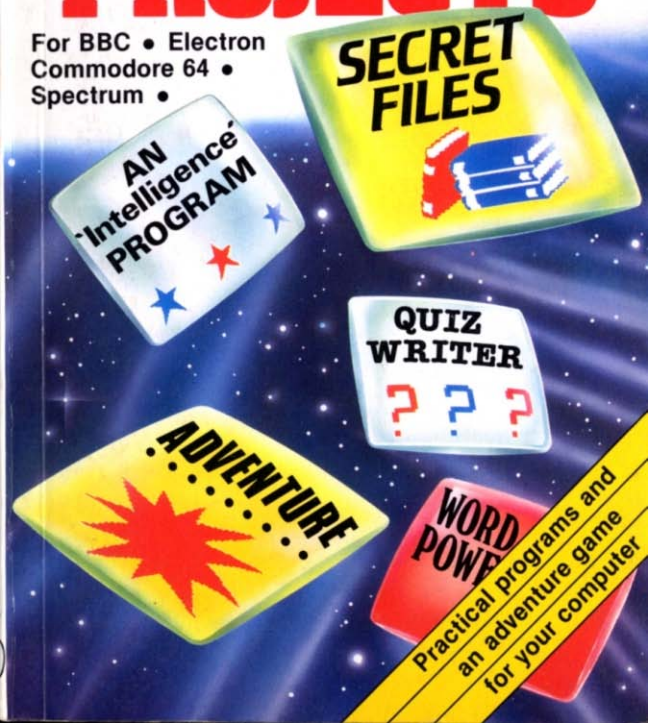
A PICCOLO FACTBOOK

COMPUTER PROJECTS

For BBC • Electron
Commodore 64 •
Spectrum •



Practical programs and an adventure game for your computer

First published 1985 by Piper Books Ltd
and exclusively distributed under the
Piccolo imprint by Pan Books Ltd,
18–21 Cavaye Place, London SW10 9PG

ISBN 0 330 28517 3

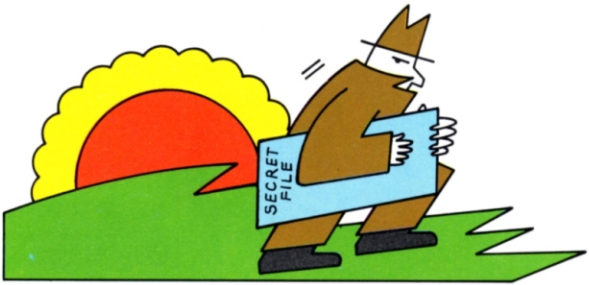
9 8 7 6 5 4 3 2 1

Printed in Spain by Graficas Reunidas S.A., Madrid.

COMPUTER PROJECTS

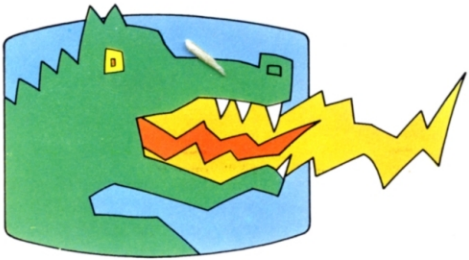
Piccolo
A Piper Book

Contents



COMPUTER PROJECTS

By Malcolm Neave



Editor: Jacqui Bailey
Designer: David Jefferis

Piccolo
A Piper Book

Helpful Hints

The five programs in this book have all been chosen to illustrate the different ways that computers can be used — in research, in education, in business and just for fun.

The programs are written in **BASIC**, which is the computer language that almost all home computers use. Each different make of computer, however, uses a slightly different version of BASIC (known as its 'dialect'). For this reason, some of the lines given in the main programs may have to be altered for your computer.

The main programs have been written for use with the BBC and ELECTRON computers. If you have a ZX SPECTRUM or a COMMODORE 64, you will need to change those lines in the programs that are marked:

■ — for Commodore 64
▲ — for Spectrum
● — for both machines

The lines to be inserted in place of these marked lines are given at the end of each of the main programs.

Saving and Loading Data Files

You will need a tape recorder when using this book, as all but one of the programs allow you to set up your own information files, which you can use again and again.

For example, in the 'Quiz Writer' program, you will be asked to write your own questions and answers for the quiz. This means that you can write any number of different quizzes, using the same basic program. You can then save each quiz game that you write onto a tape, and load it back into the program next time you want to use it.

Saving and loading information in this way is known as setting up a 'Data File'. **Data** simply means any additional information which you put into your computer to use with a program.

The program will tell you when you can save the data file, and will ask you for a name to store with the file. You can choose whatever name you like, as long as it is *not more than six letters long*. When you want to load that file back in again, the program will ask you to type in the file name; it will then be able to find that file on the tape — providing you have put the right tape into the recorder of course! Otherwise, the basic procedure for saving and loading is the same as that described in the manual which came with your machine.

It is a good idea to save the program listings for each program, as well — you do not want to have to type in the listings again each time you want to run the program! Save the listings as soon as you have finished typing them in. Or, if you don't want to type in a program in one go, you can save what you have done and then load it back in to continue later. Put the programs on a separate tape from your files, and use the procedure given in your manual.

Note for Spectrum Users

When saving data files, the Spectrum will save each part of the file separately. It will save the first part and then you will see the 'Start tape' message reappear on the screen. Simply press ENTER again and the Spectrum will go on to save the second part of the file, and so on.

Keep pressing ENTER each time the 'Start tape' message reappears, until either the screen displays a 'STOP' message, or you are taken back into the main program.

Typing in Programs

When typing in programs from a book or magazine, you must type in the lines

exactly as shown. Be especially careful to distinguish between the number 0 (shown as Ø) and the letter O, the number 1 and the letter I, the number 5 and the letter S, and the punctuation marks for colon (:) and semicolon (;). It is also important to put the *spaces* between the letters, numbers or symbols in the right places, especially on the BBC and ELECTRON machines.

When you have finished typing in a program, if you type RUN, and the program does not work you have probably made a typing error.

Most machines will often give you a message about where the error might be, such as 'No variable at line. . . .', or 'Syntax error at line. . . .'. In any event, you will have to check through the program lines on your screen very carefully to find the error — it could be as simple as missing out some quotation marks, or forgetting to type in a line number!

Learning About Programming

As well as the program itself, each chapter contains a description of how to use the program once you have got it typed in and running, and detailed 'Line Notes' to help you understand what each line in the program does.

However, there are a few terms which are used regularly throughout all the programs and which you may not be familiar with. If so, a description of each of these terms is given below.

About Variables

A program can be made to store information (such as numbers, symbols, letters or words) in a number of ways.

If you want your program to store an amount of something that is fixed and will not change, you simply use the numbers that express that amount, such as 45, 3.7, or −27. These numbers are known as **Constants**.

However, if you want to be able to change the value of something this is known as a **Variable**, and you give that variable a code name, such as NQ or A. You can then give the variable a value, for example, LET A=4 gives A the value 4, and then you can change the value of A later on in the program by saying LET A=A+6. This would then give A the value 10.

You can also allow the user to decide what the value will be, while the program is

running, by using the command INPUT. For example, in the 'Quiz' program, line 130 prints a message on the screen asking the user how many questions they wish to put into their quiz. This is then followed by the command INPUT NQ, which means that the computer will take the number typed in by the user, 10 for example, and will give the variable NQ (for Number of Questions) the value 10.

String Variables

String variables are used to store single letters or punctuation marks, or groups of letters and numbers, such as 'FRED', '31ST' or 'TRY–3'. String variables use code names ending with the symbol $, such as A$ or N$. If you give a string variable a specific meaning, that meaning must be enclosed by quotation marks, such as LET N$="FRED".

You can also allow the user to assign a meaning to a string. For example, you may have a program line:

```
10 PRINT "WHAT IS
   YOUR NAME ?":
   INPUT N$
```

If you then type in the name JANE, N$ will contain "JANE", so if you then have a line in the program saying:

```
20 PRINT "HELLO";N$
```

the computer will print out HELLO JANE.

There are four particular commands which are only used with strings:

1) LEN gives you the length of a string, so if N$ contains the word "FREDA", then LEN(N$) will contain the number 5.

2) LEFT$ makes a new string using the letters starting from the left of the string, so LEFT$(N$,4) will contain the word "FRED".

3) RIGHT$ uses characters (letters, numbers or symbols) from the right end of the string, so RIGHT$(N$,2) will contain "DA".

4) MID$ takes characters from the middle of the string, so MID$(N$,2,3) will contain the three characters starting from the second character in the word, i.e. "RED", or MID$(N$,3,1) will be "E".

The Spectrum uses the command TO, instead, to perform the last three of these functions. So N$(TO 4) will give "FRED", N$(4 TO) will give "DA", N$(2 TO 4) will give "RED" and N$(3) will give "E".

About Arrays

An **array** is used to store a list of things, such as names,

dates, or a list of numbers, in the computer's memory so that it can find them again whenever it needs to.

Before you use an array, you must set up space for it in the computer's memory by using the DIM command. For example, DIM A(3Ø) would set up a row of 30 empty spaces (numbered from 1 to 30) in a list called 'A'. Thirty numbers can now be stored in list 'A' — one in each space. If you wanted to store thirty names or words in the list instead of numbers you would have to call the list A$(3Ø). For the Spectrum, you would call the list A$(3Ø,12). The figure 12, in this example, states that none of the words in the list are longer than 12 letters.

To store a particular word or number in one of the spaces in the array you must give the computer the name of the array and the number of the space where the item is to be stored. For example:

```
LET A$(1Ø)="FRED"
```

will tell the computer to store the word FRED in list A$ at space 10. Or:

```
LET A(8)=13
```

will make it store the number 13 in list A at space 8.

The computer can also be made to 'read' a list of things given to it in a DATA line and then to store them in a particular array. (This is used in the 'Adventure' program.) For example, in the program lines shown below, the counter K tells the computer to read the first number from DATA and store it in the first space in array X. Then to read the second number and store it in the second space, and so on until all five numbers have been read and stored:

```
1Ø FOR K=1 TO 5:READ
   X(K):NEXT K
2Ø DATA 7,3,6,2,9
```

About ASCII Codes

Each of the keys on your keyboard has it own code number. (If a key has more than one use it will have a code for each use.) These numbers are called the **ASCII codes**, and they are the same for most home micros. The ASCII codes for the keys for 0 to 9, for example, are 48 to 57.

ASCII codes can be used in a program to make the computer check the keyboard to see if a particular key has been pressed.

For example, in the 'Intelligence' program, lines 520–590 are used to check if the user has correctly typed in a Yes (Y) or No (N) reply when asked to:

Line 520: Checks the keyboard to see if a key has been pressed and sets G$ to store the letter of the key pressed. If no key is pressed then G$="" (nothing) and the computer is told to stay where it is until a key is pressed.

Line 530: Sets the variable G to equal the ASCII code of the key pressed. If the code is more than 96 this means that the key is a lower-case (small) letter. (The ASCII codes for capital letters start at 65, and lower-case letters start at 97.) The computer is looking for a capital letter (a Y or N) so it is told to convert the code in G to a capital-letter code by subtracting 32. (The code for lower-case n, for example, is 110, so 110−32=78, which is the code for capital N.)

Line 540: The new code in G is now checked. If it is not 78 (for N) or 89 (for Y) it is not the reply the computer needs, so it goes back to line 520 to wait for the correct key to be pressed.

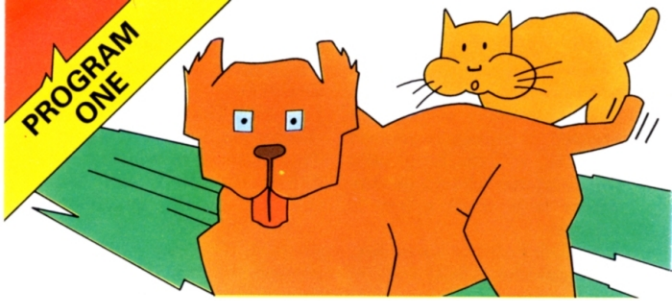
Line 550: If it is the code for Y or N the computer takes the letter stored in G$, which is now equal to the code in G, and prints it on the screen so that you can see what you typed in for the first time. (If you had typed in the wrong letter nothing would have appeared on the screen.)

Line 560: The computer now checks the keyboard again. This time it is waiting for either the RETURN key (ENTER on the Spectrum) or the DELETE key to be pressed. The key that you press is stored in J$ — if you do not press a key the computer will wait.

Line 570: The code of the key in J$ is then stored in J. If this code is not 13 (for RETURN or ENTER), and not 127 (for DELETE) the computer waits again. (Note: for the Spectrum the DELETE code is 12, for the Commodore it is 20.)

Line 580: If the code is 13, the computer knows that you are happy with the reply you have given and it carries on with the rest of the program.

Line 590: Otherwise (if the code is 127), it has been told that you wish to change your mind, so it rubs out the Y or N that it printed on the screen. It does this by moving left one place on the screen, and printing a blank space on top of the letter. CHR$(8) is the command to move left. Then it goes back to line 520 to get your new reply.



An Intelligence Program

A great deal of research is currently being carried out to design the 'fifth generation' of computers. One of the most exciting parts of this research is the work being done on Artificial Intelligence, or A.I. Whilst science fiction is full of stories of super-intelligent computers, the machines that we use in the real world are really high-speed idiots, blindly obeying the instructions they have been given.

Many A.I. researchers think that the only way to develop an intelligent computer is to give it an 'intelligent' program — one which allows the computer to think for itself, rather than simply accepting the information that is fed into it.

The program in this chapter is a very simple version of a 'learning' program. What the computer is learning about here is different kinds of animals, but this type of program could also be used with other subjects (see page 17 for some suggestions.)

It is important to remember, however, that this program does not enable the computer to *understand* the information it 'learns'. All it does is to use what it has been told in 'parrot-fashion'. If you tell it that goldfish live in trees it will accept this 'fact' without question. In computer terms this is called GIGO — garbage in, garbage out!

Using the Program

When the program is run, it will ask whether you have a file of information to use — for now, type N for 'No'. Now think of an animal, such as a dog. The computer will ask 'IS IT DOMESTIC?' — as the answer is 'Yes' for a dog, type in Y.

At this stage, the computer only knows one domestic animal, so it will print 'I THINK IT IS A CAT. AM I RIGHT?'. As you chose a dog, your reply will be N. You will then be asked for the correct animal's name, and for a question to help the computer distinguish that animal from the other animals it knows.

This information is 'remembered' by the computer, and each time you choose a new animal the computer will add it to its list. When you want to finish playing the program, type N to the question about another go. The computer will then ask whether you want it to save what it has learnt, to use again another time.

If you answer N, the program will stop, and all the names and questions you have put into the program will be wiped out. If you answer Y, you will be asked to choose a name for the file. The name must not be more than six letters long.

When you have typed in the name (and pressed the RETURN or ENTER key) the computer will save the file of information on tape. This file can now be used the next time you run the program by answering Y to the question 'DO YOU HAVE A FILE?'. (See page 6 for more details about saving and loading files and programs.)

How it Works

The program uses four arrays (or lists) to store information that it has 'learnt':

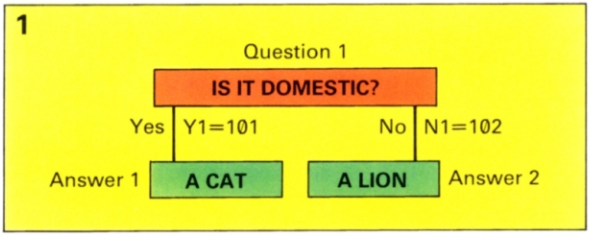
Q$ — contains a list of the questions which it can ask to identify each animal.

