -1:SOUND N,1:
    NEXT:SOUND 1,1:END
 40  Z=0
 50  BX=31:BY=15
 60  Y=1:Q=RND(2):IF Q=1 THEN X=1:ELSE X=-1
 70  BAT=28
 80  FOR N=2 TO 62
 90  SET(N,2,4):SET(N,31,4)
100  NEXT N
110  FOR N=2 TO 31
120  SET(2,N,4):SET(62,N,4)
130  NEXT N
140  FOR N=0 TO 31:RESET(0,N):RESET(1,N):NEXT N
150  FOR M=6 TO 12 STEP 2
160  IF M=6 THEN C=3
170  IF M=8 THEN C=8
```

```
180 IF M=10 THEN C=2
190 IF M=12 THEN C=7
200 FOR N=4 TO 61:SET(N,M,C):NEXT N
210 NEXT M
220 SET(BX,BY,5)
230 IF R=0 THEN A$=CHR$(128)+STRING$(11,175)+
   "breakout"+STRING$(12,175):ELSE GOTO 280
240 FOR N=0 TO 32:PRINT@0,RIGHT$(A$,N);
250 FOR M=0 TO 50:NEXT M
260 NEXT N
270 FOR N=0 TO 2000:NEXT N:PRINT@0,"":R=1
280 A$=INKEY$:IF A$="" THEN GOTO 280
290 SET(BX,BY,6)
300 BX=BX+X:BY=BY+Y
310 IF BX>=62 OR BX<4 THEN X=-X:BX=BX+X:SOUND 100,1
320 IF BY<4 THEN Y=-Y:SOUND 120,1:BY=BY+Y:Q=RND(2):
   IF Q=1 THEN X=1:ELSE X=-1
330 IF BY>=30 THEN B=B-1:GOTO 20
340 IF Y=-1 AND POINT(BX,BY)<>4 AND POINT(BX,BY)<>6
   THEN Y=1
350 IF Y=1 AND POINT(BX,BY)=3 THEN Y=-1
360 IF POINT(BX,BY)=3 THEN SCORE=SCORE+50:SOUND
   255,1
370 IF POINT(BX,BY)=8 THEN SCORE=SCORE+25:SOUND
   250,1
380 IF POINT(BX,BY)=2 THEN SCORE=SCORE+15:SOUND
   245,1
390 IF POINT(BX,BY)=7 THEN SCORE=SCORE+10:SOUND
   240,1
400 IF (BX=BAT+2 OR BX=BAT+1 OR BX=BAT+3) AND
   BY>=28 THEN X=0:Y=-1:SOUND 170,1
410 IF (BX=BAT+4 OR BX=BAT+5) AND BY>=28 THEN X=1:
   Y=-1:SOUND 170,1
420 IF (BX=BAT OR BX=BAT-1) AND BY>=28 THEN X=-1:
   Y=-1:SOUND 170,1
430 SET(BX,BY,5)
440 FOR N=0 TO 4:SET(BAT+N,28,1):NEXT
450 IF Z<0 THEN SET(BAT+6,28,6):SET(BAT+8,28,6)
460 IF Z>0 THEN SET(BAT-2,28,6):SET(BAT-4,28,6)
470 B$=INKEY$:IF B$=""THEN GOTO 540
480 IF A$=B$ THEN A$="":Z=0:GOTO 540
490 IF B$="W" THEN Z=1
500 IF B$=CHR$(9) THEN Z=2
510 IF B$="Q" THEN Z=-1
520 IF B$=CHR$(8) THEN Z=-2
530 A$=B$:BAT=BAT+Z
540 IF BAT<4 THEN BAT=BAT+1:GOTO 540
550 IF BAT>56 THEN BAT=BAT-1:GOTO 550
560 PRINT@0,CHR$(128);"SCORE:";SCORE;:PRINT@20,
   "BALLS:";B;
570 SET(63,28,4):SET(3,28,4):RESET(1,28)
580 GOTO 290
```

When you have played Breakout for a while you will probably want to know how it works. If you look through the program you will find that all the commands are familiar to you, but you will probably still not know how the program works.

However, if you read what each section does and then look back at the program you will probably find that you can work out what everything does. It would be a good idea if you experimented with the program and altered it about a bit. You may even find that you can make the program a lot better!

Here is what each section does:

Line 10 sets up the number of balls you have.

Line 20 clears screen to cyan background.

Line 30 checks to see if you have run out of balls. If you have, a FOR . . . NEXT loop makes a series of noises and the program stops.

Lines 40–70 set up the rest of the variables to be used in the program (see next page for what each variable is used for).

Lines 80–130 draw the border in red around the screen.

Line 140 deletes a band of cyan running down the left-hand side of the screen (neatens up the display).

Lines 150–210 draw the wall. Lines 160–190 determines the colour, depending on the value of M.

Line 220 draws the ball.

Lines 230–270 makes the word BREAKOUT appear from the left-hand side of the screen. Notice the way characters are added to the variable A$ using the + sign. This is acceptable when the characters being added are in quotation marks, or if you use the CHR$ or STRING$ commands.

Line 280 waits for a key to be pressed before starting.

Line 290 rubs out the ball.

Lines 300–320 move the ball and make sure that it doesn't go off the screen.

Lines 330 checks to see if the ball has hit the ground. If it has deduct one from the number of balls you have left and go to line 10 to start again.

Line 340 makes the ball bounce up off the blue bricks.

Lines 350–390 check to see if the ball has hit a brick and increases the score if it has. The score is dependent on the colour of the brick.

Lines 400–420 check to see if the ball has hit the bat, bouncing it in the correct direction if it has.

Line 430 draws the ball.

Line 440 draws the bat.

Lines 450–460 rub out the trail left by the bat.

Lines 470 – 520 check to see if you are pressing the control keys and alter the variable Z accordingly.

Line 530 adds the value of Z to the variable BAT. This makes the bat move.

Lines 540–550 make sure that the bat doesn't go off the screen.

Line 560 displays the score.

Line 570— if the bat hits the border it knocks a bit out. This line makes sure the gap doesn't stay there for long.

Line 580 — go back to line 290 and start the main routine again.

The main variables are:

B	— number of balls left
Z	— direction the bat is moving in. If Z is 1 the bat moves right, if Z is −1 the bat moves left
BX and BY	— position of the ball
X and Y	— direction of the ball
BAT	— position of the bat
N and M	— used in FOR. . . NEXT loops

31
PEEK and POKE

Before we look at what the PEEK and POKE commands are and what they do, we need to know how the Dragon's memory is set out.

There are two types of memory, ROM (Read Only Memory) and RAM (Random Access Memory). To explain exactly what each type of memory does would be quite complex, but here is a simplified explanation.

Imagine that your Dragon's memory is made up of glass boxes and that each one has a number on the front of it for identification (*memory addresses*). Each glass box also has a number between 0 and 255 inside it (*memory contents*).

Some of these glass boxes are sealed (these represent the ROM), while others have no lids at all (these represent the RAM).

We cannot change the numbers in the sealed boxes as there is no way for us to get into them, but we can look at the numbers (or PEEK at them).

We can also look into the open boxes and see what numbers they contain, but as they are open we can also change the numbers which are inside (or POKE new numbers into their locations).

There are different types of ROM memory. Imagine that you want to talk to a person who doesn't understand your language. You would need an interpreter. As the Dragon's CPU (Central Processing Unit — the brains of the computer) does not understand BASIC it also needs an INTERPRETER to translate your programs into its own special language. The interpreter is a type of ROM, as is the CPU itself.

The ROM memory is a permanent type of memory whose contents cannot be changed (like the sealed glass boxes).

The RAM, however, loses its contents when you turn off the computer. It is also possible to alter the RAM's contents, by entering a program for example.

Not all the RAM is available for storing programs as some is used by the computer as work space.

Now that we know how the Dragon's memory is organised we can find out what the PEEK and POKE commands are.

The PEEK command looks into a memory location (glass box) and sees what number is stored there (the number in the box). You can PEEK into any memory location, whether it is in the RAM or the ROM.

The POKE command changes the number in a memory location (the number in the box). You can only POKE into the RAM as the ROM cannot be changed. If you try to POKE into the ROM nothing happens.

You may think that there isn't much point being able to look at parts of the memory and change them. However, if you clear the screen and type this in you may think otherwise:

POKE 1024,65

You may not at first notice what happened. A letter A appeared in the top left-hand corner of the screen.

Part of the Dragon's RAM is used for storing the contents of the screen, so if you change the contents of this memory different things can be displayed on the screen. The number 1024 is the memory address of the top left-hand corner of the screen. The number 65 is the ASCII (American Standard Code for Information Interchange) code for the letter A.

If you change the 1024 to any number between 1024 and 1535 you can make the A appear anywhere on the screen.

Pages 136 to 137 of the Dragon manual give a list of the characters available on the Dragon together with their ASCII codes.

If you now type:

PRINT PEEK(1024)

the number 65 will be displayed (as long as the A is still in the top left-hand corner of the screen. If it isn't a different number will be displayed).

What you have told the computer to do in this command is to look into the memory location that has the address 1024 and display its contents. The memory location must be enclosed in brackets after the PEEK command.

Try PEEKing to different memory locations between 0 and 65535 to see what numbers are stored there.

Here is a program which allows you to draw pictures on the screen.

You can have any colour background and you can draw in any colour.

To change the colour that you are drawing in type the number corresponding to the colour you want.

To change the background colour hold the SHIFT key down and type the number corresponding to the background colour that you want.

Both the background colours and the drawing colours correspond to the numbers used with the CLS command. The cursor is moved with the arrow keys.

Artist

```
10 CLS 0
20 C=159
30 CURSOR=1504
40 PRINT@0,"DRAW COLOUR:";CHR$(C);"  CURSOR
  COLOUR:";CHR$(CR);
```

```
 50 CR=C+16:IF CR>255 THEN CR=143
 60 IF C=128 THEN CR=88
 70 POKE CURSOR,CR
 80 A$=INKEY$:IF A$=""THEN 40
 90 IF A$=CHR$(9) THEN D=1
100 IF A$=CHR$(8) THEN D=-1
110 IF A$=CHR$(10) THEN D=32
120 IF A$=CHR$(94) THEN D=-32
130 IF A$="1" THEN C=143
140 IF A$="2" THEN C=159
150 IF A$="3" THEN C=175
160 IF A$="4" THEN C=191
170 IF A$="5" THEN C=207
180 IF A$="6" THEN C=223
190 IF A$="7" THEN C=239
200 IF A$="8" THEN C=255
210 IF A$="0" THEN C=128
220 IF A$="!" THEN CLS 1
230 IF A$=CHR$(34) THEN CLS 2
240 IF A$="#" THEN CLS 3
250 IF A$="$" THEN CLS 4
260 IF A$="%" THEN CLS 5
270 IF A$="&" THEN CLS 6
280 IF A$="'" THEN CLS 7
290 IF A$="(" THEN CLS 8
300 IF A$=")" THEN CLS 0
310 POKE CURSOR,C
320 CURSOR=CURSOR+D:D=0
330 IF CURSOR>1535 THEN CURSOR=CURSOR-32:GOTO 330
340 IF CURSOR<1056 THEN CURSOR=CURSOR+32:GOTO 340
350 GOTO 40
```

Here is a description of what each section does.

Line 10 clears the screen to a black background.

Lines 20–30 set up variables. C is the colour you are drawing in and CURSOR is the position of the cursor on the screen.

Line 40 displays the cursor and draws colours.

Lines 50–60 work out the cursor colour.

Line 70 POKEs the cursor onto the screen.

Lines 80–120 check to see if you want to move. The variable D, when added to the variable CURSOR will move the cursor about.

Lines 130–210 control the changing of the draw colour.

Lines 220–300 control the changing of the background colour.

Line 310 POKEs the trail before the cursor moves.

Line 320 moves the cursor.

Lines 330–340 make sure that the cursor doesn't go off the top or bottom of the screen.

Line 350 goes back to line 40 to start the whole drawing routine again.

32
Using joysticks

This chapter is for those of you who have joysticks fitted to your Dragon.

Your joysticks should be plugged into the sockets marked JSTK L AND JSTK R on the left-hand side of your Dragon (it doesn't matter which joystick goes in which socket). When you have done this you can get down to using your joysticks in programs.

The first thing we need to know is how to find the position the joysticks are in.

Type this short program in and then try moving the joysticks about:

Joystick Test

```
 10 CLS
 20 PRINT@0,"RIGHT JOYSTICK"
 30 PRINT"LEFT-RIGHT";JOYSTK(0)
 40 PRINT"UP-DOWN";JOYSTK(1)
 50 IF PEEK(65280)=126 OR PEEK(65280)=254 THEN
    PRINT"BUTTON pressed":ELSE PRINT"BUTTON NOT
    PRESSED"
 60 PRINT
 70 PRINT STRING$(32,45)
 80 PRINT"LEFT JOYSTICK"
 90 PRINT"LEFT-RIGHT";JOYSTK(2)
100 PRINT"UP-DOWN";JOYSTK(3)
110 IF PEEK(65280)=125 OR PEEK(65280)=253 THEN
    PRINT"BUTTON pressed":ELSE PRINT"BUTTON NOT
    PRESSED"
120 IF PEEK(65280)=124 OR PEEK(65280)=252 THEN
    PRINT"BOTH BUTTONS PRESSED":ELSE PRINT
130 GOTO 20
```

If you look at lines 30, 40, 80 and 90 you will see a new command — JOYSTK.

This command is used to find the current position of each joystick. The number in brackets tells the computer which joystick you want and whether you want the left-right or up-down position.

If you look at lines 30, 40, 90 and 100 you will see which number corresponds with each joystick and its position.

Lines 50, 110 and 120 check to see if the joystick buttons are being pressed. PEEK(65280) returns different numbers according to which buttons are being pressed — see program for numbers.

Here are some alterations to make to your Invader program to make it work on joysticks:

```
 90 IF JOYSTK(0)> 31 AND JOYSTK(1)< 31 THEN BASE =BASE+ 1
100 IF JOYSTK(0)< 31 and JOYSTK(1)< 31 THEN BASE=BASE – 1
140 IF (PEEK(65280) = 126 OR PEEK(65280) = 254) AND MISSILE= 0
THEN MISSILE= 1:MM = BASE + 1:SOUND 100,1 DELETE LINE 110
```

Your base is now moved using the right joystick. The base only moves if the joystick is forward as well as left or right. To fire use the red firing button.

Here is a conversion to use joysticks on your Breakout game:

```
280 IF PEEK(65280) = 255 OR PEEK(65280) = 127 THEN GOTO 280
470 IF JOYSTK(0)< 31 AND JOYSTK(1)< 31 THEN Z= –2
480 IF JOYSTK(0)> 31 AND JOYSTK(1)< 31 THEN Z = 2
490 IF JOYSTK(1)>31 THEN Z = 0
    DELETE LINES 500, 510, 520 AND 530
```

The bat is now moved in the same way as the base in the Invader program. To serve a new ball you should press the red fire button.

Finally, here is a conversion to allow you to use joysticks on the drawing program at the end of the last chapter:

```
 90 IF JOYSTK(0) = 63 THEN D = 1
100 IF JOYSTK(0) = 0 THEN D = –1
110 IF JOYSTK(1) = 63 THEN D = 32
120 IF JOYSTK(1) = 0 THEN D = –32
130 IF (PEEK(65280) = 126 OR PEEK(65280) = 254) AND C< >128
THEN C = C + 16:IF C> 255 THEN C = 128:GOTO 220
140 IF (PEEK(65280) = 126 OR PEEK(65280) = 254) AND C = 128
THEN C = 143
    DELETE LINES 150–210
```

You now control the cursor with the joystick and change the draw colour by holding down the fire button until you get the right colour.

33
PLAY

So far the noises that we have been able to make on the Dragon have only been bleeps of one kind of another. The Dragon is, however, capable of playing very complex tunes in five octaves, complete with sharp and flat notes.

Normal music can be directly converted using the PLAY command. All you have to do is tell the computer what octave you are using, how long each note should be, the volume and, of course, the notes that you want to be played.

Type in the following example:

Star Wars

```
10 CLS
20 PRINT"I WILL NOW PLAY THE THEME TUNE"
30 PRINT"OF STAR WARS!"
40 FOR N=0 TO 1000:NEXT
50 PLAY"T3DDDL2GO+DL5CO-BAO+L2GDL5CO-BAO+L2GDL5CO-
  BO+CO-L3A"
60 PRINT"AREN'T I FANTASTIC?"
```

If we look at line 50 we can break the tune down into commands and notes. T3 means Tempo 3, tempo being the speed that the tune is played at. The number 3 is the actual speed and can be replaced with any number between 1 and 155 with 155 being the fastest (if you don't specify the tempo the Dragon plays the tune at Tempo 2).

The three D's are the musical note D.

L2 is the length of the note, with L1 the length of a normal note, L2 being half a note, L4 a quarter note and so on. The note can be as short as L255 (one 255th of a note), but you will probably never need a note that short.

G is another musical note (all the letters from A to G are musical notes so we will ignore them for now).

O + means to play the next notes one octave higher.

O – means to play the next notes one octave lower (there are five octaves available on the Dragon).

All the other commands in this tune are the same as the ones explained here but with different values after them.

There are several other commands which can be included in the PLAY command. One of these is Pause. Type this in:

PLAY"ABCDEFGP1GFEDCBA"

This line will play all the notes from A to G, pause for a short while and then play the same notes backwards.

The number after the P is the length of the pause and corresponds with the note lengths.

Another useful command in the PLAY command is V for Volume. The volume is also followed by a number, in this case from 0to 31. If you do not state the volume the computer takes it as 15.

We said earlier that sharp and flat notes can be obtained using the PLAY command. We obtain these notes by adding a symbol after the musical note. For example the note C sharp is indicated by C # in a play command. The + sign can also indicate a sharp.

Flats are indicated by a – sign.

Here is a short program which plays every note that the Dragon can produce:

Notes

```
10 CLS
20 PLAY"O1"
30 FOR N=0 TO 4
40 PLAY"ABCDEFGA#BC#D#E#F#G#"
50 PLAY"A-B-CD-E-F-G-"
60 IF N<>4 THEN PLAY"O+"
70 NEXT N
```

Line 20 sets up the octave that the notes will start in and then lines 40 and 50 play the notes. Line 60 increases the octave (as long as the variable N does not equal 4, if it does then we have already played five octaves and the computer can't go any higher).

If you are familiar with music then you will know about dotted notes. It is possible to obtain dotted notes simply by putting a fullstop after the note (a dot makes a note half as long again).

In the previous program we used the command O+, which increased the octave by one. This plus sign can also be used with the V, L and T commands. The plus sign could be replaced by:

–	substracts one from the value (eg O – decreases the octave by one)
>	multiplies the value by two (eg V> doubles the current volume)
<	divides the value by two (eg L< halves the note length)

As you can probably see from looking at the PLAY command the music is enclosed in quotation marks. For this reason it is possible to store your tune in a string variable and play it as often as you like without typing the tune in again.

Try this example program:

Jingle Bells

```
10 A$="O3L4CAGFL2CL6CCL4CAGFL2DL4B-AGL2EL4O+CCO-B-
   GL2A"
20 FOR N=0 TO 2
30 PLAY A$:PLAY"T>"
40 NEXT N
```

When you RUN this program you will hear Jingle Bells being played faster and faster.

The whole tune was stored in the variable A$ and then played by line 30. Line 30 also increased the tempo.

Line 3 0 is made up of two PLAY commands which can be compressed into one using the eXecute command. Change line 30 to:

30 PLAY "XA$;T> "

This single command tells the computer to eXecute (or play) the tune stored in the variable A$ and then double the tempo.

As you can see the variable A$ is followed by a semi-colon and must ALWAYS be followed by a semi-colon.

It is also possible to use numbers instead of letters to represent the musical notes in the PLAY command.

Page 110 of the Dragon manual gives an illustration of a piano keyboard and shows each note with its equivalent number. If you do use numbers instead of letters you must put a semi-colon after each number.

34
CLEAR

Those of you who have been experimenting with string variables may have encountered the OS (Out of String space) error and wondered what causes this.

This short program may help to explain this:

```
10 A$ = STRING$(201,175)
20 PRINT A$
```

If you RUN this program you will get an OS error. However if you change line 1 0 to:

```
10 A$=STRING$(200,175)
```

and RUN the program then you will see the 200 blue squares PRINTed out after being stored in the variable A$. From this we can work out that the maximum number of characters that we can store in a string is 200.

So what do we do when we need to store more than 200 characters in a string? The answer is fairly simple. We CLEAR some extra memory so that we can store more characters in our variables.

To illustrate how the CLEAR command is used type in this short program:

```
10 CLEAR 1000
20 A$= STRING$(250,175)
30 B$= A$:C$= A$:D$ = A$
50 PRINT A$;B$;C$;D$
```

When you RUN this program each of the variables A$, B$, C$ and D$ will have 250 blue squares stored in them. The contents of these variables are then PRINTed by line 50.

However, if you add this line:

```
40 E$=A$
```

and then RUN the program you will receive an OS error because we are trying to store more than 1000 characters in our variables (each of the variables A$, B$, C$ and D$ contains 250 characters, 4*250 = 1000 characters).

We can, however, still store the standard 200 characters in E$ (and all the other variables for that matter).

One thing to remember. We can still only store a maximum of 255 characters in one string variable, even using the CLEAR command, but these extra characters still come in handy sometimes.

35
High resolution graphics

Low resolution graphics can produce quite good effects and can liven up your programs quite a lot. These types of graphics, however, are chunky and leave a lot to be desired. The Dragon can produce much better graphics, but with the sacrifice of some colour. Low resolution graphics use a screen size of 64 squares across by 32 squares down, but high resolution graphics can go up to 256 squares across and 192 down.

This section covers all the high resolution graphics commands, some of which will be quite similar to the low resolution graphics commands.

PCLEAR

Before we start drawing pictures in high resolution we need to tell the computer how many different pictures we are going to draw. This is because on the Dragon it is possible to draw pictures on different 'pages' and then swap them about quickly (like a flick book).

If you don't need to swap the pictures about then you only need to reserve enough memory for one screenful of pages. To reserve memory for these pages we use the PCLEAR command followed by the number of pages that we want.

For example, to reserve memory for three pages we use the command:

```
10 PCLEAR 3
```

If you do not specify how many pages you want to use the Dragon reserves enough memory for four pages.

Reference chart

Listed below are the graphics modes available on the Dragon, the size of the points in each mode, the number of pages needed and the colour sets available with each mode. When writing your own graphics program you can look back at this table to help you work out how many pages you need to reserve for each mode, and which mode you want to use.

Graphics Modes

PMODE	SIZE OF POINT	No. OF PAGES	COLOURS AVAILABLE SCREEN 1,0	SCREEN 1,1
0	■■ ■■	1	0,1	0,5
1	■■ ■■	2	1,2,3,4	5,6,7,8
2	■■	2	0,1	0,5
3	■■	4	1,2,3,4	5,6,7,8
4	■	4	0,1	0,5

PMODE

Now that we know how to reserve memory for our pictures we need to know how to choose the graphics mode that we want to work in.

There are five different high resolution graphics modes available on the Dragon, each one slightly different from the others. We choose the graphics mode, and also which page we want to start drawing in with the PMODE command.

For example, if you wanted to work in the highest graphics mode (which only allows two colours but has a screen size of 255 by 192 squares) and we wanted to start drawing in the first page we would use a line like this:

```
30 PMODE 4,1
```

SCREEN

When you use high resolution graphics you have a choice between two different colour modes. In some modes you are allowed green and black or buff and black, and in others you are allowed blue, red, green and yellow or magenta, orange, buff or cyan.

To swap between these colour modes we use the SCREEN command. When we use the screen command we have to state whether we are using text or graphics and which colour set we want (yes you can swap the colour set for text!).

An example of the SCREEN command is this line:

```
50 SCREEN 1,0
```

This example line tells the computer that we want to use graphics and that we want the first colour set.

PCLS, PSET, PRESET and PPOINT

PCLS, PSET, PRESET and PPOINT are the high resolution equivalents to CLS, SET, RESET and POINT. As we already know how to use them for low resolution graphics we won't go into their high resolution use in great detail.

The PCLS command can be followed by a number which tells the computer which colour you want the background to be.

The background colours which you are allowed vary with each graphic mode, as you can see if you look in Appendix B.

PSET is used in exactly the same way as the SET command, that is, it is followed by three numbers in brackets — the coordinates of the point to be SET and the colour that you want it to be.

Although each graphics mode has 255 squares across and 192 squares down the size of the point which is SET varies. To demonstrate this try the following program:

```
10 MODE0,1
20 SCREEN 1,0
30 PCLS 3
40 PSET(128,96,2):GOTO 40
```

When you RUN this program you will see a black dot appear in the middle of the screen.

If you change the 0 in line 10 the dot will vary in size and colour, but will always stay in the same place.

If you look at the table on page 90 you will see a table of the different features which go with each graphics mode.

The size of point column shows you the shape of the points that you can SET in each mode.

If you look at the no. of pages column you will see that some modes need more pages than others, so you must always remember to use the PCLEAR command to reserve enough pages for the mode you want to use.

We haven't reserved any pages in the above program as the Dragon will automatically reserve 4 pages which is enough for our purposes.

The PRESET command is used in exactly the same way as its low resolution equivalent, as is the PPOINT command.

The following program fills the PMODE 1 screen with randomly coloured dots and then deletes all the dots, colour by colour:

```
10 PMODE 1,1
20 SCREEN 1,0
30 PCLS
40 FOR N=0 TO 255 STEP 2
50 FOR M=0 TO 191 STEP 2
60 C=RND(4)
```

```
 70 PSET(N,M,C)
 80 NEXT M
 90 NEXT N
100 FOR C=2 TO 4
110 FOR N=0 TO 255 STEP 2
120 FOR M=0 TO 191 STEP 2
130 IF PPOINT(N,M)=C THEN PRESET (N,M)
140 NEXT M
150 NEXT N
160 NEXT C
170 GOTO 170
```

All of the commands in the program on the previous page will be familiar to you so you should be able to see how the program works.

Lines 40, 50, 110 and 120 may puzzle you because of the STEP 2 at the end of each line. These commands are added because of the shape of the points that we are SETting (see page 90).

Try taking the STEP 2 off the end of these lines and see what happens.

The PCLS, PSET, PRESET and PPOINT are only a few of the high resolution graphics commands available to you. There are many more commands which are a lot more versatile and useful, but which are also harder to use. For this reason it would be advisable if you spent some time experimenting with these high resolution commands to get the hang of them properly.

LINE

The LINE command, as you may expect, draws a line between any two points on the screen. Try this short example program:

```
10 PMODE 3,1
20 SCREEN 1,0
30 PCLS
40 LINE (0,0) – (255,0),PSET
50 GOTO 50
```

When you RUN this program you will see a red line appear across the top of the screen. Line 40 tells the computer to draw a line from the square 0 squares across and 0 squares down to the square 255 squares across and 0 down. The PSET command at the end means that we want the line to be drawn. If we wanted to delete a line we would simply replace the PSET with PRESET.

The LINE command can also be used to draw rectangles. Try changing line 40 to:

```
40 LINE (0,0)–(255,191),PSET,B
```

When you RUN the program this time you will see a line drawn around the edge of the screen. What we have told the computer to do this time is to draw a box with one corner in the square 0 squares across and 0 squares down and the opposite corner 255 squares across and 191 squares down.

The LINE command can fill in boxes as well as draw them. Try changing line 40 to:

```
40 LINE (0,0) – (255,191),PSET,BF
```

The program will now fill in a box which covers the whole screen.

To change the colour of the line that we are drawing in we use COLOR command (yes COLOR not COLOUR, for some reason the Dragon uses the American spelling). The COLOR command is followed by the colour that you want to draw in and the background colour that you want, eg if you wanted to draw in red with a green background you would use the command COLOR 3,1.

PAINT

The PAINT command does just what its name suggests — it PAINTs parts (or all) of the screen in different colours. All we have to do is tell the computer where we want it to start PAINTing, what colour we want it to PAINT in and which colour we want it to stop at. This short program gives an example:

```
10 PMODE 3,1
20 SCREEN 1,0
30 PCLS
40 COLOR 2
50 LINE(90,90) – (140,90),PSET
60 LINE – (115,60),PSET
70 LINE – (90,90),PSET
80 PAINT(96,86),3,2
90 GOTO 90
```

Lines 60–70 may confuse you slightly because we have left off the coordinates of the starting square. The Dragon does allow this and if we do leave off the first coordinates it automatically starts the line from the last point drawn. We could replace these lines with:

```
60 LINE(140,90) – (115,60),PSET
70 LINE(115,60) – (90,90),PSET
```

This version obviously takes up much more memory, so it is best to use the first version above.

Line 80 contains the PAINT command which fills in the triangle. The first two numbers in brackets tell the computer where we want to start PAINTing from (number of squares across and number of squares down). The third number tells the computer which colour you want to PAINT in. The final number is the colour which the PAINTing must stop at, in other words the colour that the shape was drawn in.

One thing that you must always remember is *never* to leave a gap in the shape to be PAINTed because otherwise the colour leaks out and fills the whole screen.

CIRCLE

The CIRCLE command is probably the most complicated command that we have met so far, but don't let that worry you. With a bit of practice you should be able to draw circles, ellipses and arcs with no trouble at all.

The first thing to learn is how to draw a simple circle using the CIRCLE command. Try typing in this program:

```
10 PMODE 3,1
20 SCREEN 1,0
30 PCLS
40 CIRCLE (128,96),20,2
50 GOTO50
```

If you look at line 40 you will see how the CIRCLE command is used in its simplest form.

The two numbers in brackets are, as you may expect, the coordinates of the centre of the circle. The third number (20) is the radius of the circle (the distance from the centre of the circle to the edge) and is measured in squares. The final number is the colour of the circle.

Now that we can draw a circle we can find out how to draw ellipses or 'stretched' circles. If you make this alteration to line 40 you will be able to see what is done:

```
40 CIRCLE (128,96),20,2,2
```

This makes the ellipse twice as high as it is wide. Changing line 40 to:

```
40 CIRCLE (128,96),20,2,.5
```

makes the circle half as high as it is wide.

Now that we can draw circles and ellipses we can go on to draw arcs. All we have to do is add the start and end points at the end of the CIRCLE command. The start point is from 0 to I (with 0 being at 3 o'clock) and the end point is also from 0 to 1 (with .5 being 9 o'clock). Try changing line 40 to:

40 CIRCLE (128,96),40,2,1,0,.5

Try varying the 0 and .5 at the end to see what effects they produce.

The following program illustrates the LINE, CIRCLE and PAINT commands (with a bit of SOUND here and there as well):

Modern Art

```
 10 PMODE 3,1
 20 SCREEN 1,0
 30 PCLS
 40 FOR N=0 TO RND(20)+20
 50 C=RND(4)
 60 COLOR C,1
 70 X=RND(255):Y=RND(191)
 80 A=X+RND(100):B=Y+RND(100)
 90 IF A>255 OR B>191 THEN GOTO 70
100 LINE(X,Y)-(A,B),PSET,BF
110 SOUND RND(255),1
120 NEXT N
130 FOR N=0 TO RND(20)+10
140 C=RND(4)
150 X=RND(255):Y=RND(191):R=RND(40)
160 CIRCLE(X,Y),R,C
170 PAINT(X,Y),RND(4),C
180 SOUND RND(255),1
190 NEXT N
200 FOR N=1 TO 3000
210 NEXT
220 GOTO 10
```

PCOPY

Earlier on in this book we said that it was possible to flick through pages on the screen to produce a flick-book effect. As you know, each different graphics mode needs a different number of pages with PMODE 0 needing one page, PMODEs 1 and 2 needing two pages and PMODEs 3 and 4 needing four pages. It is possible to take any page and put it on top of any other page, and as we have up to eight pages we can produce simple animated pictures using this method. When a graphics mode needs more than one page the screen is divided up in this way:

PMODEs 1 and 2:	**PAGE ONE**
	PAGE TWO
PMODEs 3 and 4:	**PAGE ONE**
	PAGE TWO
	PAGE THREE
	PAGE FOUR

If you look at these diagrams of how the pages are placed one above the other you will be able to see that in PMODEs 1 and 2 page one occupies the top half of the screen and page two occupies the bottom half of the screen. PMODEs 3 and 4 are similar except each page occupies a quarter of the screen.

Putting one page on top of another is very simple. All you have to do is tell the computer which page you want to put on top of another. To do this we use the PCOPY command. For example, if we wanted to put page four on top of page one we would use a line like this:

50 PCOPY 4 TO 1

We can, of course, take a page off the screen or put a page which was off the screen onto it using the same method (remember we do have eight pages to work with).

The following program shows how to use the PCOPY command to copy pages onto and off the screen:

Page Swap

```
 10 PCLEAR 6
 20 PMODE 3,1
 30 SCREEN 1,0
 40 PCLS
 50 LINE(0,0)-(255,47),PSET,BF
 60 COLOR 3
 70 LINE(0,48)-(255,95),PSET,BF
 80 COLOR 2
 90 LINE(0,96)-(255,143),PSET,BF
100 COLOR 4
110 LINE(0,144)-(255,191),PSET,B
120 FOR N=0 TO 2000:NEXT
130 PCOPY 3 TO 5
140 PCOPY 4 TO 6
150 PCOPY 1 TO 4
160 FOR N=0 TO 500:NEXT
170 PCOPY 6 TO 1
180 FOR N=0 TO 500:NEXT
190 PCOPY 2 TO 3
200 FOR N=0 TO 500:NEXT
210 PCOPY 5 TO 2
220 FOR N=0 TO 500:NEXT
230 GOTO 130
```

When you RUN this program the screen will be filled with three different coloured bands and a green band with a red line around it (one band to each page). After a short pause the bands will begin to change their order and then keep changing their order until you stop the program. The main part of the program is lines 130–230 which transfer the pages on and off the screen. You may wonder why we have to copy pages 3 and 4 off the screen. This is because if we don't keep a copy of them off the screen we will eventually end up with only two colours.

DRAW

The DRAW command is the most versatile of all the high resolution graphics commands. Like the PLAY command, the DRAW command can be followed either by a string variable containing a series of commands, or a series of commands enclosed in quotation marks.

The commands within the quotation marks (or string variable) are very easy to use and remember. For example, if we wanted the computer to draw a line 10 squares long going straight up we would use a line similar to this:

```
100 DRAW"U10"
```

If you look at this line you will see that U means Up. In the same way R means Right, L means Left and D means Down.

After each direction command we tell the computer how many squares we want it to draw in that direction.

Try the following short program:

```
  10 PMODE 3,1
  20 SCREEN 1,1
  30 PCLS
  40 DRAW"BM20,180;R215;U100;L215;D100"
1000 GOTO 1000
```

When you RUN this program you will see a box drawn on the screen. The DRAW command in line 40 draws this box by drawing a line Right 215 squares, Up 100 squares, Left 215 squares and Down 100 squares.

The BM20,180 command tells the computer to move to the position 20 squares across and 180 squares down *without* drawing a line.

We can now add to the program to make the box look a bit like a house:

```
50 DRAW"U100;E50;R114;F50"
```

This line incorporates two new commands which can be used with DRAW — E and F. E means 'draw a line at 45 degrees' and F means 'draw a line at 135 degrees'.

As well as E and F, there are two other diagonal drawing commands — G and H. G means 'draw a line at 225 degrees' and H means 'draw a line at 315 degrees'.

Let's add some windows to our house:

```
60 DRAW"BM25,85;R40;D30;L40;U30"
70 DRAW"BM230,85;L40;D30;R40;U30"
80 DRAW"BM25,175;R40;U30;L40;D30"
90 DRAW"BM230,175;L40;U30;R40;D30"
```

The windows look a bit plain don't they? If we wanted to split each window into four panes with the wood between each pane coloured cyan we would have to change the draw colour. To do this we use the C for Colour command:

```
100 DRAW"C3;BM45,86;D28;U14;L18;R36"
110 DRAW"BM210,86;D28;U14;L18;R36"
120 DRAW"BM45,146;D28;U14;L18;R36"
130 DRAW"BM210,146;D28;U14;L18;R36"
```

Now that we have windows in our house we need a door so that we can get in:

```
140 DRAW"C4;BM110,180;U40;R36;D40"
```

And a letter box:

```
150 DRAW"BM122,165;R12;D4;L12;U4"
```

Finally we can add a bit of colour:

```
160 PAINT(150,60),2,4
170 PAINT(22,178),4,4
180 PAINT(112,178),3,4
```

Our house is now finished (unless you want to add a chimney that is) but there are still a few commands which we haven't learnt yet. The first of these is the Scale command.

Try typing in this short program:

```
10 PMODE4,1
20 SCREEN 1,1
30 PCLS
40 DRAW"BM128,96;U5;D10;U5;L5;;R10"
50 GOTO 50
```

This program draws a small cross in the centre of the screen. If we wanted to make the cross twice as big without re-drawing it we would add this line:

```
35 DRAW "S8"
```

The Scale command enlarges drawings in quarters of their actual size. For example, Scale 8 is eight quarters (which is two) and makes the drawing twice as big. In the same way Scale 12 (which is twelve quarters or 3) makes a drawing three times as big, and Scale 1 (or one quarter) makes a drawing a quarter of its actual size.

Try putting different numbers between 1 and 62 after the Scale command in line 35.

Another thing which the DRAW command can do is rotate a picture. The A command is followed by a number between 0 and 3 which corresponds to the angle that you want the drawing rotated through, as below:

0 = 0 degrees	1 = 90 degrees
2 = 180 degrees	3 = 270 degrees

Try entering the following program:

Rotating Penant

```
 10 PMODE 4
 20 SCREEN 1,1
 30 PCLS 1
 40 COLOR 2
 50 DRAW"S8"
 60 A$="BM120,104;L12;E6;F6;R12"
 70 DRAW A$
 80 FOR N=0 TO 500:NEXT
 90 PCLS 1
100 DRAW"A1":DRAW A$
110 FOR N=0 TO 500:NEXT
120 PCLS 1
130 DRAW"A2":DRAW A$
140 FOR N=0 TO 500:NEXT
150 PCLS 1
160 DRAW"A3":DRAW A$
170 FOR N=0 TO 500:NEXT
180 PCLS 1
190 DRAW"A0":DRAW A$
200 GOTO 80
```

When you RUN this program you will see a little penant rotating about the point where the stick ends and the triangle starts.

If you look through the program then you will see that we have stored the drawing commands in the string A$, in the same way as we did with the PLAY commands. This saves us having to retype these commands each time we draw the penant.

Lines 100–190 rotate the penant and redraw it before pausing, clearing the screen and rotating the shape another 90 degrees.

As with the PLAY command it is possible to eXecute a set of commands which are stored in a string variable using the X command. For example, in the last program we could change line 100 to:

100 DRAW"A1;S A$"

The **M** command which we used with the B command to determine the start point of our drawings can also be used to move to another point. For example, if we wanted to move left five squares and down 10 squares we could use a line like this:

100 DRAW"M−5,10"

To move left or up we put a − in front of the number of squares that we want to move. To move right or down we just specify how many squares we want to move, as in the above example.

If you want to move without drawing then you simply add a B before the direction command, eg.

DRAW"BR10"

moves the drawing cursor 10 squares to the right, without drawing.

One last thing about the DRAW command. Throughout this chapter we have been putting a semi-colon between each command, but this is not necessary. The only reason we have put these in is they make it easier to read the lines.

GET and PUT

In most programs using graphics you need to be able to move something around the screen. You could, of course, move the object by PRESETting it and then draw it again somewhere else, but this takes time, especially with a large drawing. A much faster (and easier) way is to GET the picture from somewhere on the screen and PUT it back where you want it.

Try the following program:

Bouncing Ball

```
 10 PMODE 4,1
 20 SCREEN 1,1
 30 PCLS
 40 DIM SHAPE(19,19)
 50 CIRCLE (10,10),8,1
 60 GET(0,0)-(20,20),SHAPE,G
 70 X=1:Y=1
 80 A=2:B=2
 90 PUT (X,Y)-(X+20,Y+20),SHAPE,PSET
100 X=X+A:Y=Y+B
```

```
110 IF Y>170 OR Y<2 THEN B=-B
120 IF X>234 OR X<2 THEN A=-A
130 GOTO 90
```

This program makes a ball bounce about the screen.

Line 60 contains the GET command which takes the ball (which occupies the space from two squares across and two squares down to 18 squares across and 18 squares down — the space around it stops the ball from leaving a trail) and stores it in the array variable SHAPE. The G at the end of the line makes sure that the picture is recorded in great detail. If the G is left off the shape is sometimes not PUT back where you want it.

Line 40 DIMensions the variable which we store the circle in. In this case we have used an array variable with 20 rows and 20 columns because the circle has a radius of eight squares, giving it a diameter of 16 squares. Using a 20 by 20 array allows us to GET two blank squares all round the circle so that when the circle is moved the blank squares rub out any trail left behind it.

However, unmentioned (or perhaps unknown) by other manuals is the fact that it is possible to store your shape in a simple one — dimensional array. This has the advantage that less memory is used in storing shapes, so more and larger shapes can be stored. Take our circle for example. All we have to do is to work out the area of screen which the circle takes up (20 squares by 20 squares, multiply 21 by 21 which gives 441 squares) and divide this number by 8, which gives 55.125, which we round up to 56. We then divide this number by 5, which gives 11.2, which we round up to 12. So all we need is a one dimensional array to contain 12 numbers. Try changing line 40 to:

40 DIM SHAPE(11)

Here is a table showing what to divide the area of your shape by in each graphics mode:

PMODE	1st DIVISOR	2nd DIVISOR
4	8	5
3	8	5
2	16	5
1	16	5
0	32	5

As you can probably see, this method of using a one — dimensional array saves a lot of memory (429 bytes of memory in this case), so it is well worth remembering.

Line 90 PUTs the shape back on the screen. Notice that the PUT command ends in a PSET command. This can be replaced with the

PRESET command which will delete a shape. AND, OR and NOT can also take PSET's place.

AND compares each point of the shape with the part of the screen that it will occupy. If both the point of the shape AND the point on the screen which it will occupy are SET then that part of the shape will be SET. If either or both of them are not SET that part of the shape will not be SET.

OR compares the points in the same way as AND but will SET the part if either or both of the points are SET. This makes it look as if the shape is on top of whatever is behind it.

NOT is much simpler than AND and OR. This command reverses the colour of the background when a shape is PUT onto it.

Try replacing the PSET at the end of line 90 with AND, OR and NOT and see what effects they produce.

We have now covered all the high resolution graphics commands. With some practice you should be able to produce quite effective displays which will liven up your programs no end. It would be a good idea if you did practise with high resolution graphics before you go on to the next chapter as these commands take a while to get used to.

36 PRINT USING

The PRINT USING command is used to neaten up your displays and is very useful for displaying tables. There are several different ways of using the PRINT USING command and each one is explained below.

The # symbol is used to round up numbers. For example, if you wanted to round up the number 123.456 to two decimal places you would use:

```
PRINT USING"###.##";123.456
```

In this command we have enclosed three # symbols, a full stop and another two # symbols in quotation marks after the PRINT USING command. This tells the computer that we want the following number (123.456) displayed with three numbers before the decimal point and two numbers after.

If we wanted to display a number with a comma to the left of every third number (eg 1,000,000) we would use this command:

```
PRINT USING "#######,";1000000
```

The comma at the end of the # symbols means that we want a comma after every third number.

The ** symbols tells the computer to fill up any spaces before a number with asterisks. For example, the following command displays the number 235 with three asterisks before it:

```
PRINT USING"**####";235
```

The $ symbol is used to display a dollar sign before a number (to represent money). All you have to do is put the $ symbol at the start of the PRINT USING command:

```
PRINT USING"$#####.##";123.45
```

If you want the $ symbol directly before the number with no spaces in between you should use this kind of command:

```
PRINT USING"**$#####.##";123.45
```

The + sign tells the computer to display a number together with a plus or minus sign, depending on whether the number is positive or negative. The plus or minus sign can either be before or after the number, depending on where you put the + symbol.

PRINT USING"+ # # #";123

(displays the + before the number)

PRINT USING" # # # + "; – 123

(displays the – after the number)

If you want to display a number in exponential form (eg 1.34E+ 06 is an exponential number and means 1.34 times 10 to the power of 6) you should add four ↑ symbols at the end of the PRINT USING command. To display 9876 in exponential form you would use:

PRINT USING" # # # #↑↑↑↑";9876

The ! symbol is used to display only the first character of a string, eg to display the first letter of the string TOTAL you would use:

PRINT USING"!";"TOTAL"

If you wanted to make sure that a string of letter didn't take up more than a certain amount of space on a table you would use the % symbol with the PRINT USING command. So to make sure that the string 'TOTAL' doesn't exceed three characters you would use:

PRINT USING"% %";"TOTAL"

This command will display TOT (first three characters of the string "TOTAL") as there is one space between the two % symbols, and each % symbol counts as a space.

The PRINT USING command can, of course, be used with variables instead of numbers at the end. The only reason we have used numbers in our examples is to allow you to try each one and see what effect it has.

37
Storing information on tape

As well as recording programs on tape it is also possible to store numbers and letters on the tape. To do this we PRINT the numbers and letters onto the tape and then INPUT them back again.

Let's look at the program which we used to check the volume level on the tape:

```
10 OPEN"O", #-1,"FILE"
20 FOR N = 0 TO 2000
30 PRINT#-1,65;
40 NEXT
50 CLOSE#-1
```

Line 10 of this program tells the computer to open an Output file (the 'O' in quotation marks stands for Output) on the tape and give it the name 'FILE' (in the same way as you give a program a name).

Line 30 tells the Dragon to store a number 65 on the tape (the semi-colon makes sure that the next number or letter is stored directly after the last one without too much of a gap left in between, the same way as a semi-colon does when you PRINT onto the screen).

Line 50 tells the computer that you have now finished with the tape and to close the file.

Reading information back off the tape is just as easy as storing it. If you look at the program on the next page which reads the numbers back off the tape you will see how this is done:

```
10 OPEN"I", #-,"FILE"
20 FOR N = 0 TO 2000
30 INPUT#-1,A
40 PRINT CHR$(A);
50 NEXT
60 CLOSE#-1
```

Line 10 of the first program is similar to the start of the program above, except in this case we are telling the computer to open a file to Input (the 'I' in quotation marks stands for Input) information from the tape. Line 30 actually INPUTs a number and stores it in the variable A, before line 40 displays the character with the CHR$ code 'A'.

Line 60 tells the computer that we have now finished with the tape.

Storing and reading back letters is just as simple as storing and reading back numbers. The letter which you wish to store must be enclosed in quotation marks (in the same way as you do with a normal PRINT command) and must be read back into a variable.

Variables can also be stored simply by using a command like this (after opening a file for Output of course):

PRINT #-1,A

Reading in variables is just the same as reading in a number, as it is only the contents of the variable which are recorded, not the variable itself.

It is possible to check to see if you have reached the end of the information stored on the tape by using the EOF(-1) command. For example, to tell the computer that you have finished with the tape when it gets to the end of the information you would use a line like this:

80 IF EOF(-1) THEN CLOSE #-1

It is possible to store a long list of numbers, characters or strings on tape simply by placing a comma or semi-colon in between each one. A semi-colon stores the information close together, therefore saving time and space on the tape.

If you need to store graphics characters on tape then you need to store their ASCII codes on tape. Try this program.

Graphics Recorder

```
10 CLEAR 256
20 FOR N=128 TO 255:A$=A$+CHR$(N):NEXT N
30 PRINT A$
40 OPEN"O",#-1,"GRAPHICS"
50 FOR N=1 TO LEN(A$)
60 PRINT#-1,ASC(MID$(A$,N,1));
70 NEXT N
80 CLOSE#-1
```

Put a blank tape in your tape recorder and set it on record. Then RUN the program. You should see the set of graphics symbols appear on the screen and then the tape will start.

When it has finished rewind the tape, set it on PLAY and RUN the program on the following page:

Graphic Loader

```
10 CLEAR 256
20 OPEN"I",#-1,"GRAPHICS"
30 INPUT#-1,A
40 A$=A$+CHR$(A)
50 PRINT CHR$(A);
60 IF EOF(-1) THEN CLOSE#-1:GOTO 60
70 GOTO 20
80 PRINT:PRINT A$
```

This program will load back the graphics symbols which were saved with the first program, displaying them as they are loaded, and then all in one go when they have all been loaded in.

You will see from these programs that data is recorded and loaded in blocks. The reason for this is the Dragon pauses every now and then when loading data to store it in the right memory location.

It is also possible to store music on the tape straight from the Dragon. Try the following program:

Music Saver

```
10 CLS
20 MOTOR ON
30 PRINT"PLEASE SET TAPE ON record"
40 PLAY"O3L4CAGFL2CL6CCL4CAGFL2DL4B-AGL2EL4O+CC
  O-B-GL2A"
50 PRINT"PLEASE REWING TAPE AND SET ON   play"
60 AUDIO ON
```

As you will see from this program, if you use the PLAY command (or, to some extent, the SOUND command) while the tape is running on record the notes being PLAYed will be recorded.

38
Using a printer

The Dragon is capable of sending information to a printer as well as the screen. Any printer should work as long as it works from a parallel Centronics interface, but it is best to ask if a printer is compatible with the Dragon before buying it.

The printer plugs into the PI/O socket on the left of the computer and the plug should be connected as shown in the Additional Information leaflet included with your Dragon

A program can be output to the printer simply by typing LLIST. Printing text on the printer is just as easy. All you have to do is add #-2 to the PRINT statement. So to print the word HELLO on the printer you would use PRINT #-2,"HELLO". Remember that it is possible to print lowercase letters by including inverse characters in your PRINT statement.

It is also possible to find the position of the print head. This is done with the POS(-2) command.

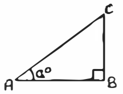
The following example prints all the normal text characters on the printer in columns 32 characters wide:

```
10 FOR N=32 TO 127
20 PRINT #-2,CHR$(N);
30 IF POS(-2)=32 THEN PRINT #-2)""
40 NEXT
```

All the PRINT USING functions work on the printer, as does the TAB command (in this case you have many more columns so the number after the TAB command can be much bigger). PRINT @ does not work on printers.

39
Trigonometric functions

Look at this triangle:

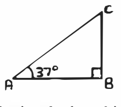


As you can see it is a right-angled triangle because one of its sides (BC) is at an angle of 90 degrees from one of the others (AB). If we know the length of the line AC, and also what the angle a is, then it is possible to work out the lengths of the sides AB and BC. To do this we need to use the sines and cosines.

The sine of an angle is the length of the opposite side divided by the length of the longest side (hypotenuse). As we are using the angle a the opposite side is BC. If we say that BC is 30 units long, and the hypotenuse is 50 units long then the sine of the angle a is 0.6.

The cosine of an angle is the length of the adjacent side (in this case AB) divided by the length of the hypotenuse. In our example the adjacent side is 40 units long. 40 divided by 50 is 0.8.

Now that we know what sines and cosines are we can put them to use. Look at this second triangle:



Again it is a right-angled triangle, but this time one of the angles is labelled and we are told the length of the hypotenuse. We can use these two pieces of information, together with your Dragon, to work out the lengths of the other two sides. Let's take side BC first.

BC is the side which is OPPOSITE to the angle which we know, so we need to use sine to work out its length. Your Dragon knows the sine of every angle, so all we have to do is ask for it. Unfortunately the Dragon asks for angles to be given in RADIANS which are measurements of angles in circular units. To convert angles to radians we need to divide the number of degrees by 57.2957805. Using this information we can find the length of the opposite side in this way:

```
PRINT SIN(37/57.2957805)*50
```

This produces the answer 30.0907507 which is the length of the opposite side. We multiply the sine of 37 degrees by 50 (the length of the hypotenuse) to find the length of the opposite side.

Finding the length of the adjacent side is very similar. In this case we use cosine instead of sine and use the same routine:

```
PRINT COS(37/57.2957805)*50
```

This produces the answer 39.9317759 which is the length of the adjacent side.

The tangent of an angle is the ratio between the length of the opposite side and the adjacent side. For example, the tangent in our example is 0.753554031 which is 30.0907507 divided by 39.9317759. On the Dragon we find tangents with the TAN command, so in this case we use:

```
PRINT TAN(37/57.2957805)
```

remembering to convert from degrees into radians which gives the answer 0.753554031.

If you already know the ratio between the opposite and adjacent sides of a triangle you can work out the angle by using ATN, which is the inverse of TAN. If you type:

```
PRINT ATN(.753554031)*57.2957805
```

you get the answer 36.9999999, which is as near to 37 as you could possibly get.

40
Numeric functions

We have already met some of the Dragon's numeric functions — RND, POINT, PPOINT, TAN, SIN, COS, PEEK and JOYSTK. There are, however, some others which we haven't met, and these are explained in alphabetical order over the next few pages.

ABS

ABS is used to convert a negative number into an ABSolute, or positive number:

PRINT ABS(-12)

Returns the answer 12. If a number is already positive then it remains the same. ABS can also be used with variables:

PRINT ABS(A)

returns the positive value of the number stored in the variable A.

EXP

EXP is used to raise e (base of natural logarithms) to the power of any number:

PRINT EXP(3)

will cube e, giving the answer 20.0855369

FIX

FIX is a little-used function which truncates, or cuts off all of a number after the decimal point:

PRINT FIX(12.3625;INT(-12.3625)

returns the answers 12 and -12.

INT

INT is similar to FIX with the difference being that it rounds a number down, rather than just truncating it:

PRINT INT(12.3625);INT(12.3625)

will return the answers 12 and -13.

LOG

LOG is the inverse of EXP and is used to find natural logarithms:

PRINT LOG (15.5)

returns the answer 2.74084003

MEM

Have you ever wondered just how much memory you have left for your programs? Well, the MEM command is for just that purpose. Try typing in:

PRINT MEM

If you do not have a program in memory then you will receive the answer 24871, which is just over 24K of memory (divide by 1024 to find total memory in kilobytes).

While we are on the subject of memory we might just as well mention why we can only use 24K of the 32K we are supposed to have on the Dragon. 1024 bytes (or 1K) of memory are used by the Dragon for remembering such things as which line is currently being carried out. Another 512 bytes are used by the screen (that's how you can POKE onto the screen), and another 6K are used for the high resolution graphics pages (up to 12K can be set apart for this use). The other 512 bytes are used for various control lines, and also for variable storage.

Although it is not possible to use any of the text screen memory, or any of the other memory essential to the Dragon when it is working, we can use the high resolution graphics pages. To do this we type PCLEAR 1, which gives us another 4.5K. We can access the other 1.5K by typing in:

POKE 25,6:POKE27,6:POKE29,6:POKE31,6

We now have 31015 bytes (or just over 30K) left for programs, although we can not now use high resolution graphics.

POS

The POS command can be used to tell the POSition of the printer head. If you type:

PRINT POS(-2)

You will be told how far across the printer head is (if, of course, you have a printer connected).

SGN

SGN is used to find out if a number is positive or negative. If the number is positive then you will receive the answer 1. If it is negative you will receive the answer -1. If the number is 0 you will receive the answer 0:

PRINT SGN(10);SGN(-10);SGN(0)

SQR

SQR is used to find the square root of a number:

PRINT SQR(25)

will give you the answer 5.

Appendices

APPENDIX A
A list of commands

This chapter gives a brief run-down of every command which is available on the Dragon. Some of them, such as the mathematical functions, have not been covered in the rest of this book so it is worth reading through this section:

ABS(n) — this function converts the number n (n can be any number) to its positive value: eg PRINT ABS(-12) will display the number 12.

AND — this function has three uses:
(i) In the IF. . . THEN. . . ELSE statement AND can be used in this way:

IF A= 1 AND B= 1 THEN C = 1

(If the value of the variable A is 1 AND the value of the variable B is I then store the number 1 in the variable C)
(ii) AND can be used with the PUT command to compare each point of a shape with the part of the screen that it will occupy. If both are SET before the shape is PUT onto the screen the shape will be visible, otherwise it will be invisible.
(iii) AND is also a LOGICAL OPERATOR. It can be used to compare two numbers in a way similar to the way AND is used with the PUT command. If both the numbers are TRUE (1) then the result will be true. If either of the numbers are false (0) the answer will be false. PRINT 1 and 1 returns the number 1 but PRINT 1 AND 0 returns the number 0.

ASC(''c'') — this function returns the ASCII (American Standard Code for Information Interchange) code of any character, eg PRINT ASC(''A'') returns 65. The character *must* be enclosed in quotation marks and brackets.

ATN(n) — returns the arctangent of the number n (n can be any number).

AUDIO ON — re-routes the sound from the cassette recorder through the television speaker.

AUDIO OFF — cancels AUDIO ON.

CHR$(n) — used with the PRINT command to display the character with the code n (n is any number between 31 and 255).

CIRCLE(x,y),r,c,hw,s,e — this command is used to draw circles and ellipses in high resolution graphics. Variable x is the number of squares across that you want the centre of the circle to be and y is the number of squares down that you want the centre of the circle to be. The variable r is the radius of the circle and c is the colour; hw is the height-width ration of the circle, s is the start point of the circle (or arc) and e is the end point (from 0 to 1).

CLEAR v,s — reserves extra memory for string variables. Variable v is the number of characters that you want and s (optional) sets a limit to the amount of memory that your programs can take up.

CLOAD"program" — loads a program from the cassette tape into the Dragon's memory. The 'program' is the name that the program was saved under.

CLOADM"program",o — similar to CLOAD but loads machine code programs. Typing o allows you to load the program into a specific part of the RAM memory different to the one that it was originally saved from (optional).

CLOSE#-1 — closes cassette file.

CLS n — clears the screen to the colour n (n is any number from 0 to 8). If n is left out the screen is cleared to green. Colour codes are:

0 — Black	1 — Green	2 — Yellow	3 — Blue
4 — Red	5 — Buff	6 — Cyan	7 — Magenta
8 — Orange			

COLOR b,f — used in high resolution graphics to define the background and foreground colours. See Appendix B for colours available in each mode.

CONT — continues a program from where it was stopped. It only works when the program has been stopped using the BREAK key or a STOP command in the program. It does not work after more lines have been entered or an error has occurred.

COS(n) — returns the cosine of n (n can be any number).

CSAVE"program" — saves a copy of the program currently in memory onto cassette tape. The program is the name which the program is saved under and can be any string of up to eight characters starting with a letter.

CSAVE"program",A — saves a program in ASCII format, allowing it to be read into a program like a normal data file.

CSAVEM"program",start,end,entry — similar to CSAVE but saves machine code programs. The start is the starting point in memory that the program is stored in and end is the end point. The entry is the memory address that the program is carried out in.

DATA a,b,c,d. . . — stores a list of numbers or characters. Each number or string of characters should be separated by a comma. Characters must be enclosed in quotation marks.

DEF FNa(d) = formula — defines a function with the nameFNa(d); a can be any variable, d is a dummy variable and formula is the calculation that you want carried out.

DEF USRn = address — defines the starting address of a machine code routine which is called by the USR function. Variable n is any number between 0 and 9. The address is the memory address of the start of the routine and must be from 0 to 65535.

DEL start-end — deletes several lines of a program: start is the line to be deleted and end is the last line. All lines between start and end are deleted.

DIM variable(n1,n2) — reserves memory for an array with the name variable with n1 rows and n2 columns. The n2 may be left out to reserve memory for an array with the name variable to store nl numbers.

DRAW "sub-commands" — used for drawing in high resolution graphics. The 'sub-commands' can either be a series of sub-commands enclosed in quotation marks or stored in a string variable. See pages 73–76 for an explanation of sub-commands.

EDIT line number — enters EDITOR and allows alterations to be made to program lines. See page 37 for a list of ways to alter lines.

EOF(-1) — checks to see if there is any more information left in a tape file.

EXEC address — executes machine code routine starting at memory location address.

EXP(x) — raises natural logarithms to the power of x.

FIX(x) — cuts all numbers after the decimal point off.

FOR v = n1 to n2 — start repeating all commands between here and the NEXT command n2 — nl times (eg if nl is 2 and n2 is 13 then the commands are carried out 13 – 2 = 11 times). The variable v is used as a counter.

GET (x1,y1)-(x2,y2),array name,G — high resolution graphics command. Stores rectangles of screen from the point xl squares across and yl squares down t o the point x2 squares across and y2 squares down i n the two dimensional array name. The G makes sure that every detail of the rectangle is recorded.

GOSUB line number — goes to the start of a subroutine and carries on with the program from there.

GOTOline number — similar to GOSUB but goes to any line, not necessa rily the start of a subroutine.

HEX$(n) — converts n to its hexadecimal (or base 16) equivalent. The answer is returned in string form and can be stored in a string variable.

IF condition THEN action 1 ELSE action 2 — compares two or more variables or numbers. If a certain condition concerning the numbers of variables is fulfilled then action I carried out, otherwise action 2 is carried out.

**INKEY$** — scans the keyboard. It is possible to record which key is being pressed in a string variable (eg A$ = INKEY$)

INPUT"message";variable — displays the message enclosed in quotation marks (optional) before waiting for a reply which is then stored in the variable 'variable'.

INSTR(n,s$,t$) — searches through the string variable s$ for the string of characters t$, starting at the nth character of the string s$. This function will return the position of the first character of the string t$ in the string s$. If t$ is not in s$ the number 0 will be returned.

INT(n) — rounds the number n down to the nearest whole number.

JOYSTK(n) — returns the current position of the joysticks. Where n is a number between 0 and 3:
0 — current left-right position of right joystick
1 — current up-down position of right joystick
2 — current left-right position of left joystick
3 — current up-down position of left joystick

LEFT$(a$,n) — returns the first n characters of the string variable a$.

LEN(a$) — returns the length of the string variable a$.

LINE(x1,y1)-(x2,y2),command,BF — draws a line from the point x1 squares across and y1 squares down t o the point x2 squares across and y2 squares down. The command may either be PSET (draws the line) or PRESET (erases the line). Variables B and F are optional: B draws a rectangle with the points (x1,y1) to (x2,y2) as opposite corners, BF fills in the rectangle.

LINE INPUT"message";variable — similar to INPUT but allows commas and up to 255 characters to be entered.

LIST start-end — displays the program on the screen. The screen automatically scrolls if the program does not all fit. The start and end may be any line number, and either or both may be left out. See page 13.

LOG(n) — returns the natural logarithm n.

MEM — returns the amount of memory free for programs.

MID$(a$,n,c) — finds the first c characters after the nth character of the string a$.

MOTOR OFF — turns off the cassette motor.

MOTOR ON — turns on the cassette motor.

NEW — deletes the program currently in memory.

NEXT — marks the end of a FOR. . . NEXT loop.

NOT — used with the PUT function to PUT a shape onto the screen in its negative colour.

ON n GOSUB line numbers — goes to the line number which marks the start of the nth subroutine in the list line numbers. Each line number in the list should be separated by a comma.

ON n GOTO line numbers — similar to ON GOSUB but goes to any line number, not to the start of a subroutine.

OPEN "c", #-1,"filename" — opens a file with the name 'filename' on the tape. The c can either be O to output data to the tape or I to read data in.

OR — can be used in three ways:
(i) It can be used in the IF. . . THEN statement. If either or both of two conditions are fulfilled then the action is carried out.
(ii) It can be used with PUT to give an impression of one object being on top of another. Compares each point of the shape with the part of the screen that it will occupy. If either or both are SET then that point of the shape will be SET.
(iii) It can be used as a LOGICAL OPERATOR. Compares two numbers and if either or both are true (1) then the result will be true. If both are false (0) then the result will be false.

PAINT(x,y),c,b — fills in a shape starting at the point x squares across and y squares down. Variable c is the colour that the shape should be coloured and b is the border colour of the shape.

PCLEAR n — reserves n pages for use in high resolution graphics.

PCLS n — similar to CLS but is used in high resolution graphics.

PCOPY p1 TO p2 — copies the picture on page pl onto page p2

PEEK (address) — looks into a memory address and returns the number stored there.

PLAY "music" — plays the contents of 'music', a list of notes and sub-commands which are either stored in a string variable or enclosed in quotation marks.

PMODE mode, page — defines the graphics mode that we want to work in. Where mode is the number of the graphics mode (see Appendix B) and page is the page which we wish to start drawing in.

POINT(x,y) — tests the point at x squares across and y squares down to see whether it is SET or not. If the point is SET its colour will be returned.

POKE address,value — stores the number value in the memory address, 'address'. Only works in RAM.

POS(n) — returns the current print position. If n is 0 the current cursor position is returned. If n is -2 the current position of the printer head is returned.

PPOINT(x,y) — same as POINT but is used in high resolution graphics.

PRESET(x,y) — deletes point which is x squares across and y squares down in high resolution graphics.

PRINT — used to display characters on the screen.

PRINT @position,characters — displays the characters stated at the position stated.

PRINT #device number,characters — if device number is -1 then the stated characters are stored on the tape. If device number is -2 then the stated characters are printed on the printer.

PRINT USING"format symbols";number — displays the number or variable number in the format stated by the format symbols. See pages 103–104 for format symbols.

PSET(x,y,c) — high resolution graphics command. Lights up the point which is x squares across and y squares down in the colour c. Colours available vary from mode to mode (see Appendix B).

PUT(x1,y1(-(x2,y2),shape,command — copies the shape stored in the variable shape onto the area of screen from xl squares across and y1 squares down to x2 squares across and y2 squares down. The shape must have previously been stored in the variable shape by a GET command. The command may either be PSET, PRESET, OR, NOT or AND.

READ variable — reads a number or string of letters from a DATA statement. The number or string is stored in the variable 'variable'.

REM comments — allows you to add comments into your program (to remind you what a section does, for example). REM statements are ignored by the computer when the program is executed.

RENUM new,old,increment — renumbers a program's line numbers starting at line old replacing it with the line new and renumbering in steps of increment.

RESET(x,y) — similar to PRESET but only works in low resolution graphics.

RESTORE — sets the data pointer to the first piece of data in the first DATA statement.

RETURN — marks the end of a subroutine. Program goes back to the next command after the GOSUB command which sent the computer to that subroutine.

RIGHT$(v$,n) — returns the last n characters of the string variable v$.

RND(n) — returns a random number. If n is 0 then the random number is between 0 and 1. If n is morethan 1 then the random number is between I and n.

RUN line number— starts executing a program starting at the line number stated. If the number is left out the program is executed starting at the first line of the program.

SCREEN screen type,colour set — used to define which colour set you want. The screen type is either 0 or I, 0 for text and 1 for high resolution graphics. See Appendix B for colour sets available with each graphics mode.

SET(x,y,c) — similar to PSET but only works in low resolution graphics.

SGN(n) — used to find whether a number is positive or negative. If number n is negative -1 will be returned, if n is positive 1 will be returned and if n is 0 then 0 will be returned.

SIN(n) — returns the sine of the number n.

SKIPF"program name" — searches for the end of the program with the name program name on the tape.

SOUND p,d — plays a note with pitch p for duration d. Variables p and d can be any number betwen I and 255.

SQR(n) — returns the square root of the number n.

STEP n — used in a FOR. . . NEXT loop to increment the counter in steps of n.

STOP — stops execution of program. Allows program to be continued using the CONT command.

STR$(n) — stores the number n in a string variable. (eg A$= STR$(12) stores the number 12 in the variable A$).

STRING$(n,c) — returns a string of n characters with the code c.

TAB(n) — used with the PRINT command to specify the column which you want to start PRINTing in.

TAN(n) — returns the tangent of the number n.

TIMER — returns current value of built-in timer. Timer may be reset by the command TIMER= 0.

TROFF — turns off trace.

TRON — turns on trace. Constantly displays which program line is being carried out.

USR(n) — carries out the machine code routine which has the number n (previously set by DEF USR).

VAL(s$) — converts the first character of the string variable s$ to a number as long as the first character is a number. (eg PRINT VAL("1") displays the number 1).

VARPTR(v) — returns the memory address which the variable v is stored in.

APPENDIX B
Graphics information

Listed below are the graphics modes available on the Dragon, the size of the points in each mode, the number of pages needed and the colour sets available with each mode. There is also a list of the colour codes.

Graphics Modes

PMODE	SIZE OF POINT	No. OF PAGES	COLOURS AVAILABLE	
			SCREEN 1,0	SCREEN I,1
0	■■ ■■	1	0,1	0,5
1	■■ ■■	2	1,2,3,4	5,6,7,8
2	■■	2	0,1	0,5
3	■■	4	1,2,3,4	5,6,7,8
4	■	4	0,1	0,5

Colour Codes

(0) BLACK	(1) GREEN	(2) YELLOW	(3) BLUE
(4) RED	(5) BUFF	(6) CYAN	(7) MAGENTA
(8) ORANGE			

APPENDIX C

This section contains several programs which put to use all of the commands which you have learned throughout this book. The programs are designed to show off all the best points of your Dragon — the graphics and sound capabilities, as well as its powerful BASIC language.

The programs are a mixture of games, such as Hangman and Gallery, and useful programs, such as Revision Aid and Clock. There is also a selection of graphics programs such as Artist which allows you to draw pictures in any of the graphics modes, and 3-D Plot which draws a three-dimensional picture.

Each program comes with a description of what the program does and how to use it, as well as an explanation of how it works.

The description of the program will tell you what happens in each stage of the program and also tells you how to operate it. This section will also help you to make adaptions (where possible) to the program.

Following each program is a full explanation of how the program works, section by section. If you read this section thoroughly you should be able to fully understand how the program works.

Each program is a direct listing of the running program and should be typed in exactly as it is printed, no matter how strange it looks (especially with the Clock program).

Hangman

This game is a version of the old favourite, Hangman. The idea of the game is to work out what word the Dragon has picked. You do this by choosing a letter which you think may be in the word and then typing it in. If the letter which you choose is in the word then the computer will display that letter at the bottom of the screen where it should go. If the letter appears more than once in the word then it will be displayed in every place that it occurs.

If you get a letter wrong then the computer starts to draw a gallows on the screen. Each time you make a mistake another part of the gallows is added, and eventually a man is drawn, piece by piece, waiting to be hung. If you do not guess the word by the time the picture is complete the man is hung and you lose the game.

If you think you know the word then you may type it in and see if it is right. If you are correct then the screen clears and the man is displayed (without the gallows). The tune Born Free is then played to finish off the game.

To help you remember which letters you have tried, every time you enter a letter it is displayed at the top of the screen. The word which you are trying to guess is displayed at the bottom of the screen, with any letters which you haven't yet guessed being replaced by dots.

This program makes use of the low resolution capabilities of the Dragon, as well as the sound. You may extend the computer's vocabulary of words by adding more between lines 650 and 1000 using the DATA statement. Words should be entered in the same way as those in lines 590 to 650 and the number 27 in the RND statement in line 50 should be changed to the number of words which you have in the list.