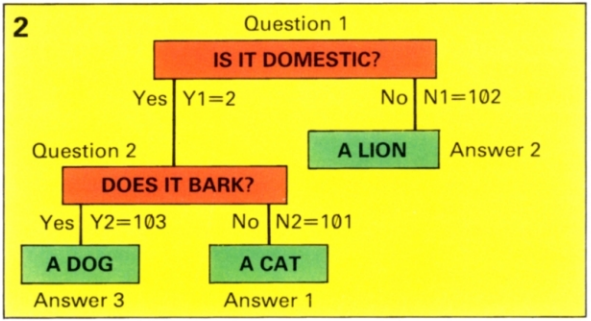
Y — gives it the number of the next question to ask if the reply to the previous question is 'Yes'.

N — gives it the question number if the reply is 'No'.

A$ — contains the list of animals' names.

Diagram 1 (on the next page) shows the situation when the program starts. There is only one question stored in Q$ to begin with — Q$(1) 'IS IT DOMESTIC?'; and two animals stored in A$ — one that is domestic in A$(1) 'A CAT', and one that isn't in A$(2) 'A LION'. If you answer Y to Question 1,





the program will go to the number stored in Y(1), which will be the number 101. (A number stored in the Y or in the N arrays which is less than 100 will be the number of the next question the computer should ask after a Yes or No answer. A number over 100 will be the number of the name of the animal which the computer should print out on the screen.) The number 101 in Y(1) tells the computer to print the name of the animal in A$(1) — the cat.

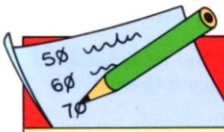
However, if you are thinking of another animal, such as a dog, you will tell the computer that it is wrong and type in the correct answer. This name will be stored in the next space in array A$, which is A$(3). You will then type in a question which will identify a cat from a dog, such as 'DOES IT BARK' — this will be stored in Q$(2).

As the answer to Question 2 is 'Yes' for a dog, the number or 'pointer' in Y(2) will be set to 103 — to print out the name 'DOG' stored in A$(3). But for a cat, the answer to Question 2 is 'No', so the number in N(2) will be set to 101 — to print 'CAT'.

Finally, the number stored in Y(1) — for a 'Yes' answer to Question 1 — is changed so that it will now send the computer to the next question it has learnt, Q$(2). The result of all this is shown in Diagram 2.

As you continue to play the program, more names and more questions will be added to the arrays A$ and Q$, and the Yes/No pointers will be set accordingly.

For example, if the next animal you think of after a dog is not domestic, i.e. a zebra, the name 'ZEBRA' will be stored in A$(4). The question for zebra will be stored in Q$(3). The number stored in N(1) (for a 'No' answer to 'IS IT DOMESTIC?') will be changed from 102 (to print out 'LION') to 3, to point to Question 3, and the Yes pointer for Question 3 is set to 104 to print out A$(4) — 'ZEBRA'. A 'No' answer to Question 3 leads to the number stored in N(3), which is now 102, to printout 'LION'.



Line Notes

10 ..... Program title (to save or load program).

50 ..... Set up space for arrays (up to 100 animals and questions).

60–70 ..... Ask if you have a file to use; go to subroutine to check reply. If Yes, skip next two lines.

80–90 ..... (If No) set up one question, the array number (pointer) for its Yes/No answer, and the names of the first two animals; then skip to next section.

More next page. . .

100–170 ..... Otherwise, ask for file name, open file and read in information.
180 ..... Clear the screen to start.
190 ..... Start with question 1.
200–210 ..... Print question and go to subroutine to check reply.
220–230 ..... Follow pointer to next stage, depending on Yes or No reply.
240 ..... If pointer is less than 100 it is the next question, go to previous instruction to print it out.
250 ..... Store last Yes/No reply for later use.
260–270 ..... Print computer's guess, ask if correct.
280 ..... Get reply, if Yes, animal is already known so skip next lot of instructions.
290–300 ..... If No, ask for correct animal and add to end of list.
310–320 ..... Ask for new question and add to list.
330–340 ..... Depending on reply, change last question's Yes or No pointer to lead to new question (see note for line 250).
350–360 ..... Ask for answer to new question for new animal; check reply.
370 ..... If Yes, then new Y pointer leads to new animal and N pointer leads to old animal.
380 ..... If No, then vice-versa.
390 ..... Give option to check new animal and new question typed in correctly, if not, go back and ask for correct name and question.
400 ..... Ask if you want to carry on.
410 ..... Check reply, if Yes remember new information and start again.
420 ..... Otherwise, ask whether to save file.
430 ..... Check reply, if No, end program.
440–500 ..... Otherwise, ask for name of file, open file and save information on tape.
510 ..... Clear screen and end program.

SUBROUTINE for Y/N Reply:

520–590 ..... Check that the reply has been entered and that it is either a Y or N (see pages 10–11 for more details).

PROGRAM VARIATIONS

You could also use this program to learn about towns by changing lines 80 and 90 to:

```
80 LET NQ=1: LET Q$(1)="IS IT IN SCOT
   LAND":LET Y(1)=101:LET N(1)=102
90 LET NA=2:LET A$(1)="EDINBURGH":LET
   A$(2)="LONDON"
```

Or, you could teach it about your friends, by changing the same lines to:

```
80 LET NQ=1: LET Q$(1)="IS IT A GIRL":LET
   Y(1)=101:LET N(1)=102
90 LET NA=2:LET A$(1)="SUSAN":LET
   A$(2)="JOHN"
```

The Program

Main program listing (For BBC/ELECTRON)

```
 10 REM "ANIMALS"
▲50 DIM Q$(100),Y(100),N(100),A$(100)
■60 CLS:PRINT:PRINT "DO YOU HAVE A
   FILE TO USE ? ";
 70 GOSUB 520:IF G$="Y" THEN GOTO 100
 80 LET NQ=1:LET Q$(1)="IS IT DOMEST
   IC":LET Y(1)=101:LET N(1)=102
```

More next page. . .

```
  90 LET NA=2:LET A$(1)="A CAT":LET
     A$(2)="A LION":GOTO 180
▲ 100 PRINT "WHAT IS IT CALLED ";
     :INPUT F$
  110 PRINT:PRINT "LOADING FILE"
  120 PRINT "PUT TAPE IN AND PRESS
     PLAY"
● 130 LET F=OPENIN(F$):INPUT ≠F,NQ
     :INPUT ≠F,NA
● 140 FOR K=1 TO NQ:INPUT ≠F,Q$(K)
     :INPUT ≠F,Y(K):INPUT ≠F,N(K)
     :NEXT K
▲ 150 FOR K=1 TO NA
● 160 INPUT ≠F,A$(K)
● 170 NEXT K:CLOSE ≠F
■ 180 CLS
  190 LET Q=1
  200 PRINT:PRINT Q$(Q):PRINT " ? ";
  210 GOSUB 520
  220 IF G$="Y" THEN LET N=Y(Q)
  230 IF G$="N" THEN LET N=N(Q)
  240 IF N<100 THEN LET Q=N:GOTO 200
  250 LET T$=G$
  260 PRINT:PRINT "I THINK IT IS ";
     A$(N-100)
  270 PRINT:PRINT "AM I RIGHT ? ";
  280 GOSUB 520:IF G$="Y" THEN
     GOTO 400
  290 PRINT:PRINT "WHAT IS THE RIGHT
     ANSWER .."
▲ 300 LET NA=NA+1:INPUT A$(NA)
  310 PRINT:PRINT "WHAT QUESTION
     SHOULD I ASK TO"
▲ 320 PRINT "TELL ";A$(N-100):PRINT
     "FROM ";A$(NA):LET NQ=NQ+1
     :INPUT Q$(NQ)
  330 IF T$="Y" THEN LET Y(Q)=NQ
```

≠ symbol in programs equals # on your keyboard !

```
  340 IF T$="N" THEN LET N(Q)=NQ
  350 PRINT:PRINT "IS THE ANSWER 'Y'
      OR 'N' FOR"
  360 PRINT A$(NA):PRINT " ? ";
      :GOSUB 520
  370 IF G$="Y" THEN LET Y(NQ)=100+NA
      :LET N(NQ)=N
  380 IF G$="N" THEN LET N(NQ)=100+NA
      :LET Y(NQ)=N
  390 PRINT "O.K. ? ";:GOSUB 520:IF
      G$="N" THEN LET NQ=NQ-1:LET
      NA=NA-1:GOTO 290
  400 PRINT:PRINT "DO YOU WANT ANOTHER
      GO ? ";
  410 GOSUB 520:IF G$="Y" THEN
      GOTO 180
  420 PRINT:PRINT "DO YOU WANT TO SAVE
      THIS FILE ? ";
  430 GOSUB 520:IF G$="N" THEN
      GOTO 510
▲ 440 PRINT "WHAT SHOULD I CALL IT ";
      :INPUT F$
  450 PRINT:PRINT "SAVING FILE"
  460 PRINT "PUT TAPE IN"
● 470 LET F=OPENOUT(F$):PRINT #F,NQ
      :PRINT #F,NA
● 480 FOR K=1 TO NQ:PRINT #F,Q$(K)
      :PRINT #F,Y(K):PRINT #F,N(K)
      :NEXT K
● 490 FOR K=1 TO NA:PRINT #F,A$(K)
      :NEXT K
● 500 CLOSE #F
■ 510 CLS:PRINT:PRINT "END OF PROGRAM"
      :PRINT:STOP
● 520 LET G$=INKEY$(0):IF G$=""
      THEN GOTO 520
```

More next page. . .

```
▲ 530 LET G=ASC(G$):IF G>96 THEN
      LET G=G-32
  540 IF G<>78 AND G<>89 THEN GOTO 520
  550 LET G$=CHR$(G):PRINT G$;
● 560 LET J$=INKEY$(0):IF J$="" THEN
      GOTO 560
● 570 LET J=ASC(J$):IF J<>13 AND
      J<>127 THEN GOTO 560
  580 IF J=13 THEN PRINT:RETURN
■ 590 PRINT CHR$(8);" ";CHR$(8);
      :GOTO 520
```

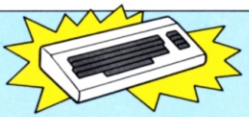
SPECTRUM ALTERATIONS

```
50 DIM Q$(100,26):DIM Y(100):DIM
   N(100):DIM A$(100,15)
130 LOAD F$ DATA Q$()
140 LOAD F$ DATA Y()
150 LOAD F$ DATA N()
160 LOAD F$ DATA A$()
170 LET NQ=Y(100):LET NA=N(100)
470 LET Y(100)=NQ:LET N(100)=NA
480 SAVE F$ DATA Q$()
490 SAVE F$ DATA Y():SAVE F$
    DATA N()
500 SAVE F$ DATA A$()
520 LET G$=INKEY$:IF G$="" THEN
    GOTO 520
530 LET G=CODE(G$):IF G>96
    THEN LET G=G-32
560 LET J$=INKEY$:IF J$="" THEN
    GOTO 560
570 LET J=CODE(J$):IF J<>13 AND
    J<>12 THEN GOTO 560
```

```
Replace INPUT X by INPUT X:PRINT X
in lines 100 300 320 and 440
```

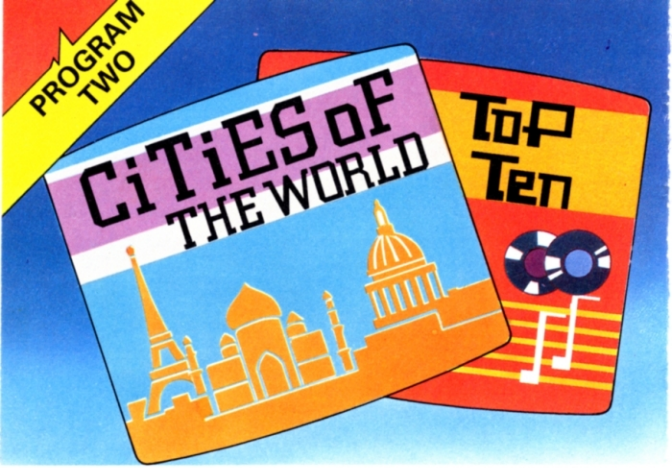
EXAMPLE:
100 PRINT "WHAT IS IT CALLED ";:INPUT F$:
PRINT F$

```
COMMODORE
ALTERATIONS
==========
```



```
130 OPEN 1,1,0,F$:INPUT#1,NQ
    :INPUT#1,NA
140 FOR K=1 TO NQ:INPUT#1,Q$(K)
    :INPUT#1,Y(K):INPUT#1,N(K)
    :NEXT K
160 INPUT#1,A$(K)
170 NEXT K:CLOSE 1
470 OPEN 1,1,1,F$:PRINT#1,NQ
    :PRINT#1,NA
480 FOR K=1 TO NQ:PRINT#1,Q$(K)
    :PRINT#1,Y(K):PRINT#1,N(K)
    :NEXT K
490 FOR K=1 TO NA:PRINT#1,A$(K)
    :NEXT K
500 CLOSE 1
520 GET G$:IF G$="" THEN GOTO 520
560 GET J$:IF J$="" THEN GOTO 560
570 LET J=ASC(J$):IF J<>13 AND J<>20
    THEN GOTO 560
590 PRINT CHR$(157);" ";CHR$(157);
    :GOTO 520

Replace CLS by PRINT CHR$(147) in
lines 60, 180 and 510
```



Quiz Writer

This program allows you to make up quizzes on any subject you want, from 'Cities of the World' to 'Top Ten Hits', and store them on tape. The quizzes can then be used as revision aids, or just for fun — see if you can baffle your friends!

Using the Program

When the program starts, you are asked whether you want to write a quiz (W), load one from tape (L), save one (S), try one (T), or end the program (E) — as you haven't got any quizzes yet, type W. You now type in the number of questions you want to set — this can be any number up to 30.

For each question you set, you must type in both the question and the right answer. You will then be asked whether you have typed them in correctly — if not, type N and you will be able to retype them.

After the last question has been entered, the program will go back to the beginning. You can now choose to

save the quiz. The computer will ask for a file name for the quiz (remember — the name should not be more than six letters). It will then display a 'SAVING' message and will proceed to save the quiz on tape.

When the saving is completed, the program returns to the list of options at the beginning — this time, try typing L, and then the name of the quiz you have just saved. Now rewind the tape and load the quiz back into the computer. You could now type T to try out your quiz.

You choose how many questions you want to try from the total number in the quiz. Each question will be chosen at random from those that you typed in. You will be given a (small) clue in the form of a line of dashes, showing how long the right answer is. You get two tries at each question, after which the computer will print the right answer. At the end of the quiz the computer will show you your score. It will then ask whether you would like a second go at those you got wrong — if Yes, type Y.

Finally, you will be asked whether you want to play this quiz again. If you type Y, you will be asked how many questions. If you type N, you will be returned to the beginning of the program. If you now type E, the program will stop, and the quiz you loaded in will be wiped out of the computer.

How it Works

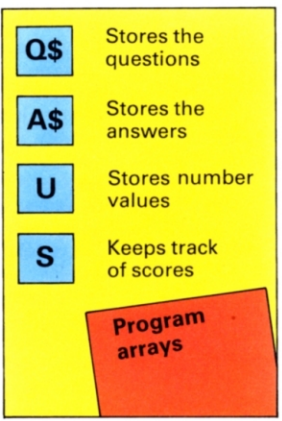
This program uses four arrays:

Q$ — stores the questions.

A$ — stores the answers.

U — stores a number value for each question as it is used: Ø = question not used yet; 1 = right first time; 2 = right on second try; 3 = wrong — the computer gave the right answer.

S — keeps track of how many 1's, 2's and 3's you have scored altogether.



The U array has two purposes. Firstly, it makes sure that the questions selected for each test are all different. A question number is chosen at random and the U array is checked — if the question has a value that is anything other than Ø it will have already been used, so the computer chooses another question.

Secondly, if you ask to re-try the questions that you got wrong, the computer looks at the U array for each question in turn — if the value for a question is 3, then that question is printed out again.

Line Notes

10	Program title, for saving and loading program lines.
50	Set up space for arrays (up to 30 questions and answers).
60–70	Clear screen, and ask which option.
80–110	Go to subroutine (5) to check reply; depending on reply, skip to correct instructions for option chosen.
120	If reply anything other than W (and not one of other options) go back and ask again.
130	(If choose write option) ask how many questions there will be.
140–160	For each question, print its number, ask for the question and its answer.
170	Go to subroutine (2) to check that the answer has been typed in capital letters; if not, change the lower-case (small) letters to capital letters; then store the answer in A$.

180 ..... Ask if question and answer are correct, and go to subroutine (5) to check reply.
190 ..... If No, go back and ask for them again.
200 ..... Repeat until all questions and answers have been typed in; go back to beginning of program.
210–240 ..... (If choose save option) ask what to call the quiz; print saving message; open file; save information on tape and go back to beginning of program.
250–280 ..... (If choose load option) ask for name of

file; print loading message; open file; load in information and go back to beginning of program.
290–300 ..... (If choose try option) set array U to Ø; clear screen; ask how many questions to set.
310 ..... Check if number asked for is within range of number of questions in quiz, if not, go back and ask again.
320 ..... Set score array to Ø; set question counter to Ø.
330 ..... Set question counter to next question.
340 ..... Choose a question number at random; if question already used, go back and choose again.

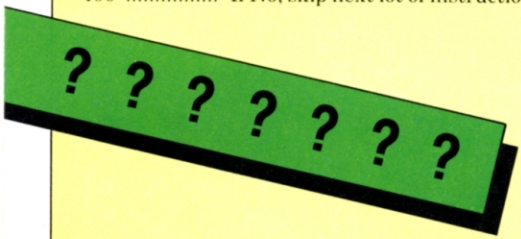
More next page. . .

350 ..... Go to subroutine (1) to ask question and check answer; until all questions completed go to subroutine (4) to wait for RETURN or ENTER key to be pressed and go back to ask next question.

360 ..... Go to subroutine (3) to print score.

370–390 ..... Ask whether to repeat the questions you got wrong.

400 ..... If No, skip next lot of instructions.



410 ..... Otherwise, clear previous scores and set question counter to Ø.

420 ..... For each question answered, check if did not get correct answer last time, if so, go to subroutine (1) and ask question again.

430 ..... Repeat until come to end of questions; go to subroutine (3) for new score.

440–470 ..... Ask if want another go; if Yes, go back to start of this section; otherwise, go back to beginning of the program.

480 ..... (If choose end option) clear screen and stop.

SUBROUTINE (1) to Ask Question and Get Answer:

490–520 ..... Print the question number; set number of attempts at this question to Ø; print out the question and a line of dashes.

530 ..... Add 1 to the number of attempts; get answer and go to subroutine (2) for capital letter check.

540 ..... If answer correct, say so.

550 ..... If wrong on first try, say so and go back for a second try.

560 ..... If wrong on second try, increase attempts' counter from 2 to 3 and print correct answer.

570 ..... Store the number of tries in array U and add 1 to the appropriate score (1st try, 2nd try, or wrong), then return to main program.

SUBROUTINE (2) to Convert to Capital Letters:

580 ..... Clear a temporary storage space.

590 ..... Work out the ASCII code for each letter of each word; if it is a lower-case letter convert it to a capital letter (see page 10 for details about ASCII codes).

600–610 ..... Add each conversion to the temporary store; when all the letters have been checked, go back to main program.

SUBROUTINE (3) to Print Score:

620 ..... Print message and wait for RETURN or ENTER key to be pressed.

630–690 ..... Print out score; return to main program.

SUBROUTINE (4) to Wait for RETURN or ENTER Key:

700 ..... Print message.

710 ..... Check keyboard; if no key pressed keep checking.

720 ..... If key pressed is not RETURN or ENTER (not ASCII code 13), go back and check keyboard again.

730 ..... Otherwise, return to main program.

More next page. . .

SUBROUTINE (5) to Check Single Letter Reply:

740	Check reply key has been pressed, if not, keep checking.
750–760	If reply is in lower case, convert to capital letter and print on screen.
770	Check keyboard, if no key pressed, keep checking.
780	If key pressed is not RETURN/ENTER (code 13) or DELETE (code 127), check keyboard again.
790	If RETURN/ENTER key pressed, reply was correct, go back to main program.
800	If DELETE key pressed, rub out reply entered and go back for new reply. (Allows users to change their mind!)

The Program

Main program listing (For BBC/ELECTRON)

```
 10 REM "QUIZZES"
▲50 DIM Q$(30),A$(30),U(30),S(3)
■60 CLS:PRINT:PRINT "DO YOU WANT TO
   ...":PRINT "WRITE A QUIZ (W) ";
 70 PRINT "LOAD ONE (L)":PRINT "SAVE
   ONE (S) TRY ONE (T)":PRINT "OR
   END (E) ? ";
 80 GOSUB 740:IF G$="L" THEN GOTO 250
 90 IF G$="S" THEN GOTO 210
 100 IF G$="T" THEN GOTO 290
 110 IF G$="E" THEN GOTO 480
 120 IF G$<>"W" THEN GOTO 60
▲130 PRINT "HOW MANY QUESTIONS WILL
    THERE BE ";:INPUT NQ
```

```
■ 140 FOR K=1 TO NQ:CLS:PRINT:PRINT
      "QUESTION ";K
▲ 150 PRINT:PRINT "TYPE IN THE
      QUESTION...":INPUT Q$(K)
▲ 160 PRINT "AND NOW THE CORRECT
      ANSWER...":INPUT R$
  170 GOSUB 580:LET A$(K)=T$
  180 PRINT "CORRECT (Y/N) ? ";
      :GOSUB 740:IF G$<>"Y" AND
      G$<>"N" THEN GOTO 180
  190 IF G$="N" THEN GOTO 150
  200 NEXT K:GOTO 60
● 210 CLS:PRINT:PRINT "WHAT SHALL
      I CALL THE QUIZ ";:INPUT F$
  220 PRINT:PRINT "SAVING THE QUIZ"
      :PRINT:PRINT "PUT TAPE IN"
● 230 LET F=OPENOUT(F$):PRINT #F,NQ
● 240 FOR K=1 TO NQ:PRINT #F,Q$(K)
      :PRINT #F,A$(K):NEXT K:CLOSE #F
      :GOTO 60
▲ 250 PRINT "WHAT IS THE QUIZ CALLED ";
      :INPUT F$
  260 PRINT:PRINT "PUT TAPE IN AND
      PRESS PLAY"
● 270 LET F=OPENIN(F$):INPUT #F,NQ
● 280 FOR K=1 TO NQ:INPUT #F,Q$(K)
      :INPUT #F,A$(K):NEXT K:CLOSE #F
      :GOTO 60
  290 FOR K=1 TO NQ:LET U(K)=0:NEXT K
■ 300 CLS:PRINT:PRINT "HOW MANY
      QUESTIONS (1 - ";NQ;") ";
▲ 310 INPUT N:IF N<1 OR N>NQ THEN
      GOTO 300
  320 LET S(1)=0:LET S(2)=0:LET S(3)=0
      :LET Q=0
```

More next page. . .

```
  330 LET Q=Q+1
▲ 340 LET P=INT(NQ*RND(1)+1):IF U(P)>0
      THEN GOTO 340
  350 GOSUB 490:IF Q<N THEN GOSUB 700
      :GOTO 330
  360 GOSUB 620
  370 PRINT:PRINT "DO YOU WANT TO
      REPEAT THOSE THAT"
  380 PRINT "YOU GOT WRONG FIRST TIME
      (Y/N) ? ";
  390 GOSUB 740:IF G$<>"Y" AND G$<>"N"
      THEN GOTO 370
  400 IF G$="N" THEN GOTO 440
  410 LET S(1)=0:LET S(2)=0:LET S(3)=0
      :LET Q=0
  420 FOR P=1 TO NQ:IF U(P)=3 THEN LET
      Q=Q+1:GOSUB 490
  430 NEXT P:GOSUB 620
■ 440 CLS:PRINT:PRINT "ANOTHER GO FROM
      THIS QUIZ (Y/N) ? ";
  450 GOSUB 740:IF G$<>"Y" AND G$<>"N"
      THEN GOTO 440
  460 IF G$="Y" THEN GOTO 290
  470 GOTO 60
■ 480 CLS:PRINT:PRINT "END OF PROGRAM"
      :STOP
■ 490 CLS:PRINT:PRINT "QUESTION ";Q
  500 PRINT:PRINT Q$(P):LET A=0
▲ 510 PRINT " ";:FOR K=1 TO LEN(A$(P))
  520 PRINT "-";:NEXT K:PRINT
▲ 530 LET A=A+1:INPUT R$:GOSUB 580
▲ 540 IF T$=A$(P) THEN PRINT "CORRECT
      !!":GOTO 570
  550 IF A=1 THEN PRINT "WRONG - TRY
      AGAIN..":GOTO 530
  560 LET A=3:PRINT "NO, THE RIGHT
      ANSWER IS ":PRINT A$(P)
```

```
 570 LET U(P)=A:LET S(A)=S(A)+1
     :RETURN
 580 LET T$="":FOR R=1 TO LEN(R$)
▲590 LET C=ASC(MID$(R$,R,1)):IF C>96
     THEN LET C=C-32
 600 LET T$=T$+CHR$(C)
 610 NEXT R:RETURN
 620 PRINT:PRINT "***END OF QUIZ***"
     :GOSUB 700
■630 CLS:PRINT:PRINT "YOUR RESULTS
     WERE ..."
 640 PRINT:PRINT "RIGHT FIRST TIME
     ...";S(1)
 650 PRINT:PRINT "RIGHT SECOND TRY
     ...";S(2)
 660 PRINT:PRINT "WRONG ..........
     ..";S(3)
 670 PRINT:PRINT "================
     ======="
 680 PRINT:PRINT "TOTAL ..........
     ..";Q
 690 PRINT:PRINT:RETURN
 700 PRINT:PRINT "PRESS RETURN TO
     CONTINUE .."
●710 LET G$=INKEY$(0):IF G$="" THEN
     GOTO 710
▲720 LET G=ASC(G$):IF G<>13 THEN
     GOTO 710
 730 RETURN
●740 LET G$=INKEY$(0):IF G$=""
     THEN 740
▲750 LET G=ASC(G$):IF G>96 THEN
     LET G=G-32
 760 LET G$=CHR$(G):PRINT G$;
```

More next page. . .

```
● 770 LET J$=INKEY$(0):IF J$="" THEN
     GOTO 770
● 780 LET J=ASC(J$):IF J<>13 AND
     J<>127 THEN GOTO 770
  790 IF J=13 THEN PRINT:RETURN
■ 800 PRINT CHR$(8);" ";CHR$(8);
     :GOTO 740
```

SPECTRUM ALTERATIONS
==========

```
 50 DIM Q$(30,30):DIM A$(30,20):DIM
    U(30):DIM L(30):DIM S(3)
160 PRINT:PRINT "AND NOW THE CORRECT
    ANSWER...":INPUT R$:PRINT R$:LET
    L(K)=LEN (R$)
230 LET L(30)=NQ
240 SAVE F$ DATA Q$():SAVE F$ DATA
    L():SAVE F$ DATA A$():GOTO 60
270 LOAD F$ DATA Q$():LOAD F$
    DATA L()
280 LOAD F$ DATA A$():LET NQ=L(30)
    :GOTO 60
340 LET P=INT(NQ*RND+1):IF U(P)>0
    THEN GOTO 340
510 PRINT " ";:FOR K=1 TO L(P)
530 LET A=A+1:INPUT R$:PRINT " ";R$
    :GOSUB 580
540 IF T$=A$(P)(TO L(P)) THEN PRINT
    "CORRECT !!":GOTO 570
590 LET C=CODE R$(R):IF C>96 THEN
    LET C=C-32
710 LET G$=INKEY$:IF G$="" THEN
    GOTO 710
720 LET G=CODE(G$):IF G<>13 THEN
    GOTO 730
```

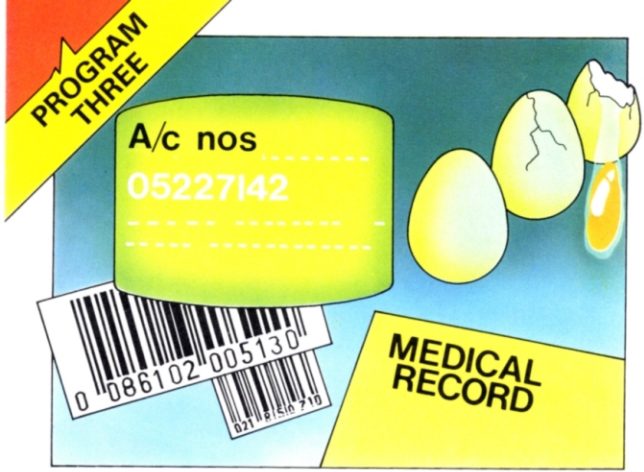
```
740 LET G$=INKEY$:IF G$="" THEN
    GOTO 740
750 LET G=CODE(G$):IF G>96 THEN
    LET G=G-32
770 LET J$=INKEY$:IF J$="" THEN
    GOTO 770
780 LET J=CODE(J$):IF J<>13 AND
    J<>12 THEN GOTO 770
```

Replace INPUT X by INPUT X:PRINT X in lines 130 150 210 250 and 310

COMMODORE
ALTERATIONS
==========

```
230 OPEN 1,1,1,F$:PRINT#1,NQ
240 FOR K=1 TO NQ:PRINT#1,Q$(K)
    :PRINT#1,A$(K):NEXT K:CLOSE 1
    :GOTO 60
270 OPEN 1,1,0,F$:INPUT#1,NQ
280 FOR K=1 TO NQ:INPUT#1,Q$(K)
    :INPUT#1,A$(K):NEXT K:CLOSE 1
    :GOTO 60
710 GET G$:IF G$="" THEN GOTO 710
740 GET G$:IF G$="" THEN GOTO 740
770 GET J$:IF J$="" THEN GOTO 770
780 LET J=ASC(J$):IF J<>13 AND J<>20
    THEN GOTO 770
800 PRINT CHR$(157);" ";CHR$(157);
    :GOTO 740
```

Replace CLS by PRINT CHR$(147) in lines 60 140, 210, 300, 440, 480 490 and 630



Secret Files

Computers are used to store all sorts of information, ranging from the price of eggs in the local supermarket to the amount of money you have in the bank or your medical record! It is the ability of the computer to search quickly through large amounts of information, selecting the items wanted and sorting them into any required order, which explains their importance in the business world.

This program illustrates the main features of a file-handling system. It allows you to add and remove information from a file, to search for a particular item or to look at the whole file, to sort it into a particular order, and to save the information on tape for later use.

In this example, we are storing names and telephone numbers, but any other information, or data, can be stored in the same way. One name and one telephone number together is known as a record.

Using the Program

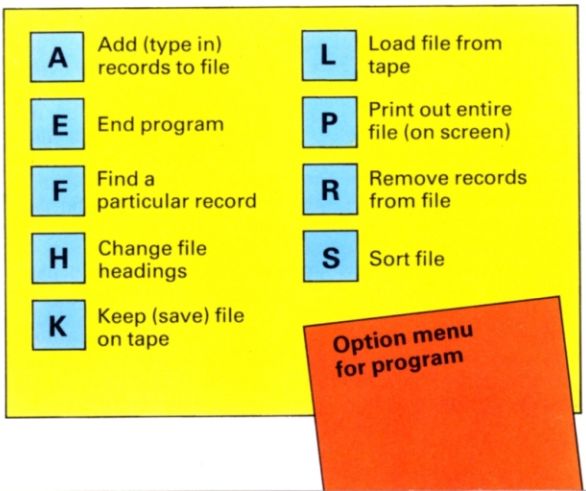
When the program starts, it will display a list of the different things that you can choose to do with your file — this list is known as a **menu.**

To choose a particular option from the menu, just press the correct letter key. If you press A, for example, you will be asked to type in the names and telephone numbers that you want to store in your file (when you have put some records in your file, you can also use this option to add extra records later).

To get back to the menu again at any time, simply press the RETURN key or ENTER key twice.