```
10 REM HANGMAN
15 GOTO1080
20 CLS0
25 RESTORE
26 CLEAR
30 PRINT@0,"LETTERS USED:";
40 P=1
50 FORN=0 TO RND(27)
60 READ A$
70 NEXT
80 PRINT@416,STRING$(LEN(A$),46);
90 Q=0:PRINT@448,"":PRINT@448,"WHAT IS YOUR GUESS";
100 INPUT GUESS$
110 IF GUESS$=A$ THEN 1000
120 PRINT@TRIES+32,GUESS$;
130 P=INSTR(P,A$,GUESS$)
140 IF Q=1 THEN 160
150 IF LEFT$(A$,1)=GUESS$ THEN PRINT@416,GUESS$;:Q=1:
   P=2:GOTO130
160 IF P>1 THEN PRINT@415+P,GUESS$;:P=P+1:Q=1:GOTO130
170 IF Q>0 THEN P=1:GOTO90
180 TRIES=TRIES+1
190 ON TRIES GOSUB 210,230,250,270,290,310,340,360,
   380,400,420,460
200 P=1:GOTO90
210 FORN=4 TO 26:SET(N,20,3):NEXT
220 RETURN
230 FORN=6 TO 20:SET(6,N,3):NEXT
240 RETURN
250 M=12:FOR N=6 TO 12:SET(M,N,3):M=M-1:NEXT
260 RETURN
270 FORN=6 TO 20:SET(N,6,3):NEXT
280 RETURN
290 FORN=6 TO 8:SET(20,N,3):NEXT
300 RETURN
310 FORN=19 TO 21:SET(N,8,5):SET(N,10,5):NEXT
320 FORN=8 TO 10:SET(19,N,5):SET(21,N,5):NEXT
330 RETURN
340 SET(20,11,5)
350 RETURN
```

```
360 FORN=17TO20:SET(N,12,6):NEXT
370 RETURN
380 FORN=20TO23:SET(N,12,6):NEXT
390 RETURN
400 FORN=12TO15:SET(20,N,6):NEXT
410 RETURN
420 M=17:FORN=19TO16STEP-1:SET(M,N,4):M=M+1:NEXT
430 RETURN
440 M=20:FORN=16TO19:SET(M,N,4):M=M+1:NEXT
450 RETURN
460 FORN=16TO26:RESET(N,20):NEXT
470 FORN=20TO26:SET(26,N,3):NEXT
480 FORN=17TO19:RESET(N,12):NEXT
490 FORN=21TO23:RESET(N,12):NEXT
500 FORN=13TO15:SET(18,N,6):SET(22,N,6):NEXT
510 SET(19,12,6):SET(21,12,6)
520 M=17:FORN=19TO16STEP-1:RESET(M,N):M=M+1:NEXT
530 M=20:FORN=16TO19:RESET(N,M):M=M+1:NEXT
540 FORN=16TO20:SET(19,N,4):SET(21,N,4):NEXT
550 PLAY"O1V31T2L4GGL8GGL4B-AAGGF+G"
560 PRINT@416,A$;
570 PRINT@64,"ANOTHER GO (Y/N)";:INPUT ANSWER$
575 IF ANSWER$="Y" THEN20
576 IF ANSWER$="N" THEN END
577 GOTO570
580 GOTO580
590 DATA"DOCTOR","BALL","ATHLETE","HELICOPTER"
600 DATA"COMPUTER","PSYCHIC","ELEPHANT","PERSON"
610 DATA"GIRAFFE","AXE","COMPETITION","LANGUAGE"
620 DATA"FIEND","PIZZA","DRAGON","PREVIOUS"
630 DATA"SCREEN","GALLOWS","HANGMAN","PRACTICE"
640 DATA"STAR","MISTAKE","NUMBER","READING"
650 DATA"CHARACTER","SELECT","KEYBOARD","PIANO"
1000 CLS0
1010 PRINT"YOU'RE FREE!"
1020 FORZ=1 TO 12
1030 ON Z GOSUB 310,340,360,380,400,420,440
1040 NEXT
1050 PLAY"O4V31T2L2CO-GL4GAL6GFL2EC"
1060 PLAY"L4GAL6GFL2BL4GEFFL6FEL4DL2C"
1070 GOTO570
1080 CLS
1090 PRINT@12,"hangman"
1100 PRINT"THE IDEA OF HANGMAN IS TO GUESS"
1110 PRINT"A WORD WHICH THE COMPUTER HAS"
1120 PRINT"CHOSEN."
1130 PRINT"AT THE BOTTOM OF THE SCREEN WILL";
1140 PRINT"BE DISPLAYED A NUMBER OF DOTS,"
1150 PRINT"EACH ONE REPRESENTING A LETTER."
1160 PRINT"YOU MUST GUESS THE WORD BY"
1170 PRINT"ENTERING A LETTER WHICH YOU"
1180 PRINT"THINK MAY BE IN THE WORD. IF THE";
1190 PRINT"LETTER WHICH YOU PICKED IS IN"
1200 PRINT"THE WORD IT WILL BE DISPLAYED IN";
```

```
1210 PRINT"IT'S CORRECT PLACE."
1229 PRINT:PRINT"PRESS ANY KEY TO CONTINUE";
1230 IF INKEY$="" THEN 1230
1240 CLS
1250 PRINT"IF YOU THINK YOU KNOW THE WORD"
1260 PRINT"YOU MAY TYPE IT IN AND SEE IF"
1270 PRINT"YOU ARE RIGHT."
1280 PRINT"AS AN INCENTIVE FOR YOU TO GET"
1290 PRINT"THE WORD RIGHT, EVERY TIME YOU"
1300 PRINT"PICK A WRONG LETTER ANOTHER PART";
1310 PRINT"OF A GALLOWS AND A MAN WAITING"
1320 PRINT"TO BE HUNG WILL BE DRAWN ON THE"
1330 PRINT"SCREEN. WHEN THE PICTURE IS"
1340 PRINT"COMPLETED THE MAN WILL BE HUNG"
1350 PRINT"AND THE GAME ENDS."
1360 PRINT" IF YOU GUESS THE WORD CORRECTLY"
1370 PRINT"THEN THE MAN GOES FREE."
1380 PRINT:PRINT"PRESS ANY KEY TO START"
1390 IF INKEY$=""THEN 1390
1400 GOTO20
```

Commentary

Line 15 sends the program to line 1080 to display the instructions when the program is first RUN. Line 20 clears the screen to a black background and line 25 tells the computer to start READing in DATA from the first piece of DATA in the program. Line 26 then resets all the variables.

Line 30 displays the message 'LETTERS USED:' in the top left-hand corner of the screen. Line 40 sets the variable p to 1 and lines 50 – 70 READ a random word from the DATA stored in lines 590 – 650. This is done by choosing a random number between 1 and 27 and READing in each word in the list up to and including the one which has been chosen by the random number. As each word is stored in the same variable only the last one is remembered.

Line 80 PRINTs as many dots as there are letters in the word which has been chosen (and is now stored in the variable A$). The LEN(A$) section of this line works out how many characters are in the word and then the STRING$ command PRINTs that many full stops (the full stop has the code 46).

Line 90 resets the variable Q and then erases all the characters on the 14th line before displaying the message 'WHAT IS YOUR GUESS' on the 14th line. Line 100 then waits for your guess and stores it in the variable GUESS$ (to make it easy to remember).

Line 110 checks to see if you have entered the word correctly, jumping to line 1000 if you have (notice that the GOTO command has been left off. It is possible to leave out GOTO commands in an IF statement after the THEN or ELSE commands). Line 120 displays the letter which you have tried on the second line, using the variable 'TRIES' to work out where it should go.

Line 130 uses the INSTR function to work out the position of the letter which you have chosen in the word which you are trying to guess. The letter's position is stored in the variable P if it is in the word. If your letter is not in the word then the number 0 is stored in the variable P.

Line 140 jumps to line 160 if the variable Q has the value 1. Line 150 checks to see if your guess is the first letter in the word, displaying it in its right place if it is and setting the variable Q to 1 and the variable P to 2 before going back to line 130.

Line 160 checks to see if the variable P contains a number greater than one, increasing the value of the variable P and setting the variable Q to 1 before jumping to line 130 if it does.

The reason why lines 150 and 160 send the program back to line 130 is to make sure that your letter does not occur more than once in the word. The variable P keeps a record of the letter's last position in the word and so the search for the letter is continued from that point when line 130 is carried out again. For example, if the word that you were trying to guess was DOCTOR and you typed in the letter O then the variable P would first of all contain the number 2, as the first O in the word DOCTOR is the second letter in the word. When line 130 was carried out again the computer would start searching for another O after the second letter, finding that the fifth character in the word was an O. This process is repeated until the computer has checked the whole word.

Line 180 increases the value of the variable TRIES by one, and then line 190 works out what the TRIESth line number in the list is and then jumps to that subroutine. Line 200 then sets the variable P back to 1 and goes to line 90.

Lines 210–300 draw the different parts of the gallows and lines 310–450 draw the man. Lines 460–540 RESET the part of the gallows underneath the man and replace it with an open trap door. The man's arms and legs are then RESET and re-drawn in different positions. Line 550 then plays the Death March to finish off the game.

Line 560 displays the word that you have been trying to guess on the 14th line before line 570 asks you if you want another game. If you do then line 575 sends the program back to line 20 to re-start the game (without the instructions this time). If not line 576 ENDs the program. Line 577 sends the program back to line 570 if you did not reply either Y or N.

Lines 590–650 contain the list of words which the computer can choose from, all stored in DATA statements.

Lines 1000 – 1070 are the routine which is carried out if you guess the word correctly. Line 1000 clears the screen to a black background before line 1010 displays the message "YOU'RE FREE!". Lines 1020– 1040 draw the man (without the gallows) and lines 1050 – 1060 play the tune Born Free. Line 1070 then sends the program to line 570 to see if you want another go.

Lines 1090– 1400 are the instructions.

3-D Plot

This program makes use of the Dragon's highest resolution mode to draw a three-dimensional picture of a jelly-like object. The program uses the mathematical functions such as SIN and SQR to work out where each point should be.

The computer takes a long time to draw the picture (about 10 minutes) as it has to work out the position of each individual point, even though the computer does work at twice its normal speed.

Normally the CPU (Central Processing Unit) in your Dragon works at a speed of 0.9MHz (you don't need to know what this means), but some Dragons are capable of working at a speed of 1.8Mhz, twice as fast as normal. Not all Dragons can work at this speed, however, so to see if yours does or not type in:

```
POKE &HFFD7,0
```

If you can see no noticeable difference except that the cursor is blinking faster than usual then your Dragon is capable of (and is) working at 1.8Mhz. If you find that the computer has crashed (in other words it stops working) then its maximum speed is 0.9Mhz, and you will have to turn your computer off and then on again. Don't do this too quickly — turning any computer on and off quickly can seriously damage it. If your Dragon can't work at 1.8MHz then you must take out any POKE &HFFD7,0and POKE &HFFD6,0 commands in programs.

Although the program is called 3-D PLOT it does not actually draw a true 3-D picture, just one that looks three-dimensional.

```
 10 POKE &HFFD7,0
 20 PMODE4
 30 SCREEN1,1
 40 PCLS
 50 A=128:B=A*A:C=96:D=96
 60 FOR X=0 TO A
 70 S=X*X
 80 P=SQR(B-S)
 90 I=-P
100 R=SQR(S+I*I)/A
110 Q=(R-1)*SIN(24*R)
120 Y=I/3+Q*D
130 IF I=-P THEN M=Y:GOTO160
140 IF Y>M THEN M=Y:GOTO170
150 IF Y>=N THEN GOTO 200
160 N=Y
170 Y=C+Y
180 PSET(A+X,Y,1)
190 PSET(A-X,Y,1)
200 I=I+4
210 IF I<P THEN 100
220 NEXT X
230 POKE &HFFD6,0
240 GOTO240
```

Commentary

Line 10 tells the Dragon to start working at twice its normal speed. Lines 20 – 40 sets up the mode 4 screen in inverse mode.

Lines 70 – 170 works out the position of each dot before lines 180 and 190 draws it up on the screen, one each side of the centre line.

Line 230 sets the working speed back to normal. This is important as you *cannot* save or load programs while the computer is working at 1.8MHz.

Meteors

This program requires a lot of skill (and luck) if you are to survive the onslaught of meteors racing up the screen at you. The program uses normal text mode (with a few graphics symbols thrown in here and there) and quite a bit of sound.

The idea of Meteors is to dodge the two types of meteors which come racing up the screen towards you, and also to destroy as many of them as possible with your missiles. You move left and right by using the left and right arrow keys. Firing is by the space bar. You may only have one missile on the screen at one time, so make sure that you aren't going to run into another meteor when you've fired.

Your ship is represented by an inverse V and is positioned on the second line down. Your missiles are represented by 'symbols'.

There are two types of meteor. The first, an O, is worth 10 points and the second, a *, is worth 5. If either of the two types of meteor hits you, a life is lost (you have 3 lives).

To make the game harder you move randomly from left to right, making it hard to dodge the meteors. There are three skill levels, each one with the meteors coming at different speeds and you drift more often as they get harder. The number of meteors also depends on the skill level.

Sometimes if two meteors come up the screen one below the other then you can destroy both with one missile. However you only get the points for the first meteor, not both.

```
10 REM METEORS
20 GOTO590
30 CLS
40 INPUT"SKILL LEVEL (1-3)";SKILL
50 IF SKILL<1 OR SKILL>3 THEN 40
60 CLS
70 SHIP=1104:LIVES=3
80 MISSILE=0:SCORE=0
90 POKESHIP,22
100 FOR N=0 TO RND(5-SKILL)
110 PRINT@480+RND(30),"O";
120 PRINT@480+RND(30),"*";
130 NEXT
140 POKE SHIP,143
150 FOR N=0 TO SKILL*10:NEXT N
160 IF MISSILE>0 THEN POKE MISSILE,143
170 IF MISSILE>0 THEN MISSILE=MISSILE+32
180 IF MISSILE>1505 THEN MISSILE=0
190 IF PEEK(MISSILE)=106 THEN GOSUB 430
200 IF PEEK(MISSILE)=79 THEN GOSUB 430
210 IF PEEK(MISSILE+32)=106 THEN GOSUB 430
220 IF PEEK(MISSILE+32)=79 THEN GOSUB 430
230 PRINT
240 X=RND(SKILL*3):IF X=2 THEN SHIP=SHIP+1
250 IF X=3 THEN SHIP=SHIP-1
260 IF PEEK(SHIP)=106 OR PEEK(SHIP)=79 THEN GOSUB 370
270 A$=INKEY$:IF A$="" THEN 340
280 POKE SHIP,143
290 IF A$=CHR$(9) THEN SHIP=SHIP+1
300 IF A$=CHR$(8) THEN SHIP=SHIP-1
310 IF A$=" " AND MISSILE=0 THENMISSILE=SHIP
320 IF SHIP>1119 THEN SHIP=1119
330 IF SHIP<1088 THEN SHIP=1088
340 IF MISSILE>0 THEN POKE MISSILE,103
350 PRINT@0,"LIVES:";LIVES;TAB(20);"SCORE:";SCORE
```

```
360 GOTO90
370 POKESHIP,191
380 SOUND 100,2
390 FOR N=0 TO 1000:NEXT
400 LIVES=LIVES-1
410 IF LIVES=0 THEN 510
420 RETURN
430 IF PEEK(MISSILE)=106 THEN SCORE=SCORE+5:SOUND250,1
440 IF PEEK(MISSILE+32)=106 THEN SCORE=SCORE+5:SOUND
   250,1
450 IF PEEK(MISSILE)=79 THEN SCORE=SCORE+10:SOUND250,2
460 IF PEEK(MISSILE+32)=79 THEN SCORE=SCORE+10:SOUND
   250,2
470 POKE MISSILE,175
480 POKE MISSILE,143
490 POKE MISSILE+32,143
500 MISSILE=0:RETURN
510 CLS:PLAY"O1V31T2L4GGL8GGL4B-AAGGF+G"
520 PRINT@192,"YOU'RE DEAD!"
530 PRINT"BUT YOU DID SCORE";SCORE;"POINTS!"
540 PRINT"ANOTHER GO (Y/N)?"
550 A$=INKEY$
560 IF A$="Y" THEN 30
570 IF A$="N" THEN END
580 GOTO 550
590 CLS
600 PRINT@12,"meteors"
610 PRINT"THE IDEA OF THIS GAME IS TO"
620 PRINT"DODGE THE METEORS WHICH ARE"
630 PRINT"COMING UP THE SCREEN TOWARDS YOU";
640 PRINT"AND TO DESTROY AS MANY OF THEM"
650 PRINT"AS POSSIBLE."
660 PRINT"YOU MOVE USING THE LEFT AND"
670 PRINT"RIGHT ARROW KEYS AND FIRE USING"
680 PRINT"THE SPACE BAR."
690 PRINT"THE SYMBOLS USED ARE:"
700 PRINT"O - METEOR - 10 POINTS"
710 PRINT"* - METEOR - 5 POINTS"
720 PRINT"v - YOUR SHIP"
730 PRINT"' - YOUR MISSILE"
740 PRINT"PRESS ANY KEY TO START"
750 IF INKEY$=""THEN750
760 GOTO30
```

Commentary

Line 20 sends the program to line 590 to display the instructions. Line 30 clears the screen and then line 40 asks you for the skill level that you want. Line 50 makes sure that you have made a legal choice before line 60 clears the screen again.

Lines 70 and 80 set up the variables that are going to be used, with SHIP being the position of your spaceshift and LIVES being the number of lives that you have left. MISSILE is the position of your missile on the screen. As you haven't fired yet this is set to 0.

Line 90 POKEs your ship onto the screen and then lines 100–130 PRINT a random number of O's and *'s on the bottom line of the screen. Line 140 then deletes your ship by POKEing a green square on top of it. Line 150 pauses for a short while (the amount of time depending on the skill level).

Line 160 checks to see if your missile is on the screen, deleting it if it is. Line 170 moves your missile down the screen one line (if it is on the screen), and line 180 makes sure that the missile doesn't go off the bottom of the screen. Lines 190–220 checks to see if your missile has hit a meteor, jumping to line 430 if it has.

Line 230 scrolls the screen by PRINTing nothing on the bottom line of the screen. Lines 240–250 controls the random movement of your ship, and line 260 checks to see if you have been hit by a meteor.

Line 270 scans the keyboard, jumping to line 340 if nothing is being pressed. Line 280 deletes your ship before lines 290 and 300 checks to see if you are pressing the arrow keys. Line 290 checks to see if you are pressing the right arrow key, moving you right one space if you are. Line 300 checks to see if you are pressing the left arrow key, moving you left one space if you are. Line 310 checks to see if you are pressing the space bar and that you haven't already got a missile on the screen, setting the variable missile to your position if both conditions are fulfilled.

Lines 320–330 make sure that your ship doesn't go off the left or right of the screen. Line 340 puts the missile on the screen (if it's supposed to be there) and line 350 displays the number of lives you have left and your score. Line 360 then sends the program back to line 90.

Line 370 POKEs a red square on top of your ship and line 380 makes a bleep. Line 390 pauses before line 400 subtracts one from the variable LIVES. Line 410 checks to see if you have run out of lives, jumping to line 510 if you have. Line 420 then RETURNs the program back to the command immediately after the GOSUB command which sent the program to this routine.

Lines 430–460 work out which type of meteor you have hit and increase the score accordingly. Line 470 POKEs your missile onto the screen and then lines 480–490 delete the missile and the next square underneath. Line 500 resets the variable MISSILE and then RETURNs to the main program.

Line 510 clears the screen and plays the Death March. Lines 520–530 tell you that you are dead (as if you hadn't guessed that from the tune) and then line 540 asks you if you want another go. Lines 550–580 then check your reply and take the appropriate action.

Lines 590–760 are the instructions.

Artist

Artist allows you to draw in any of the graphics modes and in any colour mode. As the program stands it is designed for use with the right joystick, but it can easily be converted for use with the keyboard.

When you RUN the program you will be asked which graphics mode you want to draw in and then the colour mode. The screen then clears to the mode that you require. Drawing colours are selected from the keyboard using the keys from 1 to 4 with the key corresponding to the colour's code. The colour which you are drawing in is constantly displayed in a box at the top of the screen.

You can change the screen colour simply by pressing one of the keys from 5 to 8. The colours produced by these keys are:

5 — GREEN   6 — YELLOW   7 — BLUE   8 — RED

The screen clears immediately on pressing any of these keys and anything on the screen is lost.

You may change the colour mode at any time simply by typing N. This does not clear the screen and so your drawings remain on the screen.

The program normally draws slowly. Pressing the fire button on the joystick, however, speeds up the drawing.

If you do not have a joystick on your Dragon then you can make these alterations to the program:

```
110 GOTO 130
135 B$ = INKEY$
140 IF B$ = CHR$(9) THEN X = X + 1
150 IF B$= CHR$(8) THEN X = X - 1
160 IF B$= CHR$(10) THEN Y = Y + 1
170 IF B$= " " THEN Y = Y - 1
```

These alterations allow you to draw using the arrow keys. The program works at full speed when using the keys.