Remember that, if you press E the program will end and, if you have not saved the information in your file on tape, that information will be wiped out of the computer's memory.

Option K allows you to keep the information on tape. You will be asked for a name with which to identify that particular file (such as PHONES) and you will be asked for a password. You can choose whatever password you like, but make sure you can remember it! You will need to give this password in order to re-enter the file when you load it back



into the computer again, using option L. (The secret password will help to protect your file from being used by anyone else.)

Once you have put some records into your file, you can use the other options in the menu to change or update those records. Option S, for example, will sort all your records into a particular order. With PHONES, you can choose to sort the records alphabetically by Name (1) or by Number (2).

Finally, option H allows you to change the headings of the file so that you can use the same program to write other files containing different information. You could change the headings to 'Names' and 'Birthdays', for example, or keep a file of all the books you own by 'Title' and by 'Author'.

Warning

With some makes of computer you may find that if the information you type into the file contains commas or double quotation marks, any words after the comma or quotation marks may not appear on the screen when you press P to show the file. The best thing is to avoid using these punctuation marks in your records.

How it Works

The method used to search the file allows you to find a record even if you don't remember the exact spelling of a name. It does this by using LEN to find the length of the word that you type in, and then using LEFT$ to check your word against the same number of letters in each name in the file. (See page 9 for more details.)

For example, if you type in RA in answer to the question 'NAME TO LOOK FOR?', the command L=LEN(X$) will set L to the value 2. The next instruction:

IF X$=LEFT$ (. . . . . ,L) THEN. . .

will then check if the first two letters of each name in the file are RA. This will find all names such as RAY and RAI (and also RACHEL, RAYMOND, and so on).

The method used to sort the records into order is called a 'Bubble Sort'. Each name in the file is compared with all the ones after it. If they are in the wrong alphabetical order they are swapped round — not forgetting to swap the phone numbers too!

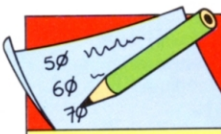
The most obvious way to swap two names that are stored in, say, A$ and B$, is to give the instruction:

LET A$=B$:LET B$=A$

However, this does not work, as the first command will store the name from B$ into A$ and will then forget the name that was originally in A$. The correct way to do the swap is to say:

LET T$=A$:LET A$=B$:LET B$=T$

T$ is used as a 'safe place' to store the name from A$, while the name from B$ is moved into A$. The name in T$ can then be placed in B$ to complete the swap.



Line Notes

10 ..... Program title, for saving and loading program lines.

50 ..... Set up space for arrays, 100 × 2 Records (R$) and 3 Headings (H$) — two for file headings and one for password.

60 ..... Set number of records to Ø; set file headings for two columns (Name and Number).

70–160 ..... Clear screen and print menu.

170 ..... Go to subroutine (3) to get reply (1 letter).

180–260 ..... Check reply against list of possible choices, and jump to correct part of program.

270 ..... If reply not one of the choices, go back and ask again.

To Add Records:

280–290 ..... Clear screen; open temporary store (X$) and ask for next Name (H$(1)) — if nothing entered, go back to menu.

More next page. . .

300 ..... Add 1 to number of records and store Name in record array with this record number. Ask for Number (H$(2)); store it, then go back to get next record.

To Find Records:

310 ..... Go to subroutine (1) to ask which heading to look for (Name or Number).

320 ..... Open temporary store; ask for Name (or Number) to look for — if nothing entered go back to menu.

330 ..... Set variable FF (record found) to Ø; store length of reply given in variable L (e.g. if reply is JOHN, L=4); set K to point to each record in turn.

340–350 ..... Check each record in file to find if first L characters match (e.g. first 4 characters are JOHN); if match found, print the Name and Number in that record, and set FF to 1 (record found).

360 ..... If no match found (so FF still Ø), say so.

370 ..... Go to subroutine (2) to wait for RETURN or ENTER key to be pressed before clearing screen to ask about another search.

To Change Headings:

380–390 ..... Ask for new column headings and store them, then go back to menu.

To Load File:

400–410 ..... Ask for file name; print loading message.

420 ..... Open file; load in headings and password.

430–450 ..... Load in number of records, then the records themselves; close file.

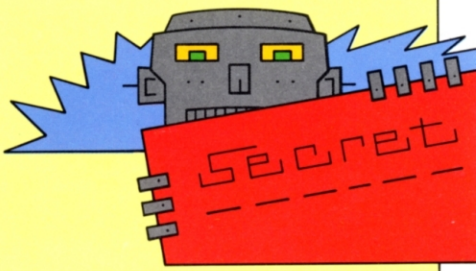
460 ..... Ask for password; if reply matches that stored in heading 3, go to menu.

470 ..... Otherwise, print message and ask for password again.

To Print File:

480 ..... Clear screen; print file headings.

490–500 ..... Print all the records on screen; go to subroutine (2) to check for RETURN or ENTER key before clearing the screen and returning to menu.



To Remove Records:

510 ..... Clear screen and set FF to 0.

520 ..... Open temporary store, ask for Name to be removed — if nothing entered, go back to menu.

530–540 ..... Check each record (as with option F), if it doesn't match, skip next instructions.

550 ..... Otherwise, print matching record; set FF to 1 to show record found.

560–570 ..... Ask whether to remove it from file; check reply, if anything other than Yes, then skip next lot of instructions.

580 ..... Otherwise, take a copy of the last record in the file and write it on top of this record; the last record is now no longer needed (as there are two copies of it) so 'lose' it by reducing the number of records by 1.

More next page. . .

590–600 ..... Now check next record; if no match found by end of file (FF still Ø), say so.

610 ..... Check for RETURN or ENTER key before clearing screen to ask about removing another record.

To Keep File:

620–650 ..... Ask for name of file; ask for password; print saving message.

660 ..... Open file, save headings and password.

670–680 ..... Save number of records, then records themselves; close file; go back to menu.

To Sort File:

690 ..... If less than 2 records cannot sort!

700 ..... Otherwise, go to subroutine (1) to ask which heading to sort by.

710–720 ..... Check each Name (or Number) with the next ones in the list in turn, if it is in the right order (alphabetically or numerically), then skip next lines.

730–740 ..... If it is in the wrong order, swap Names around, then swap the Numbers, too (or vice versa).

750 ..... Do the same for all the records in the file, then print out the sorted file.

To End Program:

760 ..... Clear screen; print message, then stop.

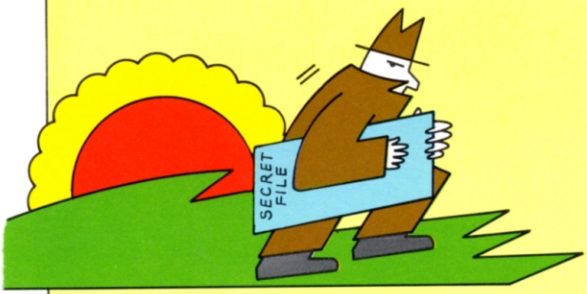
SUBROUTINE (1) to Ask Which Heading to Use:

770 ..... Print the two file headings (Name and Number).

780 ..... Check keyboard; if no key pressed, keep checking.

790 ..... If key pressed is not a 1 (ASCII code 49) or a 2 (ASCII code 50), then check keyboard again. (See page 10 for details about ASCII.)

800 ..... Otherwise, print key pressed; change ASCII code to the correct decimal number (1 or 2) by subtracting 48, and go back to main program.



SUBROUTINE (2) to Wait for RETURN/ENTER:

810 ..... Print message.
820 ..... Check keyboard; if no key pressed, keep checking.
830 ..... If key pressed is not RETURN or ENTER (not code 13), check again.
840 ..... Otherwise, return to main program.

SUBROUTINE (3) to Check Single Letter Reply:

850 ..... Check reply key has been pressed, if not, keep checking.
860–870 ..... If reply is in lower case (code is more than 96), convert to capital letter and print on screen.
880 ..... Check keyboard; if no key pressed, keep checking.
890 ..... If key pressed is not RETURN/ENTER (code 13) or DELETE (code 127), check keyboard again.

More next page. . .

900 ..... If RETURN/ENTER key pressed, reply was correct; go back to main program.
910 ..... If DELETE key pressed, rub out reply entered and go back for new reply.

The Program

Main program listing (For BBC/ELECTRON)

```
  10 REM "FILES"
▲ 50 DIM R$(100,2),H$(3)
  60 LET NR=0:LET H$(1)="NAME":LET
     H$(2)="NUMBER"
■ 70 CLS:PRINT:PRINT "OPTIONS ARE ..."
  80 PRINT "A - ADD RECORDS TO FILE"
  90 PRINT "E - END PROGRAM"
  100 PRINT "F - FIND GIVEN RECORD"
  110 PRINT "H - CHANGE HEADINGS"
  120 PRINT "K - KEEP FILE ON TAPE"
  130 PRINT "L - LOAD FILE FROM TAPE"
  140 PRINT "P - PRINT ENTIRE FILE"
  150 PRINT "R - REMOVE RECORDS FROM
      FILE"
  160 PRINT "S - SORT FILE"
  170 PRINT:PRINT "WHICH OPTION ? ";
      :GOSUB 850
  180 IF G$="A" THEN GOTO 280
  190 IF G$="E" THEN GOTO 760
  200 IF G$="F" THEN GOTO 310
  210 IF G$="H" THEN GOTO 380
  220 IF G$="K" THEN GOTO 620
  230 IF G$="L" THEN GOTO 400
  240 IF G$="P" THEN GOTO 480
```

```
  250 IF G$="R" THEN GOTO 510
  260 IF G$="S" THEN GOTO 690
  270 GOTO 70
■ 280 CLS:PRINT
▲ 290 LET X$="":PRINT:PRINT H$(1);
      :INPUT X$:IF X$="" THEN GOTO 70
▲ 300 LET NR=NR+1:LET R$(NR,1)=X$
      :PRINT H$(2);:INPUT R$(NR,2)
      :GOTO 290
  310 GOSUB 770
▲ 320 LET X$="":PRINT:PRINT H$(H);" TO
      LOOK FOR ";:INPUT X$:IF X$=""
      THEN GOTO 70
  330 LET FF=0:LET L=LEN(X$):FOR K=1
      TO NR
▲ 340 IF X$=LEFT$(R$(K,H),L) THEN
      PRINT R$(K,1),R$(K,2):LET FF=1
  350 NEXT K
  360 IF FF=0 THEN PRINT H$(H);" NOT
      IN FILE ..."
■ 370 GOSUB 810:CLS:GOTO 320
▲ 380 PRINT "HEADING 1 ";:INPUT H$(1)
      :PRINT "HEADING 2 ";:INPUT H$(2)
  390 GOTO 70
● 400 CLS:PRINT:PRINT "WHAT IS THE
      FILE CALLED ";:INPUT F$
  410 PRINT:PRINT "LOADING FILE":PRINT
      :PRINT "PUT TAPE IN AND PRESS
      PLAY"
● 420 LET F=OPENIN(F$):INPUT #F,H$(1)
      :INPUT #F,H$(2):INPUT #F,H$(3)
● 430 INPUT #F,NR:FOR K=1 TO NR
● 440 INPUT #F,R$(K,1):INPUT #F,
      R$(K,2)
● 450 NEXT K:CLOSE #F
```

More next page. . .

```
▲460 PRINT:PRINT "WHAT IS THE
     PASSWORD ";:INPUT P$:IF P$=H$(3)
     THEN GOTO 70
 470 PRINT "NO IT ISN'T !!":GOTO 460
■480 CLS:PRINT:PRINT H$(1),H$(2)
 490 FOR K=1 TO NR:PRINT R$(K,1),
     R$(K,2)
 500 NEXT K:GOSUB 810:GOTO 70
■510 CLS:PRINT:LET FF=0
▲520 LET X$="":PRINT H$(1);" TO
     REMOVE":INPUT X$:IF X$="" THEN
     GOTO 70
 530 LET L=LEN(X$):FOR K=1 TO NR
▲540 IF X$<>LEFT$(R$(K,1),L) THEN
     GOTO 590
 550 PRINT R$(K,1),R$(K,2):LET FF=1
 560 PRINT "REMOVE (Y/N) ? ";
 570 GOSUB 850:IF G$<>"Y" THEN
     GOTO 590
 580 LET R$(K,1)=R$(NR,1):LET
     R$(K,2)=R$(NR,2):LET NR=NR-1
 590 NEXT K
 600 IF FF=0 THEN PRINT H$(1);" NOT
     IN FILE"
 610 GOSUB 810:GOTO 510
●620 CLS:PRINT:PRINT "WHAT SHALL I
     CALL THE FILE ";:INPUT F$
▲630 PRINT:PRINT "WHAT IS THE
     PASSWORD ";:INPUT P$:LET
     H$(3)=P$
 640 PRINT:PRINT "KEEPING FILE"
 650 PRINT "PUT TAPE IN"
●660 LET F=OPENOUT(F$):PRINT #F,
     H$(1):PRINT #F,H$(2):PRINT #F,
     H$(3)
●670 PRINT #F,NR:FOR K=1 TO NR:PRINT
     #F,R$(K,1):PRINT #F,R$(K,2)
●680 NEXT K:CLOSE #F:GOTO 70
```

```
690 IF NR<2 THEN GOTO 480
700 GOSUB 770
710 FOR K=1 TO NR-1:FOR J=K+1 TO NR
720 IF R$(K,H)<=R$(J,H) THEN
    GOTO 750
730 LET T$=R$(K,1):LET R$(K,1)=
    R$(J,1):LET R$(J,1)=T$
740 LET T$=R$(K,2):LET R$(K,2)=
    R$(J,2):LET R$(J,2)=T$
750 NEXT J:NEXT K:GOTO 480
760 CLS:PRINT:PRINT "END OF PROGRAM"
    :PRINT:STOP
770 CLS:PRINT:PRINT H$(1);" (1) OR "
    ;H$(2);" (2) ? ";
780 LET G$=INKEY$(0):IF G$="" THEN
    GOTO 780
790 LET G=ASC(G$):IF G<>49 AND G<>50
    THEN GOTO 780
800 PRINT CHR$(G);LET H=G-48:RETURN
810 PRINT:PRINT "PRESS RETURN TO
    CONTINUE ..."
820 LET G$=INKEY$(0):IF G$="" THEN
    GOTO 820
830 LET G=ASC(G$):IF G<>13 THEN
    GOTO 820
840 RETURN
850 LET G$=INKEY$(0):IF G$="" THEN
    GOTO 850
860 LET G=ASC(G$):IF G>96 THEN
    LET G=G-32
870 LET G$=CHR$(G):PRINT G$;
880 LET J$=INKEY$(0):IF J$="" THEN
    GOTO 880
890 LET J=ASC(J$):IF J<>13 AND
    J<>127 THEN GOTO 880
```

More next page. . .

```
 900 IF J=13 THEN PRINT:RETURN
■910 PRINT CHR$(8);" ";CHR$(8);
     :GOTO 850
```

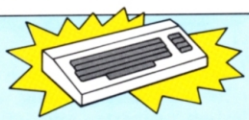
SPECTRUM ALTERATIONS
==========

```
  50 DIM R$(100,2,14):DIM H$(3,10)
 340 IF X$=R$(K,H)(TO L) THEN PRINT
     R$(K,1),R$(K,2):LET FF=1
 420 REM ** LOAD FILE **
 430 LOAD F$ DATA H$()
 440 LOAD F$ DATA R$()
 450 LET NR=VAL R$(100,1)
 460 PRINT:PRINT "WHAT IS THE
     PASSWORD ";:INPUT P$:PRINT P$
     :IF P$=H$(3)(TO LEN P$) THEN
     GOTO 70
 540 IF X$<>R$(K,1)(TO L) THEN
     GOTO 590
 660 LET R$(100,1)=STR$(NR)
 670 SAVE F$ DATA H$()
 680 SAVE F$ DATA R$():GOTO 70
 780 LET G$= INKEY$:IF G$="" THEN
     GOTO 780
 790 LET G=CODE(G$):IF G<>49 AND
     G<>50 THEN GOTO 780
 820 LET G$=INKEY$:IF G$="" THEN
     GOTO 820
 830 LET G=CODE(G$):IF G<>13 THEN
     GOTO 820
 850 LET G$=INKEY$:IF G$="" THEN
     GOTO 850
```

```
860 LET G=CODE(G$):IF G>96 THEN
    LET G=G-32
880 LET J$=INKEY$:IF J$="" THEN
    GOTO 880
890 LET J=CODE(J$):IF J<>13 AND
    J<>12 THEN GOTO 880
```

Replace INPUT X by INPUT X:PRINT X in lines 290 300 320 380 400 520 620 and 630

COMMODORE ALTERATIONS



```
420 OPEN 1,1,0,F$:INPUT#1,H$(1)
    :INPUT#1,H$(2):INPUT#1,H$(3)
430 INPUT#1,NR:FOR K=1 TO NR
440 INPUT#1,R$(K,1):INPUT#1,R$(K,2)
450 NEXT K:CLOSE 1
660 OPEN 1,1,1,F$:PRINT#1,H$(1)
    :PRINT#1,H$(2):PRINT#1,H$(3)
670 PRINT#1,NR:FOR K=1 TO NR:PRINT#1
    ,R$(K,1):PRINT#1,R$(K,2)
680 NEXT K:CLOSE 1:GOTO 70
780 GET G$:IF G$="" THEN GOTO 780
820 GET G$:IF G$="" THEN GOTO 820
850 GET G$:IF G$="" THEN GOTO 850
880 GET J$:IF J$="" THEN GOTO 880
890 LET J=ASC(J$):IF J<>13 AND J<>20
    THEN GOTO 880
910 PRINT CHR$(157);" ";CHR$(157);
    :GOTO 850
```

Replace CLS by PRINT CHR$(147) in lines 70 280 370 400 480 510 620 760 and 770

PROGRAM FOUR

Word Power

Most of the work done in offices involves typing out letters, reports and other documents. Today, a great many offices use computers instead of typewriters. Using a computer in this way is known as **word processing**, and it allows the typist to edit (change) part of a document without having to retype the whole thing. Copies of the document can then be printed onto paper, using a printer attached to the computer.

The program in this chapter is an example of a basic word processing program. It allows you to enter and to edit text (the words), to save it for later use and to alter the layout of the text, i.e. the width of the lines (known as **formatting**). If you have a printer, you can also adjust the program to print out your text onto paper.

Using the Program

When the program is run you will be presented with a blank screen, apart from the top line which says 'LINE 1' and 'COLUMN 1'. As you type words in, the cursor will move across the screen and the numbers at the top of the screen will change. The LINE number will tell you which line the cursor is on, and the COLUMN number will tell you how far along the line you are.

There is no need to press the RETURN or ENTER key when you come to the end of a line, as the text will automatically move to a new line when the last word you type is too long to fit on the screen. However, if you want to start a sentence on a new line, press RETURN/ENTER to take your cursor to the beginning of the next line. When you reach the bottom of the screen, the text will move up a line at a time — this is called 'scrolling'. (You are limited to 100 lines of text in total, but you can change this by altering the value of LS in line 50 of the program.)

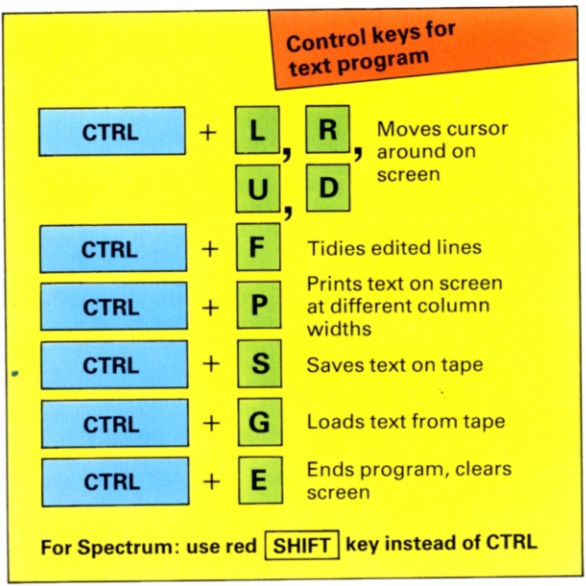
Control Keys

There are a number of special keys that you use in order to adjust your text, or

to save or load it. These are shown in the diagram below. Each time you press one of these keys, you must also hold down the CTRL (CONTROL) key. **If you have a Spectrum, use the red SHIFT key instead.**

To edit your text, you must first move the cursor to the place where you want to make the change. Use the keys CTRL and L (left), R (right), U (up) or D (down) for this. To add extra words in the middle of a line, simply type them in at the right place — the computer will move the rest of the words in that line to make room for them. To remove words, use the DELETE key as usual — the rest of the text will move to fill up the gap.

When you have finished editing you will see that some lines are now too long to see the whole line on screen, while others are too short. To tidy up your lines,



hold down the CTRL and F keys. This will adjust the line where your cursor is. To tidy the whole text, hold down the CTRL key and press F for each line.

The CTRL/P keys allow you to see your text printed on the screen at different column widths. When you press CTRL/P, you will be asked what text width you want. Try a narrow width, say 10 or 20 characters to a line.

There are also two other special commands that you can use with CTRL/P. If you type .L at the beginning of a line, for example, where you have started a new sentence, when you press CTRL/P to display the text all the words after .L will remain on a new line. If you use .P instead, the new line will also be indented 5 spaces.

Using a Printer

If you have a printer attached to your computer, you can use CTRL/P to print your text out on paper. For the BBC and Electron, add these lines to the program:

```
485 VDU2
575 VDU3
```

For the Commodore 64, the new lines are:

```
485 OPEN 1,4:CMD1,
    "TEXT"
575 PRINT#1:CLOSE 1
```

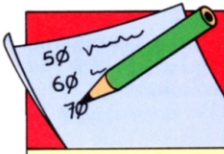
For the Spectrum, you change the instruction PRINT in the program lines 500, 510, 560 and 570 to LPRINT.

Warning

It is best to avoid using commas or double quotation marks when you are typing in your text. With some computers you may find that anything after the comma or quotation marks may not appear on the screen when you load your text back into the computer to use it another time.

Program Note

There are three variables which must be set up to suit your particular make of computer. In line 120, SW is set to the width of your screen (40 characters for the BBC, Electron and Commodore, 32 for the Spectrum), and SL is set to the number of lines for the length of your screen. In line 70, B$ contains a number of spaces between quotation marks. The number of spaces that you type in here must equal the number for the width of your screen (SW). (B$ is used to write a blank line on your screen before your text is put there.)



Line Notes

10 ..... Program title, for saving and loading program lines.

50 ..... Set up space for arrays, 100 lines of text (L$), and 10 editing commands (C).

60 ..... Remove the flashing cursor that is built into the computer (this instruction only needed for the BBC/Electron machines).

70 ..... Set variable B$ to the number of spaces required to make one line of text.

80 ..... Set counters at starting values, for the number of the column the cursor is in (CL), the line the cursor is in (LI), the number of text lines written so far (LL), the number of the first line being displayed (FL), the number of the first characters being displayed (CD).

90–110 ..... Read number of editing commands (NC), then store the ASCII codes for each command into array C — editing commands are the control keys. (See page 10 for details about ASCII codes.)

120 ..... Set screen width (SW) to number of characters and screen length (SL) to number of lines, then clear the screen.

130-140 ..... Print line position and column position of cursor at top of screen.

150 ..... Check if cursor is on top of a character, if it is, store the character in C$; otherwise, leave C$ empty.

More next page. . .

160-170 ..... Print the cursor (a dash symbol) on the screen; then print the character in C$ (makes the cursor flash on and off).

180 ..... Check keyboard; if no key pressed keep flashing the cursor.

190 ..... Otherwise, get ASCII code of key.

200 ..... Set variable for command number chosen (CM) to 0; check to see if code of key pressed matches one of the editing commands; if so, store it in CM.

210 ..... If no match found (so CM still 0), skip to section to check key.

220 ..... Otherwise, go to appropriate section of instructions for that command.

To Format a Line of Text:

230 ..... All lines beginning with REM in a program are simply there as a reminder to the programmer about which bits of the program do what! The computer just ignores them.

240 ..... If cursor line (LI) has reached the end of the text lines (LL) there are no more lines to format so go back and wait for another key.

250 ..... If this line (the line with the cursor in it) and the next line both have text in them, add a space at the end of this line to separate the two lines.

260 ..... Add the second line onto the first one.

270 ..... If this combined line is now too long to fit on screen, go and split it at the right character length and put the rest of it back on the next line.

280–290 ..... Otherwise (if length of combined line is all right), delete both the old lines and print the new one (this would leave a blank line on the screen), then check if screen needs scrolling up one line.

To Load Text from Tape:

300 ..... Reminder message.
310 ..... Clear screen, and print message.
320–330 ..... Ask for name of text file; print message; open file; load in the number of lines and then load the lines themselves into L$; close file.
340 ..... Work out number of text lines to be displayed, if they won't all fit on the screen (allowing 5 lines for gaps at top and bottom of screen), display as many as will fit.
350–360 ..... Clear the screen; print the lines on the screen, then go back and flash cursor.

To Move Cursor Down a Line:

370 ..... Reminder message.
380 ..... If cursor is already on the last line of text, then don't move.
390 ..... Otherwise, move to the next line down, then go and check if cursor has moved off the bottom of the screen and text needs scrolling up.

To Move Cursor Up a Line:

400 ..... Reminder message.
410 ..... If cursor is already on the first line of text, then don't move.
420 ..... Otherwise, move to the previous line, then check if cursor has moved off top of screen and text needs scrolling down.

To Move Cursor Left:

430 ..... Reminder message.
440 ..... If cursor not in first character column, move one place left, then go and check if still on screen.

More next page. . .

450 ..... If already in first column and on first line, don't move.

460 ..... Otherwise, move cursor to end of previous line, then check if on screen.

To Print Text at Different Widths:

470 ..... Reminder message.

480 ..... Clear screen; ask for width of text.

490–500 ..... For each text line in turn, check if first two characters are a special formatting command. If .L, print any text currently stored in temporary space A$ (the remainder of any previous line) and replace it with the new line, but not the characters '.L'.

510 ..... If characters are .P, print the text currently stored in A$ (as above), replace it with the new line and put in 5 spaces at the front of it (new paragaph).

520 ..... If there is some text in A$ and some text in the next line, add a space to A$ to separate the two lines.

530 ..... Add the second line to A$; if A$ isn't full yet go and get another line.

540–550 ..... Set up a pointer (P) to look for a space to split the lines in A$. Start with the pointer set to the width requested (TW), check each character in turn; if it isn't a space, move left and check again.

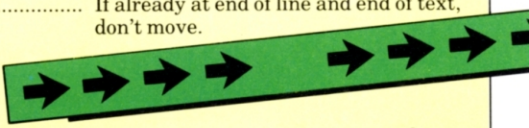
560 ..... Print the line up to this space; put the rest of the line back in A$; if A$ is still more than TW then split A$ again.

570 ..... Repeat all these instructions again (from line 490) for the next line of text; when finished, print any text left in A$ as the last line.

580 ..... Print message to press any key when ready to continue.

590 ..... Check keyboard; if no key pressed, keep checking.
600 ..... Otherwise, go back to previous lines to display text as originally typed in.

To Move Cursor Right:
610 ..... Reminder message.
620 ..... If cursor not at end of line, move right one place, then check if still on screen.
630 ..... If already at end of line and end of text, don't move.



640 ..... Otherwise, move cursor to start of next line, then go and check if still on screen.

To Save Text on Tape:
650 ..... Reminder message.
660–670 ..... Clear screen; print saving message; ask for name for text file; print message.
680–690 ..... Open file; print the number of text lines, then the lines themselves; close file; go and display text as originally typed in.

To Check Keys Being Pressed:
700 ..... If key pressed is RETURN or ENTER (code 13) then move cursor to start of next line; check if cursor still on screen.
710 ..... If key pressed is not DELETE (not code 127), skip next few instructions.
720 ..... If cursor is at start of line, cannot delete anything; check keyboard again.
730 ..... Otherwise, move cursor one place left, remove that character from the line; go and print the new version of the line.

More next page. . .

740 ..... If cursor position is more than the screen width (i.e. have reached end of line) and key pressed is space key (code 32), go to instructions to split line.

750 ..... If key pressed is not a text character, go back to flashing cursor (avoids problem if hit wrong CTRL key by mistake).

760 ..... If it is a text character, add it to the present line.

To Split Line at End of Screen Width:

770–780 ..... Set counter to the end of the screen width; check each character in text line in turn; if not a space, move left and check next character (until find space).

790–800 ..... Move characters to the right of the space to the start of the next line.

810 ..... Delete old version of the first line and replace it with the new version.

820 ..... Move the cursor down to the next line, and position it on top of the character it was over before the text was moved.

To Display Text Typed In (and Scroll if Necessary):

830 ..... Display text line beginning at character column 1; but if cursor will be off the screen, display the part of the line with the cursor in it (this scrolls the text sideways as the cursor moves to the right).

840–850 ..... (As characters are added to the line) delete the current version of the line and replace it with as much of the new version as will fit on the screen.

860 ..... If there are now more text lines than before (because words have been carried over to a new line), change the counter for the number of lines of text written so far (LL).

870 ..... If the line being displayed (i.e. with the cursor in it) is at the bottom of the screen, then add 1 to the number of the first line to be displayed; then go back and re-display all the text (this will scroll the text up a line).

880 ..... If the line being displayed is at the top of the screen and there are some earlier lines which are not being displayed, then subtract 1 from the number of the first line to be displayed; then go back and re-display all the text (this will scroll the text down a line).

890 ..... If the cursor is off the end of the text line, move it back on.

900 ..... If the text that the cursor is on is off the screen, go back to the 'sideways' scroll instruction.

910 ..... Finally, if all the text so far is being displayed properly, go back to the beginning of the program to wait for the next key pressed.

To End Program:

920 ..... Reminder message.

930 ..... Clear screen; print end message and stop program.

Spectrum Lines 805 and 845:
Tell the Spectrum to print only that number of characters that you have typed in onto a line (and not to fill up the remainder of the line with 'rubbish').

Commodore Line 990:
On the Commodore the TAB instruction only moves across the screen, not up and down, so this line is used to move the cursor to the correct line on the screen.