```
10 CLS
20 INPUT"WHICH MODE (0-4)";MODE
30 IF MODE<0 OR MODE>4 THEN 20
40 INPUT"WHICH COLOUR MODE (0/1)";CM
50 IF CM<0 OR CM>1 THEN 40
60 PMODE MODE,1
70 SCREEN1,CM
80 PCLS
90 X=1:Y=6
100 COLOUR=1:COLOR COLOUR
110 IF PEEK(5280)=254 OR PEEK(65280)=126 THEN 130
120 FOR N=0 TO 500:NEXT
130 PSET(X,Y,COLOUR)
140 IF JOYSTK(0)>40 THEN X=X+1
150 IF JOYSTK(0)<22 THEN X=X-1
160 IF JOYSTK(1)>40 THEN Y=Y+1
170 IF JOYSTK(1)<22 THEN Y=Y-1
180 IF X>255 THEN X=255
190 IF X<0 THEN X=0
200 IF Y>191 THEN Y=191
210 IF Y<6 THEN Y=6
220 PSET(X,Y,COLOUR+1)
230 A$=INKEY$
240 IF A$="" THEN 370
250 IF A$="1" THEN COLOUR=1
260 IF A$="2" THEN COLOUR=2
270 IF A$="3" THEN COLOUR=3
280 IF A$="4" THEN COLOUR=4
290 IF A$="5" THEN PCLS1
300 IF A$="6" THEN PCLS2
310 IF A$="7" THEN PCLS3
320 IF A$="8" THEN PCLS4
330 IF A$="N" AND CM=1 THEN CM=0:SCREEN1,0:GOTO350
340 IF A$="N" THEN CM=1:SCREEN1,1
350 COLOR COLOUR
360 LINE(0,0)-(254,4),PSET,BF
370 GOTO110
```

Commentary

Line 10 clears the screen and line 20 asks you which graphic mode you want, storing your answer in the variable MODE. Line 30 makes sure that you have made a legal choice, jumping to line 20 if you haven't. Line 40 asks you which colour mode you require before line 50 ensures that you have made a legal choice.

Lines 60–80set up the screen and the line 90sets up the variables which will control the position of the cursor. Line 100 sets the variable COLOUR to 1 (green) and then sets up the colour (the command COLOR COLOUR tells the computer to set the colour to the one with the code number which is stored in the variable COLOUR).

Line 110 tests to see if the right joystick button is being pressed, jumping to line 130 if it is. Line 120 causes a short pause before line 130 plots the cursor's trail.

Lines 140–170 test the position of the right joystick and alter the values of the variables X and Y (X being the horizontal position and Y being the vertical position). Lines 180–210 make sure that the cursor doesn't go off the screen (or into the box showing the draw colour). Line 220 then plots the cursor.

Line 230 scans the keyboard and stores which key is being pressed in the variable A$. Line 240 jumps to line 370 if nothing is being pressed. Lines 250–280 control the colour changing, and lines 290–320 control the changing of the screen colour. Lines 330–340 control the inverting of the screen and line 350 changes the draw colour. Line 360 draws the box at the top of the screen and line 370 sends the program back to line 110to test the joystick button again.

Alarm Clock

Alarm Clock is, as you may have guessed, a clock program with a built-in alarm. The program also keeps a record of the data and whether it is am or pm. The date is not updated, mainly because you are not likely to leave your Dragon on overnight.

When the program is RUN you are first of all asked for the day (ie Monday, Tuesday etc). You are then required to enter the date in this format:

DAY eg 12 (12th day)
MONTH eg 03 (March)
YEAR eg 83 (1983)

The next piece of information to be entered is whether it is am or pm (just type in am or pm).

Once you have entered this information you are asked when you want the alarm to go off. This information should be entered in this way:

HOURS eg 01 (1 o'clock)
MINUTES eg 10 (10 minutes past)
SECONDS eg 05 (5 seconds)

You are then asked whether you want the alarm to go off in the morning or afternoon (am or pm).

As you may have noticed, each entry must be entered as a two-digit number. This means that if, for instance, it is two minutes past the hour you must enter the minutes as 02. The same applies to the date (eg 03 for March).

The next information is the actual time. This should be entered in the same way as the alarm time. When you come to enter the seconds you should add a few seconds to the actual time. You should then press the ENTER key at exactly the same time as your watch gets to the time which you have entered. This ensures that the clock is exactly right (at least by your watch!)

If you find that the clock does not stay accurate then you will have to make some alterations to the program. These are:

(1) Add spaces anywhere between lines 230 and 390 (fine tuning).
(2) Add spaces in the FOR . . . NEXT loop on line 360 (coarse tuning).
(3) Increase the 683 in the FOR . . . NEXT loop on line 360 (drastic tuning).

Now that you have your clock running accurately you will want to know what it can do. If you press the 1 key then the day will be displayed in the space previously occupied by the am/pm. Pressing the 2 key will result in the am/pm being displayed again.

If you now press the 3 key you will see the date displayed in place of the time. When you want the time back again you should simply press the 4 key.

Pressing the 5 key results in the alarm time being displayed in place of the time. To return to the normal display just press the 6 key.

If you do not like the tune which is played when the alarm goes off then you should simply alter lines 530 and 540. If you like the tune already in the program, but would prefer it at a different speed, then you should simply alter the T3 at the start of the tune in line 520.

```
10 CLS
20 INPUT"WHICH DAY IS IT";D$
30 D$=LEFT$(D$,2)
40 DAY$=CHR$(ASC(LEFT$(D$,1))+32)
50 DAY$=DAY$+CHR$(ASC(RIGHT$(D$,1))+32)
60 PRINT"PLEASE ENTER DATE":INPUT"DAY";D$
70 INPUT"MONTH";MONTH$:INPUT"YEAR";YEAR$
80 INPUT"AM OR PM";AP$:IF AP$="AM"THEN AM=1:ELSE PM=1
90 PRINT"PLEASE ENTER ALARM TIME"
100 INPUT"HOURS";AH$:INPUT"MINUTES";AM$
110 INPUT"SECONDS";AS$
120 INPUT"AM/PM";AT$
130 PRINT"PLEASE ENTER TIME":INPUT"HOURS";HOURS$
140 INPUT"MINUTES";MINUTES$:INPUT"SECONDS";SECONDS$
150 CLS0
160 FORX=23TO40:SET(X,13,7):NEXT
```

```
170 FORY=13TO16:SET(23,Y,7):SET(40,Y,7):NEXT
180 FORX=23TO29:SET(X,16,7):NEXT:FORX=34TO40:SET(X,16,
   7):NEXT
190 FORY=16TO18:SET(29,Y,7):SET(34,Y,7):NEXT
200 FORX=29TO34:SET(X,18,7):NEXT
210 IF AM=1 THEN PRINT@271,"am";:ELSE PRINT@271,"pm";
220 IF AT=0 THEN PRINT@236,HOURS$;":";MINUTES$;":";
   SECONDS$;
230 SECOND$=STR$(VAL(SECONDS$)+1)
240 IF LEN(SECONDS$)=2 THEN SECONDS$="0"+RIGHT$(
   SECONDS$,1)
250 IF LEN(SECONDS$)=3 THEN SECONDS$=RIGHT$(SECOND$,2)
260 IF SECONDS$="60"THENSECONDS$="00":MINUTE$=STR$(VAL
   (MINUTES$)+1):ELSE IF J=0 THEN 350
270 J=0
280 IF LEN(MINUTES$)=2 THEN MINUTES$="0"+RIGHT$(
   MINUTES$,1)
290 IF LEN(MINUTES$)=3 THEN MINUTES$=RIGHT$(MINUTE$,2)
300 IF MINUTES$="60"THEN MINUTES$="00":HOURS$=STR$(VAL
   (HOURS$)+1):ELSE GOTO 350
310 IF LEN(HOURS$)=2 THEN HOURS$="0"+RIGHT$(HOURS$,1)
320 IF LEN(HOURS$)=3 THEN HOURS$=RIGHT$(HOURS$,2)
330 IF HOURS$="12" THEN HOURS$="00":MINUTES$="00":
   SECONDS$="00":IF AM=1 THEN AM=0:PM=1:ELSE PM=0:AM=1
340 IF AM=1 THEN PRINT@271,"am";:ELSE PRINT@271,"pm";
350 FOR N=0 TO 683 :NEXT
360 IF HOURS$=AH$ AND MINUTES$=AM$ AND SECONDS$=AS$
   AND AT$=AP$ THEN 510
370 SOUND 255,1
380 A$= INKEY$
390 IF A$="1" THEN PRINT@271,DAY$;
400 IF A$="2" AND AM=1 THEN PRINT@271,"am";
410 IF A$="2" AND PM=1 THEN PRINT@271,"pm";
420 IF A$="3" THEN PRINT@236,D$;":";MONTH$;":";YEAR$;:
   D=1
430 IF A$="4" THEN D=0:GOTO220
440 IF A$="5" THEN PRINT@236,AH$;":";AM$;":";AS$;:IF
   INKEY$<>"6" THEN AT=1
450 IF AT=1 AND AT$="AM" THEN PRINT@271,"am";
460 IF AT=1 AND AT$="PM" THEN PRINT@271,"pm";
470 IF A$="6" THEN AT=0:IF PM=1 THEN PRINT@271,"pm";
480 IF A$="6" AND AM=1 THEN PRINT@271,"am";
490 IF D=1 THEN 230
500 GOTO 220
510 TIMER=0
520 PLAY "T3O3L4GG;L2GDL4BB;L2BGL4GB;O4L2DDL3CO3B;"
530 PLAY "L1AL4AB;O4L2CCO3L4BA;L2BGL4GB;L2ADL4F#A;L1
  G;"
540 SECONDS$=STR$(VAL(SECONDS$)+INT(TIMER/60)+4)
550 IF VAL(SECONDS$)>60 THEN MINUTES$=STR$(VAL(
  MINUTES$)+1):J=1
560 IF VAL(SECONDS$)>60 THEN SECONDS$=STR$(VAL(
  SECONDS$)-60)
570 GOTO   240
```

Commentary

Line 10 clears the screen and line 20 asks you what the day is, storing the answer in the variable D$. Line 30 then takes the first two letters of the string D$ and then stores them in the variable D$ replacing its original contents. Lines 40 and 50 then convert the contents of the variable D$ to lower case before storing them in the variable DAY$.

Lines 60 and 70 then ask you for the date before line 80 asks you whether it is morning or afternoon, setting either the variable am or pm to 1 accordingly.

Lines 90 – 120 ask you for the time that you want the alarm to go off. You are then asked for the actual time by lines 130 – 140.

Line 150 clears the screen to a black background. The border around the clock is then drawn by lines 160 – 210. Line 220 then displays whether it is am or pm at the bottom of the clock before line 230 displays the time.

Line 240 increases the time by one second and lines 250–260 get rid of the space added to the start of the variable SECONDS$ by the STR$ function in line 240.

Lines 270–300 are similar to lines 240–260 except that they update the minutes. Lines 310–340 update the hours, with line 340 also updating the variables am and pm if it is afternoon or morning. Line 350 then displays whether it is am or pm.

Line 360 causes a delay to keep the clock accurate. Line 370 checks to see if it is time for the alarm to go off yet, jumping to line 520 if it is. Line 380 then makes the tick.

Line 390 scans the keyboard, storing the result in the variable A$. If you are pressing the 1 key then line 400 displays the day in place of the am/pm. If you are pressing the 2 key then lines 410–420 display whether it is am or pm. Line 430 displays the month in place of the time if you are pressing the 3 key. If you press the 4 key then line 440 allows the time to be displayed. Lines 450 – 470 display the alarm time if you press the 5 key, and lines 480–490 allow you to see the time/date again.

Lines 520 – 580 make up the alarm tune routine. Line 520 resets the built-in timer so that a record can be kept of how long it takes to play the tune. Lines 530–540actually PLAY the tune and then line 550 updates the time. Lines 560 and 570 update the minutes and seconds before line 580 sends the program back to line 250.

Valley of Death

Valley of Death is a program for the adventurous ones amongst you, in more ways than one! The program takes up most of the Dragon's 32K of memory and will probably take you quite a long time to type in, as you will see if you look at the program. However, the length of the program should not put you off, you can easily enter the program bit by bit and record each section as you go if you do not feel like entering it all in one go. In fact it is a good idea to save your program every now and then as you enter it. This will make sure that you don't lose several hours of work when someone jolts the power-pack plug. When you have finally entered the program and de-bugged the program your efforts will be rewarded with a very good (even if I do say so myself) adventure program.

Before we get on to how to play this game, a word or two about entering the program. If you look at lines 230–260 you will see a symbol which is not on your keyboard, the \ symbol. To get this symbol we have to confuse the computer into thinking that it has an extra key by pressing three keys at once. Sounds complicated? Well not really, all you have to do is hold down the SHIFT key, press the CLEAR key (while holding down the SHIFT key) and then press the @ key (while still holding down the other two keys). Then release the @ key before you let go of the others. You will then see the \ symbol appear on the screen. At first you will probably find this complicated, but with a bit of practice you will be able to do this quite quickly.

To help you to enter the program we have used the # symbol to represent a space, if more than one is needed in any part of the program. This means that you can count the number of spaces that you need more easily.

Now to the game. Playing Valley of Death is quite complex, but is also very rewarding. You will be amazed at the pleasure you get from hitting a dragon over the head with your sword, and realising that you have saved the life of your King.

The idea of Valley of Death is to find a key which is laying around somewhere in the Dark Dungeons of Darganyon and to take this back to the Palace. This key will then enable you to open a magical chest which contains a potion which will save the life of your dying King. However, to find the key you must first find a magical wand which is in one of the many caves scattered about the valley. This wand also allows you to cast spells.

Nothing to it, you think. However, there are some slight hazards. If you stray off the safe path running through the valley you get attacked by vicious monsters. When you first start off in the game you have only a sword to defend yourself with, but once you find the Wand you may also use spells (but only a few). There is also a magical sword in the Dark Dungeons of Darganyon which does much more harm to the monsters.

Now that you know roughly what you have to do in the game we can go through the whole thing step by step. The best place to start is probably at the beginning of the game, so that's where we'll begin.

When you RUN Valley of Death you will be confronted with the question 'LOADCHARACTEROR RESTART?'. At first you should reply R and press ENTER. You will then be presented with a list of the characters that you can be. These are:

WARRIOR
CLERIC
WIZARD
BARBARIAN

Each type of character has its good and bad points, but we'll leave you to work these out for yourself (nasty aren't we?). You will be asked which type you want to be and then for your name.

The screen will then clear and a map will be drawn on the screen. This map will be composed of a zig-zagging line going across the screen with an inverse P and K at either end. The zig-zagging line represents the safe path which keeps you safe from the monsters, as do the Palace (the P) and the Keep (the K). Scattered about the screen will be some Os which represent the caves, and also an inverse minus sign (a swamp) and two up-arrows (a forest).There is also an inverse D which represents the Dark Dungeon of Darganyon. After a short pause an inverse dollar symbol will appear on the Palace — this is you.

Below the map will be all the information that you need to know about your character — his STrength, his IQ, his ENergy, his TReasure and his EXperience (the letters in capitals are what appear on the screen). Your strength, IQ and energy can go up to a maximum of 400 points, but your treasure and experience can go as high as you like (or rather as high as you can survive which is not necessarily the same). If your energy goes down to 0 you die, but none of the other scores are really a matter of life and death.

Once the inverse dollar symbol which represents you appears on the Palace the message 'SAFE IN PALACE' will appear just under the map. You may then proceed to move around in search of adventure. To move you should use the keys:

```
R T Y
F G H
C V B
```

R moves you upwards and left, H moves you right, B moves you downwards and right. G allows you to rest, which increases your energy to a certain extent.

Now that you can move around you need to know how to fight the monsters which you are sure to meet when you wander off the path. When you meet a monster an ominous noise will come from your television's speaker and the message YOU HAVE MET A followed by the type of monster you have met will appear just below the map. You will then see the message STRIKE NOW appear at the bottom appear at the bottom of the screen. You should then press the H key as quickly as you can (H stands for Hit). If you do not press the key in time then the message TOO SLOW will appear in place of STRIKE NOW and the monster will hit you. You should remember that you do not always hit the monster, and the monster also misses you sometimes.

While all this fighting is going on the monster's energy is displayed on the bottom line of the screen. If this goes down to 0 (the energy is knocked off by you hitting the monster) then the monster dies. Your experience is then increased according to how powerful the monster was.

You now know how to move, fight (and hopefully kill) the monsters, so you are just about ready to start looking for The Key. Your first stop should either be one of the caves, the swamp or the forest (entering the Dark Dungeons of Darganyon before you have The Wand is committing suicide). We'll take the caves first.

When you enter any of the caves the map will be replaced with a picture of the cave, a pretty dark place with several objects scattered about the floor (represented by coloured blocks). You may move freely around the caves without being bothered by monsters. If you want to pick up one of the objects all you have to do is move on top of one of them. You will then be told what the object is. Each object may either be:

The Wand
The Medallion of Life
The Shield of Protection
A gem
A worthless object
A potion
A monster

We'll take each of these objects one by one and explain each one.

The Wand allows you to cast magical spells. At first you have three spells, each of which may be used up to six times. The success of the spell depends on your IQ, the higher your IQ is the more likely it is that your spell will work. Once your experience reaches 2000 you are allowed to use three more spells. The spells are:

(l) SLEEP
(2) BLINDING LIGHT
(3) MAGIC SHIELD
(4) WEB
(5) DARKNESS
(6) JELLYFIER

Each spell allows you to escape from a monster in one way or another.

The spells may only be used when fighting a monster and are used simply by typing S in reply to the prompt 'STRIKE NOW'. You will then be asked which spell you require (from 1 – 3 or 1 – 6 depending on your experience). If at any time you want to know how many spells you have left you should simply type S, but only when you are not fighting.

The Medallion of Life will probably save you in many sticky situations. This object will keep you alive for 16 moves if you are killed. If you manage to reach either the Palace or the Keep before these 16 moves are up then you are reincarnated, otherwise you die.

The Shield of Protection is another magical item and cuts down the damage which the monsters can do to you. The Shield comes into use when you are attacked by monsters.

The gems are, as you may have guessed, precious jewels. These increase your treasure, but not a lot else.

The worthless object is, surprise surprise, an object which is totally worthless.

If you find a potion it is automatically put in your backpack. You may take a potion at any time (apart from when you are fighting) simply by typing P and then the number of the potion which you want to take. The effects of the potions are numerous and you don't know what a potion might do until you try it.

You can guess what happens if the object is a monster!

Both the swamp and forest contains castles surrounded by red moats which you must swim across to reach the castle. Once you enter the castle you will see a map of it drawn on the screen with several stars scattered about. These stars are objects, either worthless stones, gems or the Amulet of the Gods (which is what you came in for). Monsters rove around in the castle as they please, so you can run into one at any time.

The door to the castle closes for a set amount of time once you enter, so you have to stay in for a short while at least. If you manage to find the Amulet of the Gods your strength, IQ and energy will be increased and you will be allowed to use each spell 100 times (if you have The Wand, that is).

Once you have found all the various objects in the caves, forest and swamp you can venture into the Dark Dungeons of Darganyon. When you enter the Dungeons the screen will clear and you will see a map similar to that of the castle except that there are a set of stairs in the top right hand corner (represented by a cyan block). Again there are stars scattered about and again there are monsters all over the place (there are many more monsters in the Dungeons). The stars in the Dungeon represent either gems, worthless objects, The Key (the finding of which is what the whole game is about) and a magic sword which does a lot more damage to the monsters than your ordinary sword.

When (or, more likely, if) you reach the stairs you will be asked whether you want to go up or down, to which you should reply U or D. Obviously if you are at the top you can't go up, and you can't go down if you're at the bottom.

Once you have the key you should try and get back to the Palace as quickly as possible so that you can open the chest. You will then be rewarded with a very nice picture of the chest, and another of the chest opened.

A few little extras to help you along are available by pressing these keys:
I — list of everything you have
E — your rating (this is based on your experience.

If your journey into the depths of a monster-infested swamp or your battle with a ferocious Wight is rudely interrupted by someone telling you that your dinner is ready then it is possible to record your character on tape. This is done by pressing the @ key. You will then be asked whether the tape is ready to which you should reply Y when it is. Your character will then be recorded.

When you wish to continue your game you should first load and RUN Valley of Death and reply L to the question 'LOAD CHARACTER OR RESTART'. You will then be asked for the character's name and told to press the PLAY button on your tape recorder. Your character will be loaded and the game will continue.

You now know all the vital things about The Valley of Death, but there are quite a few details which are left for you to find out (after all, what's the point of an adventure when you know exactly what to do?). This program should have you glued to your chair and you will probably find it hard to pull yourself away from the game until your rating has progressed to at least Apprentice Fool!

As the Valley of Death is such a long program there is no explanation following it. An explanation for this program would probably take up a whole book on its own and you will probably still not be any wiser.

Enough of these explanations. The program is sitting here waiting for you to enter and play it, so good luck, and good adventuring!