The Program

Main program listing (For BBC/ELECTRON)

```
  10 REM "TEXTED"
▲ 50 DIM L$(100),C(10)
● 60 VDU 23,1,0,0,0,0,0,0,0,0,0
  70 LET B$="  (see Note on page 50)   "
  80 LET CL=1:LET LI=1:LET LL=1:LET
     FL=1:LET CD=1
▲ 90 READ NC:FOR K=1 TO NC:READ C(K)
     :NEXT K
  100 DATA 9
▲ 110 DATA 21,4,12,18,7,19,16,6,5
● 120 LET SW=40:LET SL=25:CLS
● 130 PRINT TAB(4,1);"LINE : ";LI;"   "
● 140 PRINT TAB(20,1);"COLUMN : ";CL;
     "   "
▲ 150 LET C$=" ":IF CL <= LEN(L$(LI))
     THEN LET C$=MID$(L$(LI),CL,1)
● 160 PRINT TAB(CL-CD,3+LI-FL);"-"
● 170 PRINT TAB(CL-CD,3+LI-FL);C$
● 180 LET G$=INKEY$(0):IF G$="" THEN
     GOTO 160
▲ 190 LET G=ASC(G$)
  200 LET CM=0:FOR K=1 TO NC:IF G=C(K)
     THEN LET CM=K
  210 NEXT K:IF CM=0 THEN GOTO 700
▲ 220 ON CM GOTO 410,380,440,620,310,
     660,480,240,930
```

```
  230 REM *** FORMAT LINE ***
  240 IF LI>LL THEN GOTO 130
▲ 250 IF LEN (L$(LI)) > 0 AND LEN
      (L$(LI+1)) > 0 THEN LET L$(LI)
      =L$(LI)+" "
▲ 260 LET L$(LI)=L$(LI)+L$(LI+1):LET
      L$(LI+1)=""
▲ 270 IF LEN (L$(LI)) > SW THEN LET
      CL=SW:GOTO 770
● 280 PRINT TAB(0,3+LI-FL);B$:PRINT
      TAB(0,3+LI-FL);L$(LI)
● 290 LET LI=LI+1:PRINT TAB(0,3+LI-FL)
      ;B$:GOTO 870
  300 REM *** LOAD TEXT FILE ***
■ 310 CLS:PRINT:PRINT "LOADING TEXT"
▲ 320 PRINT "NAME OF TEXT FILE ";
      :INPUT F$:PRINT "PUT TAPE IN AND
      PRESS PLAY"
● 330 F=OPENIN(F$):INPUT #F,LL:FOR
      K=1 TO LL:INPUT #F,L$(K):NEXT K
      :CLOSE #F
  340 LET LD=LL-FL+1:IF LD>SL-5 THEN
      LET LD=SL-5
● 350 CLS:PRINT:PRINT:PRINT:FOR K=1
      TO LD
▲ 360 PRINT LEFT$(L$(K+FL-1),SW):NEXT
      K:GOTO 130
  370 REM *** MOVE DOWN A LINE ***
  380 IF LI=LL THEN GOTO 180
  390 LET LI=LI+1:GOTO 860
  400 REM *** MOVE UP A LINE ***
  410 IF LI=1 THEN GOTO 180
  420 LET LI=LI-1:GOTO 860
  430 REM *** MOVE LEFT ***
```

More next page. . .

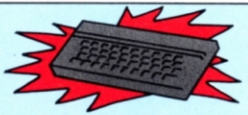
```
  440 IF CL>1 THEN LET CL=CL-1
      :GOTO 900
  450 IF LI=1 THEN GOTO 180
▲ 460 LET LI=LI-1:LET CL=LEN (L$(LI))
      :GOTO 830
  470 REM *** PRINT TEXT ***
■ 480 CLS:PRINT:PRINT "TEXT WIDTH ";
      :INPUT TW
● 490 CLS:PRINT:LET A$="":FOR K=1 TO
      LL:LET X$=LEFT$(L$(K),2)
▲ 500 IF X$=".L" THEN PRINT A$:LET
      A$=MID$(L$(K),3):GOTO 570
▲ 510 IF X$=".P" THEN PRINT A$:LET
      A$="      "+MID$(L$(K),3)
      :GOTO 570
▲ 520 IF LEN (A$) > 0 AND LEN (L$(K))
      > 0 THEN LET A$=A$+" "
▲ 530 LET A$=A$+L$(K):IF LEN (A$) <=TW
      THEN GOTO 570
  540 LET P=TW
▲ 550 IF MID$(A$,P,1)<>" " THEN LET
      P=P-1:GOTO 550
▲ 560 PRINT LEFT$(A$,P-1):LET A$=MID$
      (A$,P+1):IF LEN (A$) > TW THEN
      GOTO 540
  570 NEXT K:PRINT A$
  580 PRINT:PRINT "PRESS ANY KEY WHEN
      READY"
● 590 LET G$=INKEY$(0):IF G$="" THEN
      GOTO 590
  600 GOTO 340
  610 REM *** MOVE RIGHT ***
▲ 620 IF CL <= LEN (L$(LI)) THEN LET
      CL=CL+1:GOTO 900
  630 IF LI=LL THEN GOTO 180
  640 LET LI=LI+1:LET CL=1:GOTO 830
  650 REM *** SAVE TEXT FILE ***
```

```
■ 660 CLS:PRINT:PRINT "SAVING TEXT"
      :PRINT "NAME OF TEXT FILE ";
▲ 670 INPUT F$:PRINT:PRINT "PUT TAPE
      IN"
● 680 LET F=OPENOUT(F$):PRINT #F,LL
● 690 FOR K=1 TO LL:PRINT #F,L$(K)
      :NEXT K:CLOSE #F:GOTO 340
  700 IF G=13 THEN LET LI=LI+1:LET
      CL=1:GOTO 830
● 710 IF G<>127 THEN GOTO 740
  720 IF CL=1 THEN GOTO 180
▲ 730 LET CL=CL-1:LET L$(LI)=LEFT$
      (L$(LI),CL-1)+MID$(L$(LI),CL+1)
      :GOTO 830
  740 IF CL>SW AND G=32 THEN GOTO 770
  750 IF G<32 THEN GOTO 150
▲ 760 LET L$(LI)=LEFT$(L$(LI),CL-1)
      +CHR$(G)+MID$(L$(LI),CL):LET
      CL=CL+1:GOTO 830
  770 LET K=SW
▲ 780 IF MID$(L$(LI),K,1)<>" " THEN
      LET K=K-1:GOTO 780
▲ 790 LET L$(LI+1)=MID$(L$(LI),K+1)+
      " "+L$(LI+1)
▲ 800 LET L$(LI)=LEFT$(L$(LI),K-1)
● 810 PRINT TAB(0,3+LI-FL);B$:PRINT
      TAB(0,3+LI-FL);LEFT$(L$(LI),SW)
  820 LET LI=LI+1:LET CL=CL-K+1
  830 LET CD=1:IF CL>SW THEN LET
      CD=CL+5-SW
● 840 PRINT TAB(0,3+LI-FL);B$
● 850 PRINT TAB(0,3+LI-FL)
      ;MID$(L$(LI),CD,SW)
  860 IF LI>LL THEN LET LL=LI
  870 IF LI-FL=SL-5 THEN LET FL=FL+1
      :GOTO 340
```

More next page. . .

```
 880 IF LI-FL<0 AND FL>1 THEN LET
     FL=FL-1:GOTO 340
▲890 IF CL>LEN(L$(LI))+1 THEN LET
     CL=LEN(L$(LI))+1
 900 IF CL-CD>SW-1 OR CL-CD<0 THEN
     GOTO 830
 910 GOTO 130
 920 REM *** END PROGRAM ***
■930 CLS:PRINT:PRINT "END OF PROGRAM"
     :PRINT:STOP
```

SPECTRUM ALTERATIONS
==========



```
50 DIM L$(100,80):DIM L(100)
   :DIM C(10):DIM G(10)
60 REM
90 READ NC:FOR K=1 TO NC:READ C(K),
   G(K):NEXT K
110 DATA 197,410,205,380,61,440,60,
    620,203,310,195,660,34,480,204,
    240,200,930
120 LET SW=32:LET SL=20:CLS
130 PRINT AT 1,4;"LINE : ";
    STR$(LI);"  "
140 PRINT AT 1,20;"COLUMN : ";
    STR$(CL);"  "
150 LET C$=" ":IF CL <= L(LI) THEN
    LET C$=L$(LI)(CL)
160 PRINT AT 3+LI-FL,CL-CD;"-"
170 PRINT AT 3+LI-FL,CL-CD;C$
180 LET G$=INKEY$:IF G$="" THEN
    GOTO 160
190 LET G=CODE G$
```

```
220 GOTO G(CM)
250 IF L(LI)>0 AND L(LI+1)>0 THEN LET
    L$(LI)=L$(LI) (TO L(LI))+ " ":LET
    L(LI)=L(LI)+1
260 LET L$(LI)=L$(LI)(TO L(LI))+
    L$(LI+1):LET L$(LI+1)="":LET
    L(LI) = L(LI)+L(LI+1):LET
    L(LI+1)=0
270 IF L(LI) > SW THEN LET CL=SW
    :GOTO 770
280 PRINT AT 3+LI-FL,0;B$:PRINT AT
    3+LI-FL,0;L$(LI)(TO L(LI))
290 LET LI=LI+1:PRINT AT
    3+LI-FL,0;B$:GOTO 870
330 LOAD F$ DATA L$():LOAD F$ DATA
    L():LET LL=L(100)
350 CLS:PRINT:PRINT:PRINT:FOR K=1
    TO LD:LET S2=L(K+FL-1):IF S2>SW
    THEN LET S2=SW
360 PRINT L$(K+FL-1)(TO S2):NEXT K
    :GOTO 130
460 LET LI=LI-1:LET CL=L(LI)
    :GOTO 830
490 CLS:PRINT:LET A$="":FOR K=1
    TO LL:LET X$=L$(K) (TO 2)
500 IF X$=".L" THEN PRINT A$:LET
    A$=L$(K)(3 TO L(K)):GOTO 570
510 IF X$=".P" THEN PRINT A$:LET A$=
    "      "+L$(K)(3 TO L(K)):GOTO 570
520 IF LEN (A$) >0 AND L(K) >0 THEN
    LET A$=A$+" "
530 LET A$=A$+L$(K) (TO L(K)):IF
    LEN (A$) <= TW THEN GOTO 570
550 IF A$ (P)<>" " THEN LET P=P-1:
    GOTO 550
```

More next page. . .

```
560 PRINT A$ (TO P-1):LET A$=A$ (P+1
    TO):IF LEN (A$) > TW THEN
    GOTO 540
590 LET G$=INKEY$:IF G$="" THEN
    GOTO 590
620 IF CL<=L(LI) THEN LET CL=CL+1
    :GOTO 900
680 LET L(100)=LL:SAVE F$ DATA L$()
690 SAVE F$ DATA L():GOTO 340
710 IF G<>12 AND G<>95 THEN GOTO 740
730 LET CL=CL-1:LET L$(LI)=L$(LI)
    (TO CL-1) + L$(LI) (CL+1 TO):LET
    L(LI)=L(LI)-1:GOTO 830
760 LET L$(LI)=L$(LI)(TO CL-1)+
    CHR$(G)+L$(LI)(CL TO):LET L(LI)=
    L(LI)+1:LET CL=CL+1:GOTO 830
780 IF L$(LI)(K)<>" " THEN LET K=K-1
    :GOTO 780
790 LET L$(LI+1)=L$(LI) (K+1 TO
    L(LI))+" "+L$(LI+1):LET L(LI+1)=
    L(LI+1)+L(LI)-K+1
800 LET L$(LI)=L$(LI)(TO K-1):LET
    L(LI)=K-1
805 LET S2=L(LI):IF S2>SW THEN LET
    S2=SW
810 PRINT AT 3+LI-FL,0;B$:PRINT AT
    3+LI-FL,0;L$(LI)(TO S2)
840 PRINT AT 3+LI-FL,0;B$
845 LET S2=L(LI)-CD+1:IF S2>SW THEN
    LET S2=SW
850 PRINT AT 3+LI-FL,0;L$(LI)(CD TO
    CD+S2-1)
890 IF CL>L(LI)+1 THEN LET CL=L(LI)+1
```

Replace INPUT X by INPUT X:PRINT X in lines 320 and 670

Note: Additional lines at 805 and 845.

COMMODORE
ALTERATIONS
==========

```
60 REM
120 LET SW=40:LET SL=20:PRINT
    CHR$(147)
130 PRINT CHR$(19):PRINT TAB(4);
    "LINE : ";STR$(LI);"  ";
140 PRINT TAB(20);"COLUMN : ";
    STR$(CL);"  "
160 GOSUB 990:PRINT TAB(CL-CD);"-"
170 GOSUB 990:PRINT TAB(CL-CD);C$
180 GET G$:IF G$="" THEN GOTO 160
280 GOSUB 990:PRINT B$:GOSUB 990:
    PRINT LI$(LI)
290 LET LI=LI+1:GOSUB 990:PRINT B$:
    GOTO 870
330 OPEN 1,1,0,F$:INPUT#1,LL:FOR
    K=1 TO LL:INPUT#1,L$(K):NEXT K
    :CLOSE 1
590 GET G$:IF G$="" THEN GOTO 590
680 OPEN 1,1,1,F$:PRINT#1,LL
690 FOR K=1 TO LL:PRINT#1,L$(K)
    :NEXT K:CLOSE 1:GOTO 340
710 IF G<>20 THEN GOTO 740
810 GOSUB 990:PRINT B$:GOSUB 990:
    PRINT LEFT$(L$(LI),SW)
840 GOSUB 990:PRINT B$
850 GOSUB 990:PRINT
    MID$(L$(LI),CD,SW)
990 PRINT CHR$(19):FOR Z=1 TO LI-FL+3
    :PRINT:NEXT Z:POKE 213,40:RETURN
```

Replace CLS by PRINT CHR$(147) in lines 310 350 480 490 660 and 930

Note: Additional line at 990.

Adventure

One of the first computer games to gain a devoted following was called 'Colossal Cave'. It was written in the late 1960's, using very large computers and, before long, a whole generation of computer users were hooked on trying to solve its puzzles! Adventure games are now within the scope of the home micro, so here is one to test your wits, but don't blame me if you get addicted!

The bad news is that the program is rather long. However, remember that you can type the instructions into your machine in stages, by saving the program up to the point where you want to finish typing, then loading it back in when you are ready to continue.

Details of how the program works are given later on in this chapter, but try not to read them until after you have played the game!

Using the Program

When the program starts, you will find yourself in a stone hut — the computer will describe your location, and tell you of any objects that are there. You move from place to place by typing the direction — NORTH (N), SOUTH (S), EAST (E), WEST (W), UP (U) or DOWN (D).

You can pick things up or put them down by typing commands such as TAKE and DROP. There are a number of puzzles to solve, and in many cases the various objects you find will help you.

LIST will tell you what you are carrying at any time, and LOOK describes your location again.

There are other commands you can use as well, such as THROW, LOCK, LIGHT or SAY. You can abbreviate the commands by typing, say, TH for THROW or TA for TAKE. The trick is to find the correct command for each situation. Even though the command prompts the reply 'Don't be silly!' or 'Nothing happens' in one situation, it may work another time. If you use a

command the computer does not recognize at all, it will say so.

Typing SCORE will give you your score so far. There are five pieces of treasure, and you will earn 10 points for each piece found, plus 20 points for getting the treasure back to the stone hut — giving a maximum score of 150. If you have an accident you can still finish the game, but you will not get the maximum score!

A Clue

The best way to map a maze is to use objects as markers!

How it Works

Several arrays are necessary in this program. The first two are: V$ which holds the list of verbs you can use, and O$ which holds the objects. These two arrays enable the computer to understand the commands you give it. When you type in a command, the computer checks through it until it finds a space. To the computer, this means that it has found the end of the verb, so the word to the left of the space is compared with those in V$. If no match is found, the computer will not understand your command.

The first nine verbs in the array do not need to be used with the name of an object. But for the others, the computer will take the word to the right of the space as the object word. (If you haven't typed in a second word here, you will be asked to do so.) This will be checked against O$ and, if not found, the computer will not understand.

Array D$ stores a description of each object, and L$ stores a description of each location — these are used to set the scene when the player moves to a new point in the game.

Array L contains the location of where each object can be found; array M is also used for objects such as bridges, which can be seen from a second location, too. If a location is equal to 0, this means that the object does not exist yet, for example, the magic bridge. A location of 99, means that the object is being carried by the player. A negative location value

means the object cannot be taken by the player — this avoids problems with commands such as 'TAKE BRIDGE'!

Finally, each location has an entry in array N$, giving the numbers of the locations next-door to it. Each of these entries is 12 figures long, so it can hold a two-figure location for each of the six possible directions. For example, the entry in N$(8) is 050910110000 — this means that moving North from location 8 will take you to location 5, moving East takes you to location 9, South to location 10, and West to 11. Going Up or Down from location 8 will result in a message saying 'You can't go that way', as their direction value is 00.

More About Commands

The first six commands are directions. The others each have their own section of program. The computer decides which section to use by means of line 130, which says:

```
ON V-6 GOSUB 360,
540,. . .
```

For example, if you type LIST (verb 7), the computer works out V-6, giving 1, and jumps to the first line number (360). Typing SCORE (verb 8) causes a jump to line 540, and so on through all the numbers in line 130.