```
10 DIME(200):FORN=1TO200:E(N)=N*500:NEXT:G=1:Z=16:PL=1
20 DIMA$(4),POTION(20),SPELL(6):FORN=1TO6:SPELL(N)=6
30 NEXT:CLS:MAN=1056:I=1
40 INPUT"LOAD CHARACTER OR RESTART";A$
50 IFA$="L"THEN3690
60 IFA$<>"R"THEN40
70 ST=RND(9)+RND(9)+RND(9):IQ=RND(9)+RND(9)+RND(9)
80 EN=RND(9)+RND(9)+RND(9):ST=ST*5:IQ=IQ*5:EN=EN*6
90 PRINT"1) WARRIOR":PRINT"2) CLERIC":PRINT"3)
  BARBARIAN"
100 PRINT"4) WIZARD":INPUT"WHICH ONE (1-4)";A
```

```
110 IFA<0ORA>4ORCL$=""THENCL$="FOOL":IQ=IQ-RND(10)
120 IFA=1THENCL$="WARRIOR":ST=ST+RND(ST):IQ=IQ+RND(10)
130 IFA=2THENCL$="CLERIC":IQ=IQ+RND(10)*2
140 IFA=3THENCL$="BARBARIAN":IQ=IQ-RND(5):ST=ST+RND
    (ST)
150 IFA=3THENEN=EN+RND(ST)
160 IFA=4THENCL$="WIZARD":IQ=IQ+RND(15)*2:EN=EN+RND(5)
170 INPUT"NAME";NAME$
180 IFNAME$=""THENNAME$="MR.'X'"
190 CLS
200 PRINTSTRING$(32,175);:FORN=0TO7:PRINTCHR$(175);
210 PRINTSTRING$(30,32);CHR$(175);:NEXT
220 PRINTSTRING$(32,175)
230 A$(1)=CHR$(175)+"####################/\########"+
    CHR$(175)
240 A$(2)=CHR$(175)+"#/\/\########/\/\##/##\/\#####"+
    CHR$(175)
250 A$(3)=CHR$(175)+"p####\####/\/####\/######\/\/k"+
    CHR$(175)
260 A$(4)=CHR$(175)+"######\/\/"+STRING$(20,32)+CHR$
    (175)
270 J=RND(5)*32:PRINT@J,A$(1);:PRINTA$(2);:PRINTA$(3);
280 PRINTA$(4);
290 N=RND(288)+1024:IFPEEK(N)=96ANDPEEK(N+1)=96THEN
    POKEN,94:POKEN+1,94:ELSEGOTO290
300 N=RND(288)+1024:IFPEEK(N)=96THENPOKEN,4:ELSE300
310 PRINT@480,STRING$(30,32);
320 FORN=0TORND(6)
330 R=RND(288)+1024:IFPEEK(R)=96THENPOKER,79:ELSE330
340 NEXT
350 N=RND(286)+1024:IFPEEK(N)=96THENPOKEN,45:ELSE350
360 PRINT@352,NAME$;" THE ";CLASS$
370 PRINT@384,"ST:";ST;TAB(10);"IQ:";IQ;TAB(20);"EN:";
380 PRINTEN;:PRINT@416,"TR:";TR;TAB(20);"EX:";EX;
390 IFU=1THENRETURN
400 U=1
410 MAN=MAN+J+33
420 GOSUB1490
430 GOTO720
440 POKEMAN,36
450 IFEX>E(G)THENG=G+1:EN=EN+RND(5)*5:IQ=IQ+RND(5)*5:
    ST=ST+RND(5)*5
460 IFIQ>400THENIQ=400
470 IFEN>400THENEN=400
480 IFST>400THENST=400
490 GOSUB360
500 IFRND(4)=2ANDDEAD=0ANDCASTLE=0ANDPATH=0ANDPL=0THEN
    GOSUB2770
510 IFDEAD<>0THENDEAD=DEAD+1:IFDEAD=18ANDCASTLE<>1AND
    PL<>1THEN3770
520 IFDEAD>0AND(CASTLE=1ORPL=1)THENPRINT@320,"YOU'RE
    ALIVE AGAIN!";:FORN=0TO2000:NEXT:PRINT@320,"#######
    ############";:EN=(RND(50))*3:DEAD=0:GOSUB360
530 GOSUB2300
```

```
540 E=E+1:IFE=10THENEN=EN-1:E=0
550 IFPEEK(MAN+D)=4THENQ=MAN:W=Z:GOTO1510
560 IFPEEK(MAN+D)=111ORPEEK(MAN+D)=92THENMAN=MAN+D:
    POKEMAN-D,Z:PATH=1:Z=PEEK(MAN):GOTO660
570 IFPEEK(MAN+D)=11THENMAN=MAN+D:POKEMAN-D,Z:CASTLE=
    1:Z=PEEK(MAN):GOTO660
580 IF PEEK(MAN+D)=16 THEN MAN=MAN+D:POKEMAN-D,Z:PL=1:
    Z=PEEK(MAN):GOTO660
590 IFPEEK(MAN+D)=45THENMAN=MAN+D:POKEMAN-D,Z:Z=
    PEEK(MAN):Q=MAN:W=Z:Z=143:GOSUB800:GOTO660
600 IFF=0AND PEEK(MAN+D)=94THENMAN=MAN+D:POKEMAN-D,Z:
    Z=PEEK(MAN):Q=MAN:W=Z:Z=143:GOSUB880:GOTO660
610 IFPEEK(MAN+D)=79THENMAN=MAN+D:Q=MAN:W=79:GOSUB2520
620 IFPEEK(MAN+D)=175AND(SWAMP=1 OR F=1)THENGOSUB1500:
    MAN=Q:Z=W:SWAMP=0:F=0:GOTO660
630 IFPEEK(MAN)=175THENMAN=MAN+32
640 IFZ=175THENZ=96
650 IFPEEK(MAN+D)<>175THENMAN=MAN+D:POKEMAN-D,Z:Z=PEEK
    (MAN)
660 IFZ<>11THENCASTLE=0
670 IFZ<>16THENPL=0
680 IFZ<>92ANDZ<>111THENPATH=0
690 IFZ=207THENQ=MAN:W=Z:GOTO970
700 PRINT@320," "
710 IFPATH=1THENPRINT@320,"SAFE ON PATH
720 IFCASTLE=1THENPRINT@320,"SAFE IN KEEP"
730 IFSWAMP=1THENPRINT@320,"IN THE SWAMP"
740 IFF=1THENPRINT@320,"IN THE FOREST"
750 IFPL=1THENPRINT@320,"SAFE IN THE PALACE"
760 IFZ=159THEN970
770 D=0
780 IFPL=1ANDKEY=1THEN3370
790 GOTO440
800 FORN=32TO256STEP32:PRINT@N,CHR$(175);STRING$
    (30,32);CHR$(175);:NEXT
810 FORN=0TO70:X=RND(255)+1056
820 IFPEEK(X)=96THENPOKEX,109
830 NEXT
840 PRINT@320,"IN THE SWAMP";
850 GOSUB1470
860 PATH=0:SWAMP=1:MAN=1296
870 RETURN
880 FORN=32TO256STEP32:PRINT@N,CHR$(175);STRING$
    (30,32);CHR$(175);:NEXT
890 FORN=0TO70:X=RND(255)+1056
900 IFPEEK(X)=96THENPOKEX,94
910 NEXT
920 F=1
930 PRINT@320,"IN THE FOREST"
940 GOSUB1470
950 PATH=0:F=1:MAN=1296
960 RETURN
970 FORN=0TO288STEP32:PRINT@N,STRING$(32,32);:NEXT
980 PRINT@320,"IN THE CASTLE";
```

```
990 TIMER=0
1000 PRINT@8,STRING$(16,191):PRINT@40,CHR$(191);
1010 PRINTSTRING$(14,32);CHR$(191):PRINT@72,CHR$(191);
1020 PRINT"##";:FORN=0TO4:PRINTCHR$(191);:NEXT:PRINT
     "##";
1030 PRINTCHR$(191);CHR$(191);" ";CHR$(191);" ";CHR$
     (191)
1040 PRINT@104,CHR$(191);" ";CHR$(191);CHR$(191);" ";
1050 PRINTCHR$(191);" ";CHR$(191);" ";CHR$(191);
     "#####";CHR$(191)
1060 PRINT@136,CHR$(191);"##";CHR$(191);"######";
     CHR$(191);
1070 PRINT@136,CHR$(191);"##";CHR$(191);"######";
     CHR$(191);
1080 PRINT"####";CHR$(191):PRINT@168,CHR$(191);"##";
     CHR$(191);
1090 PRINTCHR$(191);" ";CHR$(191);CHR$(191);CHR$(191);
1100 PRINT"  ";CHR$(191);" ";CHR$(191);CHR$(191);
1110 PRINT CHR$(191):PRINT@200,CHR$(191);"######";
     CHR$(191);
1120 PRINTCHR$(191);"######";CHR$(191)
1130 PRINT@232,CHR$(191);" ";STRING$(4,191);
1140 PRINT"####";:FORN=0TO5:PRINTCHR$(191);:NEXT
1150 PRINT@264,CHR$(191);" ";CHR$(191);"####";CHR$
     (191);
1160 PRINT CHR$(191);"####";CHR$(191);CHR$(191);
     CHR$(191)
1170 PRINT@296,CHR$(191);CHR$(207);STRING$(14,191)
1180 FORN=0TORND(4)+2
1190 X=RND(9)*32:X=X+RND(14)+1032
1200 IFPEEK(X)=96THENPOKEX,106:ELSEGOTO1190
1210 NEXT
1220 MAN=1321:Z=207
1230 POKEMAN,36
1240 IFIQ>400THENIQ=400
1250 IFEN>400THENEN=400
1260 IFST>400THENST=400
1270 IFRND(4)=2ANDDEAD=0THENGOSUB2770
1280 IFDEAD<>0THENDEAD=DEAD+1:IFDEAD=18ANDCASTLE<>1AND
     PL<>1THEN3770
1290 GOSUB360
1300 GOSUB2300
1310 E=E+1:IFE=10THENEN=EN-1:E=0
1320 IFPEEK(MAN+D)=96THENMAN=MAN+D:POKEMAN-D,Z:Z=PEEK
     (MAN):GOTO1410
1330 IFPEEK(MAN+D)=106THENMAN=MAN+D:POKEMAN-D,Z:Z=96:
     ELSE1400
1340 R=RND(5)
1350 IFR=4ANDAMULET=0THENPRINT@480,"YOU'VE FOUND THE
     AMULET!";
1360 IFR=4ANDAMULET=0THENFORN=0TO2000:NEXT:PRINT@480,
     STRING$(24,32);:EN=EN+RND(10)*10
1370 IFR=4ANDAMULET=0THENIQ=IQ+RND(10)*10:ST=ST+RND
     (10)*10:FORN=1TO6:SPELL(N)=100:NEXT:AMULET=1
```

```
1380 IFR=5THENPRINT@480,"YOU'VE FOUND A PRECIOUS
     STONE!";:TR=TR+RND(5)*100
1390 IFR=5THENFORN=0TO2000:NEXT:PRINT@480,STRING$
     (31,32);
1400 IFR<4THENPRINT@480,"YOU'VE FOUND A WORTHLESS
     STONE!";:FORN=0TO2000:NEXT:PRINT@480,STRING$
     (31,32);
1410 D=0
1420 IFTIMER>1000THENPOKE1321,96
1430 IFMAN=1321THENZ=143:FORN=32TO256STEP32:PRINT@N,
     STRING$(32,32);:NEXT:PRINT@0,STRING$(32,175)
1440 IFMAN=1321THENPRINT@288,STRING$(32,175):IFF=1THEN
     GOSUB880:MAN=1164:GOTO430:ELSEGOSUB800:MAN=1164:
     GOTO430
1450 GOTO1230
1460 END
1470 POKE1097,191:POKE1098,191:POKE1099,191:POKE
     1128,191:POKE1131,191:POKE1132,191:FORN=1160TO
     1163:POKEN,191:NEXT
1480 POKE1193,191:POKE1130,159:POKE1129,96:RETURN
1490 DIMA(320):FORN=0TO320:A(N)=PEEK(N+1024):NEXT:
     RETURN
1500 FORN=0TO320:POKE1024+N,A(N):NEXT:RETURN
1510 FORN=0TO288STEP32:PRINT@N,STRING$(32,32);:NEXT
1520 PRINT@8,STRING$(16,191):PRINT@40,CHR$(191);
     "####";CHR$(191);"  ";CHR$(191);"######";CHR$(223);
     CHR$(191)
1530 PRINT@72,CHR$(191);"  ";CHR$(191);"####";CHR$
     (191);"  ";CHR$(191);CHR$(191);CHR$(191);"###";CHR$
     (191)
1540 PRINT@104,CHR$(191);"  ";:FORN=0TO3:PRINTCHR$
     (191);:NEXT:PRINT"  ";CHR$(191);"###";CHR$(191);
     "  ";CHR$(191);
1550 PRINTCHR$(191);CHR$(191)
1560 PRINT@136,CHR$(191);"  ";CHR$(191);"  ";CHR$
     (191);"##";CHR$(191);CHR$(191);"##";CHR$(191);
     "###";CHR$(191)
1570 PRINT@168,CHR$(191);"###";CHR$(191);"###";CHR$
     (191);"  ";CHR$(191);CHR$(191);"  ";CHR$(191);"  ";
     CHR$(191)
1580 PRINT@200,CHR$(191);"  ";CHR$(191);"  ";CHR$
     (191);CHR$(191);CHR$(191);"  ";CHR$(191);CHR$(191);
1590 PRINTCHR$(191);"##";CHR$(191);"  ";CHR$(191)
1600 PRINT@232,CHR$(191)"  ";CHR$(191);"  ";CHR$(191);
     "  ";CHR$(191);"  ";CHR$(191);"###";CHR$(191);CHR$
     (191);"  ";
1610 PRINTCHR$(191):PRINT@264,CHR$(191);"  ";CHR$
     (191);"###";CHR$(191);"###";STRING$(3,191);"##";
     CHR$(191)
1620 PRINT@296,CHR$(191);CHR$(207);STRING$(14,191)
1630 MAN=1321
1640 FORN=0TORND(4)+2
1650 IFO=1THENPOKEMAN,36
1660 X=RND(9)*32+RND(14)+1032:IFPEEK(X)=96THENPOKEX,
     106:ELSEGOTO1660
1670 NEXT:IFO=0THENMAN=1321:Z=207
1680 IFLEVEL=0THENTIMER=0
1690 TW=1
1700 PRINT@320,"IN THE dungeon########";
1710 IFDEAD<>0THENDEAD=DEAD+1:IFDEAD=18ANDCASTLE<>1AND
     PL<>1THEN3770
1720 POKEMAN,36
1730 IFIQ>400THENIQ=400
1740 IFEN>400THENEN=400
1750 IFST>400THENST=400
1760 GOSUB2300
1770 GOSUB360
1780 E=E+1:IFE=10THENEN=EN-1:E=0
1790 IFRND(4)=2ANDDEAD=0THENGOSUB2770
1800 IFLEVEL=0ANDTIMER>1000THENPOKE1321,96
1810 IFPEEK(MAN+D)=223THENMAN=MAN+D:Z=223:GOSUB2460:
     GOTO1930
1820 IFPEEK(MAN+D)=96THENMAN=MAN+D:POKEMAN-D,Z:Z=96:
     GOTO1840
1830 IFPEEK(MAN+D)=106THENMAN=MAN+D:POKEMAN-D,Z:Z=96:
     GEM=1
1840 IFMAN=1321THENPRINT@320,STRING$(17,32);:GOSUB
     1500:MAN=Q:Z=W:TW=0:GOTO440
1850 IFGEM=1THENR=RND(6)
1860 IFR=4ANDRND(5)=2ANDWAND=1ANDKEY=0THENPRINT@480,
     "YOU'VE FOUND the";CHR$(128);"key!";:FORN=0TO2000:
     NEXT:PRINT@480,STRING$(30,32);:KEY=1:GEM=0
1870 IFGEM=1ANDR=2ANDSD=0THENPRINT@480,"YOU'VE FOUND
     THE MAGIC SWORD!";:FORN=0TO2000:NEXT:PRINT@480,
     STRING$(31,32);:SD=1:GEM=0
1880 IFGEM=1ANDR=2THENPRINT@480,"YOU'VE FOUND A
     PRECIOUS STONE!";:FORN=0TO2000:NEXT:PRINT@480,
     STRING$(31,32);:TR=TR+RND(5)*100:GEM=0
1890 IFGEM=1THENPRINT@480,"YOU'VE FOUND A WORTHLESS
     STONE!";:FORN=0TO2000:NEXT:PRINT@480,STRING$
     (31,32);:GEM=0
1900 D=0
1910 GOTO1720
1920 END
1930 IFLEVEL=0THENO=1:GOTO1510:ELSEIFLEVEL>0ANDLEVEL<6
     THENON LEVEL GOTO1970,2050,2120,2180,2240
1940 IFLEVEL<0THENLEVEL=0:GOTO1930
1950 IFLEVEL>5THENLEVEL=5:GOTO1930
1960 O=1:GOTO1640
1970 PRINT@41,"#####";CHR$(191);"########";CHR$
     (223);CHR$(191):PRINT@73,"###";STRING$(8,191);" ";
     CHR$(191);" ";CHR$(191)
1980 PRINT@105," ";CHR$(191);"##";CHR$(191);"#######";
     CHR$(191);" ";CHR$(191):PRINT@137," ";CHR$(191);
     " ";CHR$(191);CHR$(191);"##";CHR$(191);"##";
1990 PRINTCHR$(191);" ";CHR$(191);" ";CHR$(191)
2000 PRINT@169," ";CHR$(191);" ";CHR$(191);"##";CHR$
     (191);CHR$(191);" ";CHR$(191);CHR$(191);"###";CHR$
     (191):PRINT@201," ";CHR$(191);"#####";CHR$(191);
2010 PRINT" ";CHR$(191);"##";CHR$(191);" ";CHR$(191)
2020 PRINT@233," ";STRING$(3,191);"##";STRING$(4,191);
     " ";CHR$(191);CHR$(191);" ";CHR$(191):PRINT@
     265,CHR$(223);"#######";CHR$(191);"###";CHR$(191);
2030 PRINT" ";CHR$(191):PRINT@296,STRING$(16,191);
2040 GOTO1960
2050 PRINT@41,STRING$(13,32);CHR$(223);CHR$(191):
     PRINT@73,"###";STRING$(8,191);" ";CHR$(191);" ";
     CHR$(191)
2060 PRINT@105," ";CHR$(191);" ";CHR$(191);CHR$(191);
     "#########";CHR$(191):PRINT@137," ";CHR$(191);
     "##";CHR$(191);"#####";CHR$(191);" ";CHR$(191);
2070 PRINT" ";CHR$(191)
2080 PRINT@169," ";CHR$(191);"##";CHR$(191);" ";
     STRING$(7,191);" ";CHR$(191):PRINT@201," ";CHR$
     (191);"###########";CHR$(191);" ";CHR$(191)
2090 PRINT@233," ";CHR$(191);CHR$(191);CHR$(191);
     " ";STRING$(5,191);" ";CHR$(191);CHR$(191);" ";
     CHR$(191):PRINT@265,CHR$(223);"#######";CHR$(191);
2100 PRINT"#####";CHR$(191)
2110 GOTO1960
2120 PRINT@41,"###";CHR$(191);"###";STRING$(4,191);
     "##";CHR$(223);CHR$(191):PRINT@73,"#####";CHR$
     (191);"######";CHR$(191);" ";CHR$(191)
2130 PRINT@105," ";STRING$(9,191);" ";CHR$(191);CHR$
     (191);" ";CHR$(191):PRINT@137,STRING$(14,32);CHR$
     (191):PRINT@169,STRING$(5,191);" ";CHR$(191);" ";
2140 PRINTCHR$(191);CHR$(191);" ";CHR$(191);CHR$(191);
     " ";CHR$(191)
2150 PRINT@201,CHR$(191);"##";CHR$(191);"##";CHR$
     (191);" ";CHR$(191);"###";CHR$(191);" ";CHR$(191):
     PRINT@233,"#####";CHR$(191);CHR$(191);" ";
2160 PRINTSTRING$(3,191);" ";CHR$(191);" ";CHR$(191)
2170 PRINT@265,CHR$(223);"##";STRING$(4,191);"##";CHR$
     (191);"####";CHR$(191):GOTO1960
2180 PRINT@41,"########";CHR$(191);" ";CHR$(191);"##";
     CHR$(223);CHR$(191):PRINT@73," ";STRING$(4,191);
     " ";STRING$(3,191);" ";CHR$(191);CHR$(191);"##";
2190 PRINTCHR$(191):PRINT@105,STRING$(11,32);CHR$
     (191);"##";CHR$(191):PRINT@137," ";STRING$(9,191);
     " ";CHR$(191);CHR$(191);" ";CHR$(191)
2200 PRINT@169," ";CHR$(191);" ";CHR$(191);STRING$
     (10,32);CHR$(191):PRINT@201,CHR$(191);CHR$(191);
     " ";CHR$(191);" ";CHR$(191);" ";STRING$(4,191);
2210 PRINT"##";CHR$(191);CHR$(191)
2220 PRINT@233,"#####";CHR$(191);"####";CHR$(191);CHR$
     (191);" ";CHR$(191);CHR$(191):PRINT@265,CHR$(223);
     CHR$(191);" ";STRING$(6,191);"#####";CHR$(191)
2230 GOTO1960
2240 PRINT@41,"####";CHR$(191);" ";CHR$(191);"######";
     CHR$(223);CHR$(191):PRINT@73," ";STRING$(4,191);
     "###";CHR$(191);" ";CHR$(191);" ";STRING$(3,191)
2250 PRINT@105,"##";CHR$(191);"###";CHR$(191);" ";
```

```
     STRING$(3,191);" ";STRING$(3,191):PRINT@137,"####"
     ;CHR$(191);" ";CHR$(191);"#####";STRING$(3,191)
2260 PRINT@169,STRING$(3,191);" ";STRING$(7,191);
     "###";CHR$(191):PRINT@201,"######";CHR$(191);"###"
     ;STRING$(3,191);" ";CHR$(191)
2270 PRINT@233,"##";STRING$(3,191);" ";CHR$(191);" ";
     CHR$(191);" ";CHR$(191);" ";CHR$(191);" ";CHR$
     " ";
2280 PRINT@265,CHR$(223);" ";CHR$(191);"#####";CHR$
     (191);"#####";CHR$(191):GOTO1960
2290 RETURN
2300 A$=INKEY$:IFA$=""THEN2300
2310 IFA$="R"THEND=-33
2320 IFA$="T"THEND=-32
2330 IFA$="Y"THEND=-31
2340 IFA$="F"THEND=-1
2350 IFA$="H"THEND=1
2360 IFA$="C"THEND=31
2370 IFA$="V"THEND=32
2380 IFA$="B"THEND=33
2390 IFA$="P"THENGOSUB3130
2400 IFA$="I"THENGOSUB3460
2410 IFA$="@"THEN3590
2420 IFA$="S"AND WAND=1 THEN 4200
2430 IFA$="G"AND EN<EX OR EN<100 THEN EN=EN+RND(10)*2
2440 IFA$="E"THENGOSUB4290
2450 RETURN
2460 PPRINT@320,"UP OR DOWN?";
2470 A$=INKEY$:IFA$=""THEN2470
2480 IFA$="D"THENLEVEL=LEVEL+1:GOTO2510
2490 IFA$="U"THENLEVEL=LEVEL-1:GOTO2510
2500 GOTO2470
2510 PRINT@320,"###########";:RETURN
2520 FORN=33TO288STEP32:PRINT@N,STRING$(30,128);CHR$
     (175);:NEXT
2530 FORN=0TORND(20)
2540 M=RND(319)+1024:IFPEEK(M)=128THENPOKEM,143+RND
     (7)*16:ELSEGOTO2540
2550 IFPEEK(M)=175THENPOKEM,143+RND(7)*16:GOTO2550
2560 NEXT
2570 PRINT@320,"IN A CAVE"
2580 MAN=1296
2590 POKEMAN,36
2600 GOSUB360
2610 GOSUB2300
2620 IFST>400THENST=400
2630 IFIQ>400THENIQ=400
2640 IFEN>400THENEN=400
2650 IFD=0THEN2610
2660 IFPEEK(MAN+D)=128THEN2750
2670 IFPEEK(MAN+D)<>175ANDRND(8)=5ANDMEDALLION<>1THEN
     PRINT@480,"YOU'VE FOUND THE MEDALLION!";:FORN=0TO
     2000:NEXT:PRINT@480,STRING$(31,32);:MEDALLION=1:
     GOTO2750
```