The Spectrum computer does not have the ON. . .GOSUB command; instead, the line numbers are stored in an extra array, G, and the instruction used here is GOSUB G(V).

The TAKE command checks that the object is at your present location, and that it is portable. If so, its location is set to 99 to show that you are now carrying it.

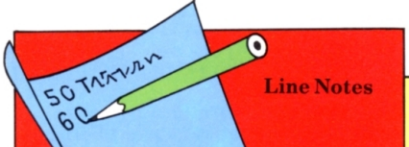
The DROP command checks that the object is being carried (location = 99) and then re-sets its location to the present position.

The THROW command works as for DROP, except when the object thrown is the AXE or the BIRD!

LIST checks the location of all objects, and prints those which are being carried.

The LOOK command prints the description of your present location. It then checks each object in turn, and prints its description if it is at that location (but not if you are carrying it).

The SCORE command looks at objects 11 to 16 (the pieces of treasure), and adds 10 points if its location is 99 (being carried) or 30 points if it is at location 1 (the hut).



Line Notes

10 ..... Program title, for saving and loading program lines.

50 ..... Set up space for arrays (see page 67).

60 ..... Use subroutine (15) to store descriptions given in DATA statements in arrays. Set variables: location (LC) is 1 (to begin game); number of objects being carried (NC) is Ø; bridge flag (BF) is Ø (1 = bridge exists); dwarf flag (DF) is Ø (1 = dwarf exists); lantern flag (LF) is Ø (1 = lantern lit); grating flag (GF) is Ø (1 = grating open); go to subroutine (14) to describe present location.

70 ..... Use subroutine (11) to obtain a command.

80 ..... If command not a direction, skip next lot of instructions.

90 ..... Otherwise, if the dwarf exists (DF=1) there is a 60% chance that the dwarf will block further movement, in which case go to subroutine (12) to say so and jump to the instructions to control the dwarf.

100 ..... Work out the location the direction command goes to.

110 ..... If the move is allowed (number of location is more than Ø), change present location to new location; use subroutine (14) to describe it, then jump to the lines to control the dwarf.

More next page. . .

120 ..... Otherwise (if LC=Ø), say so, then jump to the dwarf lines.

130 ..... (If command not a direction, i.e. verb number is more than 6) subtract 6 from the verb number and jump to relevant subroutine for that command.

140 ..... If verb = TAKE (10) and object = AXE (4), go back to get next command (avoids dwarf routine in this situation).

To Control Dwarf:

150 ..... If dwarf exists, there is a 20% chance it will throw a knife, if so, go to lines in subroutine (13), then go back for next command.

160 ..... If underground (LC more than 4) and no dwarf exists, there is a 20% chance of one appearing.

170 ..... If underground and dwarf exists, then move dwarf (8) to the present location (so dwarf keeps following you).

180 ..... If above ground (LC less than 5), remove dwarf (so does not follow you back to the hut).

190 ..... Go back for next command.

SUBROUTINE (1) to Take:

200 ..... Reminder message (see page 52 for details).

210 ..... If object's location is 99 it is already being carried, say so.

220 ..... If object not at present location, say so. (Object's location may be a plus or minus number so ABS is used here to make the computer treat any minus location as if it were a plus location.)

230 ..... If object's location number is a minus number (therefore less than Ø), it cannot be picked up.

240 ..... If number of objects already carried (NC) is 5, cannot carry any more.

250 ..... If object is not treasure (objects 11 to 15), skip next two lines.

260–270 ..... If at dragon's location or snake's location, use subroutine (12) to say that movement is blocked.

280 ..... If everything OK, say so; add 1 to number of objects carried; set object's location to 99 and return to main program.

SUBROUTINE (2) to Drop:

290 ..... Reminder message.

300 ..... If object's location not 99 it is not being carried, say so.

310 ..... Otherwise, reduce the number of objects carried by 1.

320 ..... Providing it is not object 14, or object 15 is already at this location, leave object at this location.

330 ..... Otherwise, print message; remove object 14 from program and replace it with object 16; reduce maximum possible score by 30 points.

340 ..... Return to main program.

SUBROUTINE (3) to List:

350 ..... Reminder message.

360 ..... Set carry flag (CF) to Ø (no description of objects printed yet); for each object in turn, check if it is being carried; if not, skip next two lines.

370 ..... If this is the first object to be printed (so CF=Ø), print message and set CF to 1 (makes sure message isn't printed more than once).

More next page. . .

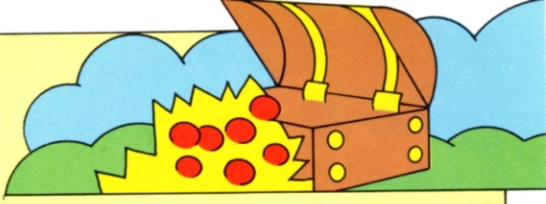
380 ..... Print description of that object.
390 ..... Repeat instructions (from line 360) for each object. If no objects carried (so CF still Ø), say so.
400 ..... Return to main program.

SUBROUTINE (4) to Wave:

410 ..... Reminder message.
420 ..... If not carrying the object, say so.
430 ..... If the object is not the right one, print silly message.
440 ..... If not at location 7 or location 14, nothing happens, so say so.
450 ..... If bridge already exists (BF=1), skip next two lines.
460 ..... Otherwise, print message.
470 ..... Create bridge by setting its location to −7 and −14 (as it cannot be carried) and skip next two lines.
480 ..... Print message.
490 ..... Set bridge location to Ø (i.e. remove bridge).
500–510 ..... Change the neighbouring locations accordingly (i.e. whether bridge exists or not). This sets the next location going West from location 7 to ØØ (can't go that way) or 14, and the next location going East from location 14 to ØØ (can't go) or Ø7.
520 ..... Make bridge flag equal to 1 minus the present BF number (i.e. 1 − 1 = Ø bridge doesn't exist, or 1 − Ø = 1 bridge exists).

SUBROUTINE (5) to Score:

530 ..... Reminder message.
540–550 ..... Set score to Ø; for each piece of treasure (objects 11 to 15), check if location is 1 (in hut) or 99 (being carried), add 30 points or 10 points respectively.



560 ..... When all treasure checked, print score and maximum score possible; if score less than maximum possible, return to main program.

570 ..... Otherwise, print finish message and stop program.

SUBROUTINE (6) to Throw:

580 ..... Reminder message.

590 ..... If not carrying object, use message in subroutine (2) to say so.

600 ..... If object is not bird (5), skip next six lines.

610–620 ..... If object 6 or object 7 are at this location, set variable B to the number of the beast present.

630 ..... If neither are here (so B is still Ø), then THROW has the same meaning as DROP so use subroutine (2).

640 ..... Reduce number of objects being carried by 1 (i.e. drop bird), and print message.

650 ..... If the bird is thrown at object 7, print message and remove object 7.

660 ..... If it is thrown at object 6, print message and remove bird.

670 ..... If object thrown is not the axe, or there is no dwarf present, use subroutine (2) for DROP.

More next page. . .

680 ..... Otherwise, there is a 10% chance the dwarf will dodge the axe, if this is the case, say so and go to DROP routine.

690 ..... If dwarf does not dodge, print message and remove dwarf, then go to DROP routine.

SUBROUTINE (7) to Fight:

700 ..... Reminder message.

710 ..... If object present is not object 6, print message.

720–730 ..... If it is, print question and wait for answer.

740 ..... If answer is given as a lower-case (small) letter, convert it to a capital letter. (See page 11 for details.)

750 ..... If answer is not Y or N, then go back and wait for another key to be pressed.

760 ..... If answer is N then return to main program.

770 ..... Otherwise (answer is Y), print message; remove object 6, then return to main program.

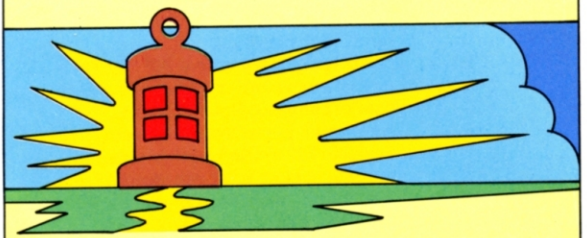
SUBROUTINE (8) to Say:

780 ..... Reminder message.

790 ..... If present location number is not 16 or 1, or word to say is not object 17, then print message followed by the word that was typed in.

800 ..... Otherwise, print message, then count up to 400 (this simply makes the computer pause for a few seconds).

810 ..... Change the present location from 1 to 16 or 16 to 1 (depending on what LC is), go to subroutine (14) to describe location; return to main program.



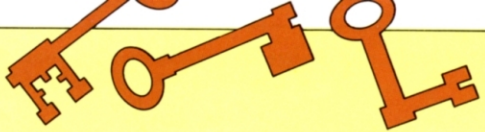
SUBROUTINE (9) to Light/Extinguish:

820 ..... Reminder message.
830 ..... If object is not lantern, print message.
840 ..... Otherwise, let lantern flag (LF) equal the number of the verb minus 16, i.e. if verb is LIGHT (17) LF=1 (on), if it is EXTINGUISH (16) LF=Ø (off).

SUBROUTINE (10) to Lock/Unlock:

850 ..... Reminder message.
860 ..... If object is not grating, say so.
870 ..... If not carrying keys (object 2), say so.
880–900 ..... Otherwise, let grating flag (GF) equal 19 minus the number of the verb — if verb is LOCK (19) GF=Ø (shut), if it is UNLOCK (18) GF=1 (open). Change neighbouring locations accordingly. This sets the next location going down from location 4 to ØØ (can't go) or Ø4; and the next location going up from location 4 to ØØ (can't go) or Ø3.
910 ..... Return to main program.

More next page. . .



SUBROUTINE (11) to Decode Command Given:

920 ..... Reminder message.

930 ..... Clear temporary stores A$ (for verb part of command) and B$ (for object part of command); ask for next command; get answer (K$); set variable LK to the length of this command; set character counter (K) to Ø.

940 ..... Add 1 to K counter; if more than the length of the command skip next line (having reached end of command without finding a space — this instruction would obviously not come into effect for the first character added).

950 ..... Otherwise, if this character in the command is not a space, then go back and increase counter number by one.

960 ..... Take all characters up to this space (or up to end of command) and store in A$ (as the verb); set variable LA to match the length of this verb.

970 ..... Set the number of this verb to Ø; set a counter C to point to each verb in turn. For each verb, check if the first letters (the number of letters given in LA) match the letters in A$; if they do, set V to the number of this verb; then set C to 99 (more than the number of verbs known) so that no more verbs will be checked.

980 ..... Otherwise, repeat instructions until each known verb has been checked; if no match found (so V is still Ø), go and print 'I don't understand' message.

990 ..... If the verb (V) is less than 10, no object word is required in the command, so return to main program.

1000 ..... Otherwise, if character counter (K) is more than or equal to the length of the command (LK) the end of the command has been reached and no second word has been typed in, so ask for the object; store it in B$; set variable LB to match the length of the word, and skip a line.

1010 ..... (If V is not less than 10) take the characters to the right of the space (found in line 950), store them in B$ instead (as the object word), and set variable LB to match the length of B$.

1020 ..... Set counter C to point to each object in turn; for each object, check if it matches the word in B$; if it does, set variable O to the object number, and set C to 99.

1030 ..... Repeat instructions until all objects checked; if match found (so O is more than Ø), return to main program.

1040–1050 ..... (If no match found) print message and go back for a new command.

SUBROUTINE (12) to Block Movement:

1060 ..... Reminder message.

1070 ..... Print message, inserting the name of the object that is blocking the way (OB), then return to main program.

SUBROUTINE (13) to Throw Axe or Knife:

1080 ..... Reminder message.

1090 ..... If the axe (object 4) already exists, print message and make dwarf appear.

1100–1120 ..... Otherwise (if axe does not exist), print message and leave axe at location.

More next page. . .

1130–1140 ..... Print message, and wait for a short time.
1150 ..... There is an 80% chance that the knife misses, if so, return to main program.
1160 ..... Otherwise, the knife hits and the player is 'killed'.
1170 ..... Go to subroutine (5) to print score, and end program.

SUBROUTINE (14); to Describe Location:

1180 ..... Reminder message.
1190 ..... Clear screen.
1200 ..... If lantern is on (LF=1) and is at present location or being carried, skip next line.
1210 ..... If not, and if present location is underground (LC is more than 4), then it is too dark to see, so print message and return to main program.
1220 ..... Print two-line description of present location.
1230 ..... Set object flag (OF) to Ø (no objects described yet); for each object in turn, if object not at this location skip next two lines. (The ABS command means 'ignore any minus signs' — this makes the computer describe any object with a minus location, even though it cannot be moved.)
1240 ..... If the object is the first one found at this location (so OF is still Ø), print opening message and set OF to 1 to avoid message being repeated.
1250 ..... Print the object's description.
1260 ..... Repeat instructions for each object.
1270 ..... If not at locations 3 or 4, skip two lines.
1280–1290 ..... Print message to say if grating is unlocked (GF=1) or locked (GF=Ø).
1300 ..... If at location 16, print message (which contains clue).

1310 ..... If there is a dwarf there and the location is underground, print a warning.
1320 ..... Return to main program.

SUBROUTINE (15) to Read Descriptions and Data:

1330 ..... Reminder message.
1340 ..... Read number of verbs (NV), then the verbs themselves and store in array V$ (and, in the Spectrum version, the subroutine line numbers in array G).
1350 ..... Read number of objects (NO); for each object, read the name of the object and put in O$, its description into D$, and two locations into L and M.
1360 ..... Read number of locations (NL); for each location, read two lines of description and put in L$(1) and L$(2), and its neighbouring locations in N$.
1370 ..... Set maximum score to 150 (for 5 pieces of treasure) and go back to main program.
1380–2070 ..... All the DATA statements, which contain the verbs, the objects, the descriptions of locations, and the numbers for the neighbouring locations.