```
2680 IFPEEK(MAN+D)=175THEN2760
2690 M=RND(10):IFM=2THENPRINT@480,"YOU'VE FOUND A
     GEM!";:FORN=0TO2000:NEXT:PRINT@480,STRING$(19,32);:
     TR=TR+RND(300):GOTO2750
2700 IFM=4ANDWAND<>1THENPRINT@480,"YOU'VE FOUND THE
     MAGIC WAND!";:FORN=0TO2000:NEXT:PRINT@480,STRING$
     (28,32);:WAND=1:GOTO2750
2710 IFM=6ANDSHIELD<>1THENPRINT@480,"YOU'VE FOUND THE
     MAGIC SHIELD!";:FORN=0TO2000:NEXT:PRINT@480,
     STRING$(31,32);:SHIELD=1:GOTO2750
2720 IFM=8THENPRINT@480,"YOU'VE FOUND A MAGIC POTION!"
     ;:FORN=0TO2000:NEXT:PRINT@480,STRING$(31,32);:
     P=P+1:POTION(P)=RND(4):GOTO2750
2730 IFM=10THENGOSUB2770:GOTO2750
2740 PRINT@480,"NOTHING OF VALUE";:FORN=0TO2000:NEXT:
     PRINT@480,STRING$(31,32);
2750 MAN=MAN+D:POKEMAN-D,128:D=0:GOTO2590
2760 PRINT@320,"#########";:MAN=Q:Z=W:GOSUB1500:GOTO
     720
2770 SOUND10,5:SOUND100,6:RESTORE:IFEX<2000ANDTW=0THEN
     FORN=0TORND(19):READMONSTER$:READHITS:NEXT
2780 M=0
2790 IFEX>=2000ORTW=1THENFORN=0TORND(19)+20:READ
     MONSTER$:READHITS:NEXT
2800 IFSWORD=1THENHITS=HITS+RND(ST)
2810 FORN=1TOLEN(MONSTER$):MID$(MONSTER$,N,1)=CHR$(ASC
     (MID$(MONSTER$,N,1))+32):IFMID$(MONSTER$,N,1)="@"
     THENMID$(MONSTER$,N,1)=CHR$(128):NEXT:ELSENEXT
2820 PRINT@320,"YOU HAVE MET A ";MONSTER$;
2830 IFCLASS$="CLERIC"AND(MONSTER$="vampire"OR
     MONSTER$="wight"ORMONSTER$="mummy"ORMONSTER$=
     "wraith"ORMONSTER$="spectre")ANDRND(3)=2THENYZ=1
2840 IFYZ=1THENFORN=0TO2000:NEXT:PRINT@320,"BUT YOU
     TURN IT AWAY!":FORN=0TO2000:NEXT:PRINT@320," ":EX=
     EX+H:RETURN:YZ=0
2850 HITS=HITS+RND(INT(HITS/2)):IFHITS<INT(EN/2)THEN
     GOTO2850
2860 H=HITS
2870 FORN=0TO1000:NEXT
2880 PRINT@448,"strike";CHR$(128);"now"
2890 SOUND50,2
2900 A$=INKEY$
2910 PRINT@480,"############################";
2920 PRINT@480,"THE MONSTER HAS";HITS;"ENERGY  ";
2930 FORN=0TO300:A$=INKEY$:IFA$<>""THEN2940:ELSENEXT:
     PRINT@448,"too";CHR$(128);"slow":FORN=0TO2000:NEXT
     :PRINT@448," ":GOTO2990
2940 IFA$="S"ANDWAND=1THENSPELL(S)=SPELL(S)-1:GOSUB
     3250:IFV=1THENEX=EX+H:EN=EN+INT(RND(H/2)):YZ=1:V=0
2950 IFYZ=1THENFORN=0TO2000:NEXT:PRINT@320," ":PRINT@
     448," ":PRINT@480,STRING$(31,32);:YZ=0:RETURN
2960 IFA$="H"ANDRND(2)=2THENG=RND(EN)+RND(ST)*SD:
     PRINT@448,"you";CHR$(128);"hit":FORN=0TO2000:NEXT:
     PRINT@448,G;"DAMAGE!";:FORN=0TO3000:NEXT:YZ=1
```

```
2970 IFYZ=1THENYZ=0:PRINT@448," ":HITS=HITS-G:G=0:ELSE
     IFA$="H"THENPRINT@448,"you";CHR$(128);"missed":
     FORN=0TO2000:NEXT:PRINT@448,STRING$(30,32);:YZ=0
2980 IFG>0THENSOUND150,2:ELSESOUND200,2
2990 PRINT@480,"THE MONSTER HAS";HITS;"ENERGY ";
3000 IFHITS<=0THEN3110
3010 SOUND200,2:PRINT@448,"the";CHR$(128);"monster";
     CHR$(128);"strikes":FORN=0TO2000:NEXT:PRINT@448," "
3020 IFRND(2)=1THENM=RND(HITS):PRINT@448,"and";CHR$
     (128);"hits";:SOUND10,2:FORN=0TO2000:NEXT:ELSE
     PRINT@448,"and";CHR$(128);"misses";:SOUND50,2:YZ=1
3030 IFYZ=1THENFORN=0TO2000:NEXT:PRINT@448," ":YZ=0
3040 EN=EN-M:IFSHIELD=1THENEN=EN+INT(RND(M/2))
3050 IFM>0THENPRINT@448,"DOING";M;"DAMAGE!";:FORN=0TO
     2000:NEXT:PRINT@448," "
3060 GOSUB360
3070 M=0
3080 IFEN<=0ANDMEDALLION<>1THENGOTO3770:ELSEIFEN<=0
     THENPRINT@320,"YOU'VE GOT 16 MOVES TO GET HOME!";:
     FORN=0TO2000:NEXT:YZ=1
3090 IFYZ=1THENPRINT@320,STRING$(31,32);:PRINT@448,
     "":PRINT@480,STRING$(31,32);:DEAD=1:YZ=0:RETURN
3100 HITS=HITS-1:EN=EN-1:GOTO2880
3110 PRINT@448," ":PRINT@480,STRING$(31,32);:PRINT@
     320,"YOU'VE KILLED IT!":FORN=225TO250:SOUNDN,1:
     NEXT:EX=EX+H:EN=EN+RND(INT(H/2))
3120 FORN=0TO2000:NEXT:PRINT@320," ":IFEN>400THENEN=
     EN-RND(INT(H/2)):RETURN:ELSERETURN
3130 PRINT@448,"POTION NUMBER";:INPUTV:IFPOTION(V)=1
     THENPRINT@448,"IT'S POISONOUS!":FORN=0TO2000:NEXT:
     PRINT@448," "
3140 IFPOTION(V)=1THENEN=EN-RND(50):IFEN<=0THEN3080:
     ELSEPOTION(V)=0:RETURN
3150 IFPOTION(V)=2THENPRINT@448,"YOUR NOSE TURNS A
     FUNNY COLOUR!":POTION(V)=0:FORN=0TO2000:NEXT:
     PRINT@448," ":RETURN
3160 IFPOTION(V)=3THENPRINT@448,"IT DOES NOTHING!";:
     FORN=0TO2000:NEXT:PRINT@448," ":POTION(V)=0:RETURN
3170 IFPOTION(V)=4THENPRINT@448,"YOU'RE iq INCREASES!
     ";:FORN=0TO2000:NEXT:PRINT@448," ":IQ=IQ+RND
     (5)*10:POTION(V)=0:RETURN
3180 IFPOTION(V)=0THENPRINT@448,"THERE'S NOTHING IN
     IT!":FORN=0TO2000:NEXT:PRINT@448,"":POTION(V)=0:
     RETURN
3190 PRINT@448,"YOU GAIN";:L=RND(10)*2:PRINTL;"POINTS
     OF ENERGY!";:EN=EN+L:POTION(V)=0:FORN=0TO2000:
     NEXT:PRINT@448," ":RETURN
3200 DATA"BANDIT",20,"BERSERKER",20,"BUGBEAR",60,
     "CARRION CRAWLER",60,"COCKATRICE",100,"DWARF",20,
     "DOPPLEGANGER",80,"ELF",20,"FIRE BEETLE",20
3210 DATA"GARGOYLE",80,"GELATINOUS CUBE",80,"GIANT
     ANT",40,"GIANT CENTIPEDE",5,"GIANT RAT",10,"GNOLL"
     ,40,"GNOME",10,"GOBLIN",10,"GREY OOZE",60
3220 DATA"HIPPOGRIFF",65,"HOBGOBLIN",15,"BLACK
```

```
     PUDDING",200,"CHIMERA",180,"DJINNI",145,"DRAGON",
     220,"GIANT",200,"GRIFFON",140,"HYDRA",160
3230 DATA"WEREBEAR",120,"MANTICORE",125,"MINOTAUR",
     120,"MUMMY",105,"OGRE",120,"OWL BEAR",110,
     "PURPLE WORM",300
3240 DATA"SPECTRE",120,"TROLL",130,"VAMPIRE",180,
     "WIGHT",60,"WRAITH",80,"HELL HOUND",140
3250 PRINT@448,"SPELL NUMBER";:INPUTS
3260 IFS>3ANDEX<2000THEN3350
3270 IFS=1ANDRND(400)<IQ ANDSP(1)>0THENPRINT@448,"THE
     MONSTER FALLS ASLEEP!";:FORN=0TO2000:V=1:IQ=IQ+RND
     (5)*5:RETURN
3280 IFS=2ANDRND(400)<IQ ANDSP(2)>0THENPRINT@448,"THE
     MONSTER IS BLINDED AND RUNS OFF!";:FORN=0TO2000:
     NEXT:PRINT@448," ":PRINT@480,STRING$(31,32);:V=1
3290 IFV=1THENIQ=IQ+RND(5)*5:RETURN
3300 IFS=3ANDSP(3)>0ANDRND(400)<IQ THENPRINT@448,"THE
     MAGIC SHIELD HOLDS!";:FORN=0TO2000:NEXT:PRINT@448,
     " ":V=1:IQ=IQ+RND(5)*5:RETURN
3310 IFS=4ANDSP(4)>0ANDRND(400)>IQ THENPRINT@448,"THE
     WEB FALLS ON THE MONSTER!";:FORN=0TO2000:NEXT:
     PRINT@448," ":V=1:IQ=IQ+RND(5)*5:RETURN
3320 IFS=5ANDSP(5)>0ANDRND(400)>IQ THENPRINT@448,"THE
     MONSTER IS ENVELOPED IN#####DARKNESS!";:FORN=0TO
     2000:NEXT:V=1
3330 IFV=1THENPRINT@448," ":PRINT@480,STRING$(31,32);:
     IQ=IQ+RND(5)*5:RETURN
3340 IFS=6ANDSP(6)>0ANDRND(400)>IQ THENPRINT@448,"THE
     MONSTER TURNS INTO JELLY!";:FORN=0TO2000:NEXT:
     PRINT@448," ":IQ=IQ+RND(5)*5:V=1:RETURN
3350 IFSP(S)<1THENPRINT@448,"YOU HAVEN'T GOT THAT
     SPELL!";:FORN=0TO2000:NEXT:PRINT@448," ":RETURN
3360 PRINT@448,"THE SPELL FAILED!";:FORN=0TO2000:NEXT:
     PRINT@448," ":RETURN
3370 CLS:PRINT"WELL DONE! YOU'VE SUCCEEDED!";:PRINT
     "THE KING IS SAVED!!!":PRINT"AND YOU MANAGED TO
     RETURN WITH:-":PRINT"the";CHR$(128);"key"
3380 PRINT"THE MAGIC WAND":IFMEDALLION=1THENPRINT"THE
     MEDALLION OF LIFE"
3390 IFSHIELD=1THENPRINT"THE MAGIC SHIELD"
3400 IFAMULET=1THENPRINT"THE AMULET OF THE GODS"
3410 IFSD=1THENPRINT"THE MAGIC SWORD"
3420 FORN=1TO20:IFPOTION(N)>0THEN3440
3430 NEXT:GOTO3450
3440 PRINT"AND SOME POTIONS!"
3450 PMODE3,1:PRINT:PCLS:GOSUB3790:PMODE3,1:SCREEN1,0:
     FORN=0TO2000:NEXT:GOTO4190
3460 FORN=320TO448STEP32:PRINT@N," ":NEXT:PRINT@480,
     STRING$(30,32);:PRINT@320,"";
3470 IFWAND=1THENPRINT"WAND"
3480 IFSD=1THENPRINT"SWORD"
3490 IFMEDALLION=1THENPRINT"MEDALLION"
3500 IFAMULET=1THENPRINT"AMULET"
3510 IFSHIELD=1THENPRINT"SHIELD"
```

```
3520 IFKEY=1THENPRINT"the";CHR$(128);"key";
3530 FORN=0TO20:IFPOTION(N)>0THEN3550
3540 NEXT:GOTO3570
3550 FORN=0TO3000:NEXT:FORN=320TO448STEP32:PRINT@N,
     " ":NEXT:PRINT@480,STRING$(30,32);:M=0:FORN=0TO20:
     IFPOTION(N)>1THENM=M+1:NEXT:ELSENEXT
3560 PRINT@320,M;"POTIONS":M=0
3570 IFAMULET=0ANDSWORD=0ANDKEY=0ANDMEDALLION=0AND
     SHIELD=0ANDWAND=0THENPRINT"NOTHING!"
3580 FORN=0TO3000:NEXT:FORN=320TO448STEP32:PRINT@N,
     " ":NEXT:PRINT@480,STRING$(30,32);:GOSUB360:RETURN
3590 CLS:PRINT"IS TAPE READY ?"
3600 A$=INKEY$:IFA$<>"Y"THEN3600
3610 PRINT"PRESS play AND record ON TAPE":FORN=0TO
     3000:NEXT
3620 PRINT"SAVING ";NAME$;" THE ";CLASS$
3630 OPEN"O",-1,NAME$
3640 PRINT#-1,CLASS$;ST;IQ;EN;ST;EX;ME;SD;WA;SH;AM;KE;
3650 FORN=0TOP:PRINT#-1,POTION(P);:NEXT
3660 CLOSE#-1
3670 PRINTNAME$;" THE ";CLASS$;" SAVED"
3680 END
3690 CLS:INPUT"CHARACTER'S NAME";NAME$
3700 PRINT"PRESS play ON TAPE":FORN=0TO3000:NEXT
3710 OPEN"I",-1,NAME$
3720 INPUT#-1,CLASS$,ST,IQ,EN,ST,EX,ME,SD,WA,SH,AM,KE:
     FORN=0TO20
3730 IFEOF(-1)THEN3760
3740 INPUT#-1,POTION(N):NEXT
3750 CLOSE#-1
3760 GOTO190
3770 PRINT@320,"YOU'RE DEAD!":IFK=1THEN3770
3780 PLAY"O1V31T2L4GGL8GGL4B-AAGGF+G":K=1:GOTO3770
3790 DRAW"BM10,180;C4;R190U90L190D90R190E45U90G45"
3800 IFZZ=0THENCIRCLE(38,80),30,4,1.48,.46,.8
3810 IFZZ=0THENCIRCLE(226,80),30,4,1.48,.46,.85
3820 IFZZ=0THENLINE(40,38)-(230,38),PSET
3830 LINE(92,90)-(120,110),PSET,B
3840 PAINT(94,92),2,4
3850 COLOR3
3860 LINE(104,92)-(108,108),PSET,BF:CIRCLE(106,98),7
3870 PAINT(20,170),2,4:PAINT(220,80),2,4:IFZZ=0THEN
     PAINT(210,70),2,4:PAINT(40,80),2,4
3880 COLOR4
3890 N=30
3900 FORM=160TO176STEP4:LINE(N,M)-(210-N,M),PSET:
     N=N-4:NEXT
3910 M=20
3920 FORN=14TO30STEP4:LINE(N,M+95)-(N,195-M),PSET:
     M=M+4:NEXT
3930 N=30:FORM=130TO114STEP-4:LINE(N,M)-(210-N,M),
     PSET:N=N-4:NEXT
3940 M=20:FORN=196TO180STEP-4:LINE(N,M+95)-(N,195-M),
     PSET:M=M+4:NEXT
3950 LINE(14,92)-(36,112),PSET,BF:COLOR3:LINE(40,92)-
```

```
     (60,112),PSET,BF:COLOR1:LINE(64,92)-(84,112),PSET,
     BF
3960 LINE(128,92)-(148,112),PSET,BF:COLOR3:LINE
     (152,92)-(172,112), PSET, BF:COLOR4:LINE(176,92)-
     (196,112),PSET,BF
3970 PAINT(34,134),3,4:PAINT(28,136),4,4:PAINT
     (24,136),1,4:PAINT(16,136),4,4
3980 IF ZZ=1 THEN 4030
3990 FORN=38TO220STEP9:CIRCLE(N,80),30,4,1.48,.46,.8:
     NEXT
4000 FORN=40TO218STEP36:PAINT(N,75),4,4:NEXT
4010 FORN=20TO180STEP36:PAINT(N,75),3,4:NEXT
4020 FORN=34TO180STEP36:PAINT(N,75),1,4:NEXT
4030 COLOR2:LINE(12,91)-(198,91),PSET
4040 COLOR4:LINE(92,90)-(120,110), PSET,B
4050 IFZZ=0THENCOLOR1:LINE(38,37)-(230,37),PSET:LINE
     (38,36)-(230,36),PSET
4060 DRAW"BM208,158;C4;E27U64G27D64":PAINT(220,130),
     3,4
4070 IFZZ=0THENZZ=1:RETURN
4080 COLOR4
4090 DRAW"BM10,90;C4;E45R190L190D44"
4100 DRAW"BM10,90;C4;E85G40R190E10G10"
4110 DRAW"BM10,90;C4;E85R190"
4120 DRAW"BM200,90;C4;E45R2D1R4D1R2U1R2U1R2"
4130 PAINT(25,80),2,4:PAINT(60,80),2,4:PAINT(90,20),2,
     4:PAINT(246,44),2,4
4140 CIRCLE(125,125),500,3
4150 CIRCLE(150,90),15,4,1,.5,0:PAINT(152,85),3,4:
     CIRCLE(150,90),15,3,1,.5,0
4160 COLOR3:LINE(146,85)-(154,70), PSET,BF:COLOR1:LINE
     (146,69)-(154,64),PSET,BF
4170 PMODE3,1:SCREEN1,0
4180 GOTO4180
4190 CLS0:PRINT@227,"THE CHEST WILL BE OPENED IN A";:
     PRINT@269,"MOMENT";:PCLS:GOTO3790
4200 FORN=320TO448STEP32:PRINT@N,"":NEXT
4210 PRINT@480,STRING$(30,32);
4220 FORN=1TO6
4230 PRINT@320+((N-1)*32),"SPELL";N;"=";SPELL(N);
4240 NEXT
4250 FORN=0TO2000:NEXT
4260 FORN=320TO448STEP32:PRINT@N,"":NEXT
4270 PRINT@480,STRING$(30,32);
4280 RETURN
4290 FORN=320TO448STEP32:PRINT@N,"":NEXT
4300 PRINT@480,STRING$(30,32);
4310 PRINT@320,"";
4320 IFEX<500THENPRINT"FISH FOOD":GOTO4580
4330 IFEX<1000THENPRINT"SWORD PRACTICE DUMMY":GOTO4580
4340 IFEX<2000THENPRINT"APPRENTICE FOOL":GOTO4580
4350 IFEX<3000THENPRINT"SNAIL SLAYER":GOTO4580
4360 IFEX<4000THENPRINT"DRAGON'S TOY":GOTO4580
4370 IFEX<5000THENPRINT"APPRENTICE SWORDSMAN":GOTO4580
```

```
4380 IFEX<6000THENPRINT"WOLF MASTER":GOTO4580
4390 IFEX<7000THENPRINT"SWORDSMAN":GOTO4580
4400 IFEX<8000THENPRINT"LION TAMER":GOTO4580
4410 IFEX<9000THENPRINT"3RD RATE HERO":GOTO4580
4420 IFEX<11000THENPRINT"MASTER OF THE SWORD":GOTO4580
4430 IFEX<12000THENPRINT"2ND RATE HERO":GOTO4580
4440 IFEX<13000THENPRINT"LORD OF THE PATH":GOTO4580
4450 IFEX<14000THENPRINT"LORD OF THE KEEP":GOTO4580
4460 IFEX<16000THENPRINT"GOBLIN SLAYER":GOTO4580
4470 IFEX<18000THENPRINT"CHAMPION":GOTO4580
4480 IFEX<22000THENPRINT"HERO - 1ST CLASS":GOTO4580
4490 IFEX<26000THENPRINT"DRAGON SLAYER":GOTO4580
4500 IFEX<30000THENPRINT"WARLORD":GOTO4580
4510 IFEX<35000THENPRINT"LORD OF THE HEROS":GOTO4580
4520 IFEX<40000THENPRINT"LORD OF THE PALACE":GOTO4580
4530 IFEX<45000THENPRINT"DEATH DEFYER":GOTO4580
4540 IFEX<50000THENPRINT"MONSTER TAMER":GOTO4580
4550 IFEX<60000THENPRINT"PRINCE OF LIGHT":GOTO4580
4560 IFEX<70000THENPRINT"RULER OF THE VALLEY":GOTO4580
4570 PRINT"MASTER OF ETERNITY"
4580 FORN=0TO2000:NEXT
4590 PRINT@320,""
4600 U=1:GOSUB360:RETURN
```

Code Breaker

Code Breaker is a version of the popular board game Mastermind. The idea of the game is to guess a code number which the computer has chosen, helped along by clues given by the computer.

When you RUN the program you will be asked how many digits you want in the code and then for the highest number that you want in the code. If you want the highest number to be 7 then all the numbers in the code will be between 1 and 7. Finally you will be asked how many guesses you want before being presented with the instructions for the game.

Once you have finished reading the instructions the screen will clear and you will be asked for your first guess. The computer will then tell you how many blacks and whites you got.

A black represents a digit which is in the code and is also in the right place. A white represents a digit which is in the code but not in the right place. For example:

The code which you are trying to guess is 81632
Your guess is 82945
You have one black (the 8 is in the right place)
You have one white (the 2 is in the computer's code but in the wrong place).

This process continues until you either guess the code correctly or run out of guesses. If you manage to guess the code correctly the screen will flash several colours accompanied by random noises. The first bar of Congratulations will then be played and you will be told how good you are at the game. You will then be asked whether or not you want another go.

If you run out of guesses you will be told the computer's code and asked if you want another go.

```
10 CLS
20 PRINT"HOW MANY DIGITS DO YOU WANT IN  THE CODE";
30 INPUTDIGIT
40 PRINT"WHAT DO YOU WANT THE HIGHEST"
50 PRINT"NUMBER IN THE CODE TO BE (1-10)";
60 INPUTHIGHEST
70 INPUT"HOW MANY GUESSES DO YOU WANT";NUMBER
80 DIMGUESS$(NUMBER),HIGHEST(DIGIT),D(DIGIT)
90 DIMBLACK(NUMBER),WHITE(NUMBER)
100 GOSUB710
110 GOSUB220:GOSUB240
120 PRINT"GUESS NUMBER";GUESS;
130 INPUT A$
140 IF LEFT$(A$,1)="Q" THEN 370
150 GOSUB480
160 GOSUB570
170 BLACK(GUESS)=BLACK:WHITE(GUESS)=WHITE
180 IF BLACK(GUESS)=DIGIT THEN 1000
190 GUESS$(GUESS)=A$
200 GUESS=GUESS+1:IF GUESS>NUMBER THEN 1140
210 GOTO300
220 GUESS=1:D$=""
230 RETURN
240 FOR H=1 TO DIGIT
250 S=RND(HIGHEST)
260 D$=D$+MID$(STR$(S),2,1)
270 NEXT
280 PRINT"I'VE CHOSEN MY SECRET CODE!"
290 RETURN
300 CLS
310 PRINT"NO.   GUESS   BLACK   WHITE"
320 FOR H=1 TO GUESS-1
330 PRINTH;TAB(7);GUESS$(H);TAB(15);BLACK(H);
340 PRINTTAB(23);WHITE(H)
350 NEXT:PRINT
360 GOTO 120
370 CLS:FOR N=50TO2STEP-2:SOUNDN,1:NEXT
380 SOUND255,2
390 PRINT"YOU'RE NOT GOOD ENOUGH!"
400 PRINT"MY SECRET CODE WAS ";
410 FOR H=1 TO 6
420 PRINT"  .";
430 SOUND H*40,1
```

```
440 FOR L=1 TO 900:NEXT
450 NEXT
460 PRINTD$:PRINT
470 GOTO1090
480 IF LEN(A$)<>DIGIT THEN 540
490 FOR H=1 TO DIGIT
500 S=VAL(MID$(A$,H,1))
510 IF S<1 OR S>HIGHEST THEN 540
520 NEXT
530 RETURN
540 PRINT"THAT'S NOT A LEGAL GUESS!"
550 PRINT"PLEASE TRY AGAIN"
560 GOTO120
570 BLACK=0:WHITE=0
580 FOR H=1 TO DIGIT
590 GUESS(H)=VAL(MID$(A$,H,1))
600 D(H)=VAL(MID$(D$,H,1))
610 IF GUESS(H)=D(H) THEN BLACK=BLACK+1:GUESS(H)=0:
    D(H)=0
620 NEXT
630 FOR H=1 TO DIGIT:IF D(H)=0 THEN 690
640 I=0:FOR L=1 TO DIGIT
650 IF D(H)=0 THEN 680
660 IF D(H)<>GUESS(L) THEN 680
670 I=1:GUESS(L)=0:D(H)=0
680 NEXT L:WHITE=WHITE+I
690 NEXT H
700 RETURN
710 CLS
720 PRINT"**********code*breaker**********"
730 PRINT"I WILL PICK A";DIGIT;"DIGIT CODE"
740 PRINT"WITH THE NUMBERS IN THE CODE"
750 PRINT"RANGING FROM 1 TO";HIGHEST;"."
760 PRINT"YOU MUST TRY AND GUESS THIS CODE"
770 PRINT"TO HELP YOU I WILL TELL YOU HOW"
780 PRINT"MANY NUMBERS YOU GOT RIGHT, AND"
790 PRINT"HOW MANY NUMBERS YOU GOT RIGHT"
800 PRINT"BUT IN THE WRONG PLACE. HOWEVER,";
810 PRINT"I WON'T TELL YOU WHICH NUMBERS"
820 PRINT"(IF ANY) YOU GOT RIGHT!"
830 PRINT:PRINT"PRESS ANY KEY TO CONTINUE";
840 IF INKEY$=""THEN 840
850 CLS
860 PRINT"I WILL TELL YOU HOW MANY NUMBERS";
870 PRINT"YOU GOT RIGHT IN THIS WAY:"
880 PRINT
890 PRINT"black IS THE NUMBER OF CORRECT"
900 PRINT"NUMBERS IN THE CORRECT PLACES"
910 PRINT
920 PRINT"white IS THE NUMBER OF CORRECT"
930 PRINT"NUMBERS IN THE WRONG PLACE"
940 PRINT
950 PRINT"PRESS ANY KEY TO START";
960 IF INKEY$=""THEN960
```

```
970 CLS
980 RETURN
990 END
1000 FORN=230TO255:SOUNDN,1:CLS RND(8):NEXT
1010 PLAY"T11L1GABO3L1.CO2L1G"
1020 CLS:PRINT"YOU GOT IT IN";GUESS;"GUESSES."
1030 IF GUESS<5 THEN RATING$="FANTASTIC!"
1040 IF GUESS=5 OR GUESS=6 THEN RATING$="REASONABLE!"
1050 IF GUESS=7 THEN RATING$="AVERAGE!"
1060 IF GUESS=8 THEN RATING$="GOOD FOR A BEGINNER!"
1070 IF GUESS>8 THEN RATING$="TERRIBLE!"
1080 PRINT"....THAT'S ";RATING$
1090 INPUT"WANT TO TRY AGAIN (Y/N)";A$
1100 IF LEFT$(A$,1)="Y" THEN 100
1110 IF LEFT$(A$,1)<>"N" THEN 1090
1120 PRINT"COWARD!"
1130 END
1140 PRINT
1150 PRINT"YOU'VE RUN OUT OF GUESSES!"
1160 GOTO390
```

Commentary

Line 10 clears the screen before lines 20–70 ask you how many digits you want in the code, the highest number that you want in the code and how many guesses you want. Lines 80 and 90 then DIMension the variables which will be used in the program, using your answers to the previous questions as guidelines.

Line 100 sends the program to the subroutine starting at line 710 (the instructions). Line 110 then sends the computer to two subroutines which set up the computer's secret code.

The main routine lies from lines 120–210. Line 120 tells you which guess you are on and then line 130 asks you for your guess. Line 140 checks to see if you want to quit, jumping to line 370 if you do. Lines 150–160 send the computer to two subroutines which check that your guess is a legal one and work out how many blacks and whites you got. Line 170 then keeps a record of how many blacks and whites you got in that go for future reference. Line 180 checks to see if you correctly guessed the code, sending the computer to line 1000 if you have. Line 190 keeps a record of your guess (again for future reference). Line 200 makes sure that you haven't used up all your guesses yet before line 210 sends the program to line 300 to display the record of your attempts at guessing the code.

Line 220 resets two of the variables and lines 240–270 set up the computer's secret code, with line 280 telling you when the code is worked out. The record of your guesses is displayed by lines 300 – 350.

Lines 370–460 make up the losing routine. Line 370 clears the screen and makes a series of noises. Line 380 then makes a final bleep. You are told by line 390 that you are not good enough and then lines 410–450 slowly PRINT a series of dots on the screen accompanied by increasingly higher notes. Line 460 tells you what the secret code was and then line 470 sends the program to line 1090 to ask whether or not you want another go.

Line 480 makes sure that your guess has the right number of digits in it, jumping to line 540 if it hasn't. Lines 490 – 520 then check each digit in your guess to make sure that none of them are too high. If any are then the computer jumps to line 540.

Lines 540 – 550 tell you that you have made an illegal guess and then line 560 sends the computer back to the main program. Lines 570 – 690 work out how many blacks and whites you got.

Lines 710–970 display the instructions. Line 1000 flashes the screen dif ferent colours while making random noises before line 1010 plays the first bar of Congratulations. You are then told how good you are at the game by lines 1020 – 1080.

Lines 1090 – 1120 asks you whether or not you want another game, jumping to line 100 if you do and PRINTing 'COWARD!' if you don't. Lines 1140 – 1150 tells you that you have run out of guesses.

Revision Aid

Revision Aid is for all those among you who are learning, or have someone in the family who is learning, another language. The program allows you to enter a set of words and their meanings and then tests you on the words, choosing one at random and asking you for its meaning. The words may also be stored on tape for future use.

When you RUN the program you will be asked whether you want to load a set of words or set up some new ones. At first you won't have any words on tape so you will reply S , but if you do have some stored on tape you should press L.

If you want to set up a new set of words then you will be asked how many words you want to learn. You will then be asked to enter the words, one by one. Each word should be entered like this:

AVOIR:TO HAVE

It doesn't matter whether the foreign word or the English translation comes first, as long as each is separated by a colon *and only a colon*. No spaces should be added in between the words and the colon.

Once you have entered all the words you will be asked whether or not you want to save them. Assuming that you don't, the screen will then clear and the computer will start testing you on the words. You will be randomly tested from English into the other language, and from the language that you are learning into English. You can have up to five attempts at each word,x and if you don't know the word you may type H (for Help). The computer will then tell you what the word is.

After each word you will be asked whether you want to be tested on another. If you do then the program will continue, otherwise you will be told how well you have done and the program will end.

If you decide to save your list of words then the screen will clear and you will be asked to prepare the tape. When you have done this you will be asked if the tape is set on record or not, before being asked for the name that you want the words to be saved under. Your list of words will then be recorded and the program will continue as normal.

If you load a set of words then you will be asked to go through a similar routine to that carried out when saving the list. You will be asked to set the tape on play before being asked for the file name. Your list will then be loaded and the program will continue.

If you want more (or less) than five attempts at guessing a word then you should alter the 5 in lines 260 and 340 to the number of guesses that you want.

You may make the computer test you on 10 words without asking you if you want another test by altering these lines:

390 IF RIGHT + WRONG = 10 THEN 430
400 GOTO 140
DELETE LINES 410–420

```
10  CLS:PRINT@10,"revision";CHR$(128);"aid"
20  INPUT"LOAD WORDS OR SET UP NEW ONES    (L/S)";A$
30  IFA$="L"THEN720
40  IFA$<>"S"THEN20
50  PRINT"HOW MANY WORDS DO YOU WANT TO    LEARN"
60  INPUT NUMBER
70  PRINT"PLEASE ENTER WORDS AND THEIR     MEANING"
80  DIM WORD$(NUMBER)
90  FOR N=1 TO NUMBER
100  LINEINPUT"?";WORD$(N):NEXT
110  INPUT"DO YOU WISH TO SAVE THESE WORDS (Y/N)";A$
120  IFA$="Y"THEN540
130  IFA$<>"N"THEN110
```

```
140 CLS:X=RND(NUMBER)
150 Z=1:S=0
160 IF MID$(WORD$(X),Z,1)=":" THEN 190
170 Z=Z+1
180 GOTO 160
190 IFRND(2)=2THEN290
200 PRINT"PLEASE TRANSLATE
210 PRINTRIGHT$(WORD$(X),LEN(WORD$(X))-Z)
220 INPUTANSWER$
230 IFANSWER$="H"THEN520
240 IFANSWER$=LEFT$(WORD$(X),Z-1)THEN380
250 PRINT"WRONG!":PRINT"TRY AGAIN":WRONG=WRONG+1
260 S=S+1:IFS=5THENPRINT"THE ANSWER IS ";:GOTO360
270 GOTO200
280 END
290 PRINT"PLEASE TRANSLATE ";LEFT$(WORD$(X),Z-1)
300 INPUT ANSWER$
310 IF ANSWER$="H" THEN 360
320 IF ANSWER$=RIGHT$(WORD$(X),LEN(WORD$(X))-Z)THEN380
330 PRINT"WRONG!":PRINT"TRY AGAIN":WRONG=WRONG+1
340 S=S+1:IF S=5 THEN PRINT"THE ANSWER IS ";:GOTO360
350 GOTO290
360 PRINTRIGHT$(WORD$(X),LEN(WORD$(X))-Z)
370 WRONG=WRONG+1:GOTO390
380 PRINT"RIGHT!":RIGHT=RIGHT+1
390 PRINT"ANOTHER (Y/N)?";
400 A$=INKEY$
410 IF A$="Y" THEN 140
420 IF A$<>"N" THEN 400
430 CLS:PRINT@256,"YOU GOT";RIGHT;"RIGHT OUT OF";
440 PRINTRIGHT+WRONG
450 IF RIGHT>WRONG THEN 490
460 IF WRONG-RIGHT<5 THEN PRINT"PRETTY BAD!":END
470 IF WRONG-RIGHT<10 THEN PRINT"TERRIBLE!":END
480 PRINT"RIDICULOUS!":END
490 IF RIGHT-WRONG>5 THEN PRINT"QUITE GOOD!":END
500 IF RIGHT-WRONG>10 THEN PRINT"EXCELLENT!":END
510 PRINT"AVERAGE!":END
520 PRINTLEFT$(WORD$(X),Z-1):WRONG=WRONG+1
530 GOTO390
540 CLS:PRINT"PLEASE PREPARE TAPE AND THEN"
550 PRINT"PRESS 'R'"
560 IFINKEY$<>"R"THEN560
570 PRINT"IS TAPE SET ON RECORD (Y/N)?"
580 A$=INKEY$
590 IFA$="Y"THEN620
600 IFA$<>"N"THEN580
610 GOTO510
620 INPUT"FILE NAME";NAME$
630 PRINT"SAVING ";NAME$;" NOW"
640 OPEN"O",#-1,NAME$
650 PRINT#-1,NUMBER
660 FORN=1TONUMBER
670 PRINT#-1,WORD$(N);
```

```
680 NEXT
690 PRINTNAME$:" SAVED"
700 CLOSE#-1
710 GOTO140
720 CLS:PRINT"PLEASE SET TAPE ON PLAY"
730 PRINT"PRESS 'R' WHEN READY"
740 IF INKEY$<>"R"THEN740
750 PRINT"PLEASE ENTER FILE NAME";
760 INPUT NAME$
770 PRINT"LOADING ";NAME$
780 OPEN"I",#-1,NAME$
790 INPUT#-1,NUMBER
800 DIMWORD$(NUMBER)
810 FORN=1TONUMBER
820 INPUT#--1,WORD$(N)
830 NEXT
840 PRINTNAME$;" LOADED"
850 CLOSE#-1
860 X=NUMBER
870 FORN=0TO1000:NEXT
880 GOTO140
```

Commentary

Line 10 clears the screen and displays the title of the program. You are then asked whether you want to load a set of words or set up some more by line 20. Lines 30–40 check that your response is a legal one and react accordingly.

Lines 50–60 ask you how many words you want to learn before lines 70–100 INPUT the words. You may notice that we have used the LINE INPUT command rather than INPUT. This allows us to enter the colon in between the words and their meanings, something which the INPUT command doesn't allow.

You are asked by line 1 0whether or not you want to save your set of words, and lines 120–130 react accordingly to your response. Line 140 clears the screen and picks a random word. Lines 150–180 find where the colon is positioned in the word and then line 190 decides whether to ask you to translate from or into English.

Lines 200–210 tell you the word which you are expected to translate, before you are asked for an answer by line 220. Line 230 makes sure that you're not asking for Help, and then Line 240 checks to see if you have got the word right. You are told that you have got the word wrong by Line 250 and Line 260 makes sure that you haven't run out of guesses, telling you the answer if you have.

Lines 290–350 are similar to lines 200–260 except that they ask you to translate the word in the other way.

Lines 360–370 tell you what the word which you are translating is, and line 380 tells you that you are right, increasing the value of the variable RIGHT as it does so. Lines 390–420 ask you whether or not you want another go and act on your response.

Lines 430–440 tell you how many you got right before lines 450–510 tell you how well you done. Lines 520–530 tell you the word which you are trying to translate.

Lines 540–610 make sure that you have the tape recorder set up correctly for saving the list of words. Line 570 asks you for the file name and then your list of words is saved on tape by lines 640–690.

Lines 720–740 make sure that you have the tape recorder correctly set up for loading in a set of words. Lines 750–760 ask you for the name that the words were saved under before lines 780–850 load in the list of words which you require. Line 870 then pauses before the computer is sent back to the main program by line 880.

APPENDIX D
Jargon Guide

If there's one thing that the computer world is full of it's jargon, and here is a guide to help you through this foreign language:

Acoustic coupler — device connected to a computer into which a telephone hand set fits. Allows computers to communicate over the telephone.
Address — an index number to memory locations, usually in binary or hexadecimal (base 16).
Assembly language — a programming language in which processes are carried out by altering memory addresses using symbolic instructions.
BASIC — Beginner's All Purpose Symbolic Instruction Code. The language which most micro-computers use, and the one which this manual teaches you.
Bit — a single binary number, either one or zero.
Bug— an error in a program, either causing it to work incorrectly or not at all.
Byte— a binary number made up of eight bits (usually). A byte can represent any number from 0 to 255 as there are 256 combinations of eight ones and zeros.
Cartridge— a unit composed of either ROM or RAM (or both) which can be plugged into a computer providing a program or extra memory.
Character set — the set of letters, numbers and symbols which are available from the computer.
CP/M — Control Program for Microcomputers. A standard disc operating system which is available on many Z80 based computers. As it is a standard language software can be easily transferred from one CP/M system to another.
CPU — Central Processing Unit. The chip at the heart of a computer which controls everything.
Cursor - character which indicates where the next piece of information will appear on the screen.
Data — information.
Debug — to remove errors from a program.
Disk — a magnetic device for the storage of programs and data. Allows very fast access to a large amount of information. (Most disk units can access information in seconds).

DOS — Disk Operating System. A program either dumped into RAM or held on ROM which controls the operation of disk.
EEPROM — Electrically Erasable Read Only Memory. Similar to a ROM but can be erased by electrical impulses.
EPROM — Erasable Programmable Read Only Memory. A memory device similar to ROM which can be erased by exposure to ultra-violet light.
Floppy disk — a magnetic-coated disk on which programs and information can be stored.
Hard copy — a printout of a program or other information on paper.
Hard disk — similar to a floppy disk but is fixed permanently inside the disk drive. Capable of storing much more information than a floppy disk.
Hardware — all the actual physical components of a computer system eg the keyboard.
Hex or Hexadecimal — base 16. A means of counting in 16s opposed to 10s using the numbers 0–9 and then the letters A-F (A = 10, B = 11 etc).
High Resolution — refers to the size of any single point which can be lit up. The smaller the point the higher the resolution.
Instruction — a set of bits which give the CPU a command to carry out.
Interface — a unit which allows the computer to be connected to another unit eg a printer.
I/O — Input/Output. A series of ports which allow the computer to interface with a device and lets the device send information back to the computer.
Kilobyte (K) — 1024 bytes of memory.
Language — a series of commands which combine to make up a program.
Machine language or machine code — the language in which the CPU works is made up of a series of hexadecimal numbers.
Memory map — table showing how the computer's memory is divided up.
Modulator — device inside the computer which turns the computer's output signal into a form which can be displayed on the television.
Modem — unit which allows computers to communicate over a normal telephone line. Must be used with an RS232 interface and British Telecom's permission must be obtained before using one.
Monitor — either a program which allows you to alter the contents of the RAM using machine code, or a TV-type unit which does the same job as the television but produces a much higher quality picture.
Parallel/Serial — the means by which a computer outputs information. A parallel interface sends information out along a series of wires, whereas a serial device uses fewer wires and sends the data out one bit at a time.
Pascal — very powerful high level language used on some computers.
Peripheral — device which connects to a computer such as a printer or disc unit.
Pixel — single dot which is displayed on the screen. Pixels are lit up in groups to form characters and pictures.

Port — a kind of window to the outside world through which information can be output and input.
Printout — same as hard copy.
Program — set of instructions which combine to make the computer carry out a useful (?) task.
PROM — Programmable Read Only Memory. A special form of ROM which can be programmed.
QWERTY — the standard typewriter style keyboard layout.
RAM — Random Access Memory. Form of memory which can have its contents altered by programming and can also have its contents read. Anything stored in RAM is lost when the power to it is stopped.
Register — a memory location in the CPU which has a specific purpose in the controlling of the computer.
ROM — Read Only Memory. Form of memory which can have its contents read but not altered.
Routine— a program, or part of the program, designed to perform a single task.
RS232 — a form of interface used for serial input and output.
Software — a program of one kind or another. Software is always stored on some kind of hardware, such as a tape, ROM or RAM.
Source code — a program which has been written in a high level language, such as BASIC, and needs to be converted into machine code.
String — a series of characters.
Stringy floppy — half way between a floppy disk and a tape. A continuous loop of tape which can be read and written to much faster than a normal tape. Must be used with a proper stringy floppy drive.
Subroutine — a program within a program. A small part of the program which has one specific task to fulfil.
Syntax — the form in which a programmable language must be.
Toolkit — a program which adds to a computer's set of commands.
Utility — a useful command, or set of commands.
Variable — a symbol or combination of symbols which is used to represent a number.
VDU — Visual Display Unit. Either a TV or monitor on which information from the computer can be displayed.
Z80 — very popular CPU which is used in many computers, such as the ZX Spectrum, TRS-80 and Aquarius.
6502 — another popular CPU, used in such computers as the Atom, Pet and Oric computers.
6809 — the CPU inside your Dragon.

Index

The Dragon Trainer is written as a combined manual and beginner's course on the power of Dragon BASIC. It is aimed at the complete beginner and assumes no previous experience of computing.

Brian Lloyd starts by explaining how to set the computer up. He then introduces the first principles of simple computer programming.

Page by page The Dragon Trainer works through each of the programming features available on the Dragon, in each case giving practical examples of the ways in which the features can be used in your own programs.

Once all the basic principles have been mastered Brian Lloyd shows how you can experiment with more sophisticated tasks such as programming in high resolution graphics, vital for your own games and applications.

To help you on your way to writing your own programs the book ends with a series of games which you can type in and play.

SUNSHINE

ISBN 0 946408 09 2

£5.95 net

BRIAN LLOYD

THE DRAGON TRAINER

SUNSHINE