The Program

Main program listing (For BBC/ELECTRON)

```
 10 REM "ADVENT"
▲50 DIM V$(20),O$(25),D$(25),L(25),
    M(25),L$(30,2),N$(30)
 60 GOSUB 1340:LET LC=1:LET NC=0
    :LET BF=0:LET DF=0:LET LF=0:LET
    GF=0:GOSUB 1190
 70 GOSUB 930
 80 IF V>6 THEN GOTO 130
▲90 IF DF=1 AND RND(1)<.6 THEN LET
    OB=8:GOSUB 1070:GOTO 150
▲100 LET NL=VAL(MID$(N$(LC),2*V-1,2))
 110 IF NL>0 THEN LET LC=NL:PRINT
    "O.K.":GOSUB 1190:GOTO 150
 120 PRINT "YOU CAN'T GO THAT WAY !"
    :GOTO 150
▲130 ON V-6 GOSUB 360,540,1190,210,
    300,420,590,710,790,830,830,
    860,860
 140 IF V=10 AND O=4 THEN GOTO 70
▲150 IF DF=1 AND RND(1)<.2 THEN
    GOSUB 1130:GOTO 70
▲160 IF LC>4 AND RND(1)<.2 AND DF=0
    THEN GOSUB 1090
 170 IF LC>4 AND DF=1 THEN LET
    L(8)=-LC
 180 IF LC<5 THEN LET DF=0:LET L(8)=0
 190 GOTO 70
 200 REM ***** TAKE *****
 210 IF L(O)=99 THEN PRINT "YOU'VE
    ALREADY GOT IT !":RETURN
 220 IF ABS(L(O))<>LC THEN PRINT "IT
    ISN'T HERE !":RETURN
```

```
230 IF L(0)<0 THEN PRINT "YOU MUST
    BE JOKING !":RETURN
240 IF NC=5 THEN PRINT "YOU CAN'T
    CARRY ANY MORE":RETURN
250 IF O<11 OR O>15 THEN GOTO 280
260 IF L(6)=-LC THEN LET OB=6
    :GOSUB 1070:RETURN
270 IF L(7)=-LC THEN LET OB=7
    :GOSUB 1070:RETURN
280 PRINT "O.K.":LET NC=NC+1:LET
    L(O)=99:RETURN
290 REM ***** DROP *****
300 IF L(O)<>99 THEN PRINT "YOU
    HAVEN'T GOT IT !":RETURN
310 LET NC=NC-1
320 IF O<>14 OR L(15)=LC THEN PRINT
    "O.K.":LET L(O)=LC:RETURN
330 PRINT "IT CRASHES TO THE FLOOR
    !!":LET L(14)=0:LET L(16)=LC
    :LET MS=MS-30
340 RETURN
350 REM ***** LIST *****
360 LET CF=0:FOR K=1 TO NO:IF
    L(K)<>99 THEN GOTO 390
370 IF CF=0 THEN PRINT "YOU ARE
    CARRYING...":LET CF=1
380 PRINT D$(K)
390 NEXT K:IF CF=0 THEN PRINT "YOU
    AREN'T CARRYING ANYTHING !"
400 RETURN
410 REM ***** WAVE *****
420 IF L(O)<>99 THEN PRINT "YOU
    HAVEN'T GOT IT !":RETURN
430 IF O<>3 THEN PRINT "IF IT MAKES
    YOU HAPPY !":RETURN
```

More next page. . .

```
 440 IF LC<>7 AND LC<>14 THEN PRINT
     "NOTHING HAPPENS":RETURN
 450 IF BF=1 THEN GOTO 480
 460 PRINT "A CRYSTAL BRIDGE
     APPEARS !!"
 470 LET L(10)=-7:LET M(10)=-14:LET
     A$="14":LET B$="07":GOTO 500
 480 PRINT "THE BRIDGE VANISHES !!"
 490 LET L(10)=0:LET M(10)=0:LET
     A$="00":LET B$="00"
▲500 LET N$(7)=LEFT$(N$(7),6)+A$+
     MID$(N$(7),9)
▲510 LET N$(14)=LEFT$(N$(14),2)+B$+
     MID$(N$(14),5)
 520 LET BF=1-BF:RETURN
 530 REM ***** SCORE *****
 540 LET SC=0:FOR K=11 TO 15:IF
     L(K)=1 THEN LET SC=SC+30
 550 IF L(K)=99 THEN LET SC=SC+10
 560 NEXT K:PRINT "YOUR SCORE IS ";
     SC;" OUT OF ";MS:IF SC<MS THEN
     RETURN
 570 PRINT:PRINT "YOU HAVE COMPLETED
     YOUR QUEST !!":PRINT:STOP
 580 REM ***** THROW *****
 590 IF L(0)<>99 THEN GOTO 290
 600 IF O<>5 THEN GOTO 670
```

```
  610 LET B=0:IF L(6)=-LC THEN LET B=6
  620 IF L(7)=-LC THEN LET B=7
  630 IF B=0 THEN GOTO 290
  640 LET NC=NC-1:PRINT "THE BIRD
      ATTACKS THE ";O$(B)
  650 IF B=7 THEN PRINT "AND DRIVES IT
      AWAY !!":LET L(7)=0:LET L(5)=LC
      :RETURN
  660 IF B=6 THEN PRINT "AND GETS
      BURNED TO A CINDER !!":LET
      L(5)=0:RETURN
  670 IF O<>4 OR DF=0 THEN GOTO 290
▲ 680 IF RND(1)<.1 THEN PRINT "THE
      DWARF DODGES IT !":GOTO 290
  690 PRINT "THE DWARF VANISHES IN
      SMOKE":LET L(8)=0:LET DF=0
      :GOTO 290
  700 REM ***** FIGHT *****
  710 IF O<>6 THEN PRINT "DON'T BE
      SILLY !!":RETURN
  720 PRINT "WHAT - WITH YOUR BARE
      HANDS ??!!"
● 730 LET G$=INKEY$(0):IF G$="" THEN
      GOTO 730
▲ 740 LET G=ASC(G$):IF G>96 THEN
      LET G=G-32
  750 LET G$=CHR$(G):IF G$<>"Y" AND
      G$<>"N" THEN GOTO 730
  760 IF G$="N" THEN RETURN
  770 PRINT "YOU DEFEAT IT !!":PRINT
      "IT'S A STRANGE WORLD !":LET
      L(6)=0:RETURN
  780 REM ***** SAY *****
  790 IF (LC<>16 AND LC<>1) OR O<>17
      THEN PRINT "O.K. '";B$;"'"
      :RETURN
```

More next page. . .

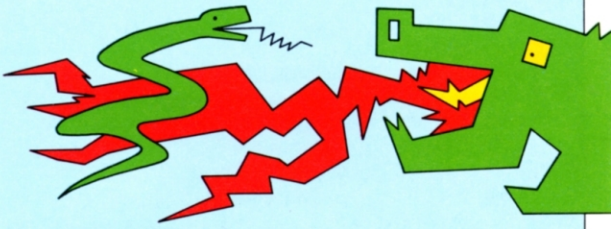
```
  800 PRINT "YOU FEEL YOURSELF
      SPINNING ROUND ...":FOR Z=1
      TO 400:NEXT Z
  810 LET LC=17-LC:GOSUB 1190:RETURN
  820 REM ***** LIGHT/EXTINGUISH *****
  830 IF O<>1 THEN PRINT "DON'T BE
      SILLY !":RETURN
  840 LET LF=V-16:RETURN
  850 REM ***** LOCK/UNLOCK *****
  860 IF O<>9 THEN PRINT "DON'T BE
      SILLY !":RETURN
  870 IF L(2)<>99 THEN PRINT "YOU
      HAVEN'T ANY KEYS !":RETURN
  880 LET GF=19-V:IF GF=0 THEN LET
      A$="00":LET B$="00":GOTO 900
  890 LET A$="04":LET B$="03"
▲ 900 LET N$(3)=LEFT$(N$(3),8)+A$+
      MID$(N$(3),11):LET N$(4)=LEFT$
      (N$(4),10)+B$
  910 RETURN
  920 REM ***** DECODE COMMAND *****
▲ 930 LET A$="":LET B$="":PRINT "WHAT
      NOW ";:INPUT K$:LET LK=LEN(K$)
      :LET K=0
  940 LET K=K+1:IF K>LK THEN GOTO 960
▲ 950 IF MID$(K$,K,1)<>" " THEN
      GOTO 940
▲ 960 LET A$=LEFT$(K$,K-1):LET
      LA=LEN(A$)
▲ 970 LET V=0:FOR C=1 TO NV:IF A$=
      LEFT$(V$(C),LA) THEN LET V=C
      :LET C=99
  980 NEXT C:IF V=0 THEN GOTO 1040
  990 IF V<10 THEN RETURN
▲1000 IF K>=LK THEN PRINT A$;" WHAT
      ...";:INPUT B$:LET LB=LEN(B$)
      :GOTO 1020
```

```
▲1010 LET B$=MID$(K$,K+1):LET
      LB=LEN(B$)
▲1020 LET O=0:FOR C=1 TO NO:IF B$=
      LEFT$(O$(C),LB) THEN LET O=C
      :LET C=99
  1030 NEXT C:IF O>0 THEN RETURN
  1040 PRINT:PRINT "SORRY - I DON'T
      UNDERSTAND"
  1050 PRINT:GOTO 930
  1060 REM ***** CAN'T DO *****
  1070 PRINT "THE ";O$(OB);" STOPS
      YOU !!":RETURN
  1080 REM ***** DWARF *****
  1090 IF L(4)<>0 THEN PRINT "A DWARF
      COMES ROUND A CORNER !":LET
      DF=1:RETURN
  1100 PRINT "SUDDENLY, A DWARF
      APPEARS. HE"
  1110 PRINT "HURLS AN AXE AT YOU,
      CURSES"
  1120 PRINT "WHEN IT MISSES, AND RUNS
      AWAY":LET L(4)=LC:RETURN
  1130 PRINT "THE DWARF THROWS A KNIFE
      AT YOU ..."
  1140 FOR Z=1 TO 400:NEXT Z
▲1150 IF RND(1)<.8 THEN PRINT "IT
      MISSES YOU !":RETURN
  1160 PRINT "IT HITS YOU... YOU SINK"
      :PRINT "TO THE FLOOR, MORTALLY
      WOUNDED !"
  1170 GOSUB 530:PRINT:STOP
  1180 REM ***** LOOK AROUND *****
■1190 CLS:PRINT
  1200 IF LF=1 AND (L(1)=LC OR
      L(1)=99) THEN GOTO 1220
```

More next page. . .

```
 1210 IF LC>4 THEN PRINT "IT IS TOO
      DARK TO SEE...":RETURN
 1220 PRINT L$(LC,1):PRINT L$(LC,2)
 1230 LET OF=0:FOR K=1 TO NO:IF
      ABS(L(K))<>LC AND ABS(M(K))<>LC
      THEN GOTO 1260
 1240 IF OF=0 THEN PRINT "YOU SEE
      ...":LET OF=1
 1250 PRINT D$(K)
 1260 NEXT K
 1270 IF LC<>3 AND LC<>4 THEN
      GOTO 1300
 1280 PRINT "THE GRATING IS ";:IF
      GF=1 THEN PRINT "UN";
 1290 PRINT "LOCKED"
 1300 IF LC=16 THEN PRINT "A HOLLOW
      VOICE SAYS 'HELLO'"
 1310 IF DF=1 AND LC>4 THEN PRINT
      "THERE IS A NASTY DWARF NEXT TO
      YOU..."
 1320 RETURN
 1330 REM ***** READ DATA *****
▲1340 READ NV:FOR K=1 TO NV:READ
      V$(K):NEXT K
 1350 READ NO:FOR K=1 TO NO:READ
      O$(K),D$(K),L(K),M(K):NEXT K
 1360 READ NL:FOR K=1 TO NL:READ
      L$(K,1),L$(K,2),N$(K):NEXT K
 1370 LET MS=150:RETURN
▲1380 DATA 19,"NORTH","EAST","SOUTH",
      "WEST","DOWN","UP","LIST",
      "SCORE","LOOK"
▲1390 DATA "TAKE","DROP","WAVE",
      "THROW","FIGHT","SAY",
      "EXTINGUISH","LIGHT"
▲1400 DATA "UNLOCK","LOCK"
```

```
1410 DATA 17
1420 DATA "LANTERN","A BRASS
     LANTERN",1,0
1430 DATA "KEYS","A BUNCH OF
     KEYS",1,0
1440 DATA "WAND","A MAGIC WAND",4,0
1450 DATA "AXE","A SMALL GOLDEN
     AXE",0,0
1460 DATA "BIRD","A LITTLE SINGING
     BIRD",5,0
```



```
1470 DATA "DRAGON","A LARGE GREEN
     DRAGON !",-15,0
1480 DATA "SNAKE","A COILED HISSING
     SNAKE",-6,0
1490 DATA "DWARF","A NASTY DWARF
     WITH A SHARP KNIFE !",0,0
1500 DATA "GRATING","A GRATING SET
     IN CONCRETE",-3,-4
1510 DATA "BRIDGE","A CRYSTAL BRIDGE
     OVER THE CHASM",0,0
1520 DATA "SILVER","SOME BARS OF
     SILVER",6,0
1530 DATA "GOLD","A PILE OF GOLD
     PIECES",14,0
1540 DATA "JEWELS","A BAG OF
     JEWELS",13,0
```

More next page. . .

```
1550 DATA "VASE","A DELICATE MING
     VASE",15,0
1560 DATA "CUSHION","AN ORNATE
     CUSHION",16,0
1570 DATA "","SOME WORTHLESS BITS OF
     POTTERY",0,0
1580 DATA "HELLO","",0,0
1590 DATA 16
1600 DATA "YOU ARE INSIDE A STONE
     HUT"
1610 DATA "THERE IS A DOOR TO THE
     WEST"
1620 DATA "000000020000"
1630 DATA "A PATH LEADS SOUTH."
1640 DATA "THERE IS A HUT TO THE
     EAST"
1650 DATA "000103000000"
1660 DATA "YOU ARE IN A SHALLOW
     DEPRESSION"
1670 DATA "A PASSAGE LEADS
     UNDERGROUND"
1680 DATA "020000000000"
1690 DATA "YOU'RE IN AN UNDERGROUND
     PASSAGE"
1700 DATA "LIGHT FILTERS THROUGH A
     GRATING"
1710 DATA "000000050000"
1720 DATA "YOU ARE IN AN EAST-WEST
     PASSAGE"
1730 DATA "WITH A HOLE IN THE SOUTH
     WALL"
1740 DATA "000408060000"
1750 DATA "YOU ARE IN A GREAT
     HALLWAY, WITH"
1760 DATA "DOORS TO THE EAST AND
     WEST"
```

```
1770 DATA "000500070000"
1780 DATA "YOU ARE ON THE EAST BANK
     OF A"
1790 DATA "DEEP CHASM"
1800 DATA "000600000000"
1810 DATA "YOU ARE IN A MAZE OF
     TWISTY"
1820 DATA "PASSAGES, ALL ALIKE"
1830 DATA "050910110000"
1840 DATA "YOU ARE IN A MAZE OF
     TWISTY"
1850 DATA "PASSAGES, ALL ALIKE"
1860 DATA "110910080000"
1870 DATA "YOU ARE IN A MAZE OF
     TWISTY"
1880 DATA "PASSAGES, ALL ALIKE"
1890 DATA "080910120000"
1900 DATA "YOU ARE IN A MAZE OF
     TWISTY"
1910 DATA "PASSAGES, ALL ALIKE"
1920 DATA "090812110000"
1930 DATA "YOU ARE IN A MAZE OF
     TWISTY"
1940 DATA "PASSAGES, ALL ALIKE"
1950 DATA "111013120000"
1960 DATA "YOU HAVF REACHED A DEAD
     END"
1970 DATA ""
1980 DATA "120000000000"
1990 DATA "YOU ARE ON THE WEST BANK
     OF THE"
2000 DATA "CHASM. A PASSAGE LEADS
     NORTH"
2010 DATA "151300000000"
2020 DATA "YOU ARE IN A LARGE CAVERN
     WITH A"
```

More next page. . .

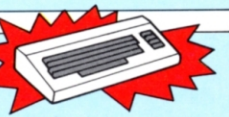
```
2030 DATA "STAIRCASE SPIRALLING
     UPWARDS"
2040 DATA "000014000016"
2050 DATA "THE WAY IS BLOCKED BY A
     ROCKFALL"
2060 DATA "A STAIRCASE SPIRALS
     DOWNWARDS"
2070 DATA "000000001500"
```

SPECTRUM ALTERATIONS

```
50 DIM V$(20,8):DIM O$(25,8):DIM
   D$(25,32):DIM G(20):DIM L(25):DIM
   M(25):DIM L$(30,2,32):DIM
   N$(30,12)
90 IF DF=1 AND RND <.6 THEN LET OB=8
   :GOSUB 1070:GOTO 150
100 LET NL=VAL N$(LC) (2*V-1 TO 2*V)
130 GOSUB G(V)
150 IF DF=1 AND RND < .2 THEN
    GOSUB 1130:GOTO 70
160 IF LC>4 AND RND < .2 AND DF=0
    THEN GOSUB 1090
500 LET N$(7) (7 TO 8) = A$
510 LET N$(14) (3 to 4) = B$
680 IF RND < .1 THEN PRINT "THE
    DWARF DODGES IT !":GOTO 290
730 LET G$=INKEY$:IF G$="" THEN
    GOTO 730
740 LET G=CODE(G$):IF G>96 THEN
    LET G=G-32
900 LET N$(3) (9 TO 10) = A$:LET
    N$(4) (11 TO 12) = B$
```

```
950 IF K$(K) <> " " THEN GOTO 940
960 LET A$ = K$ (TO K-1):LET
    LA=LEN(A$)
970 LET V=0:FOR C=1 TO NV:IF
    A$=V$(C) (TO LA) THEN LET V=C
    :LET C=99
1010 LET B$=K$ (K+1 TO):LET
     LB=LEN(B$)
1020 LET O=0:FOR C=1 TO NO:IF
     B$=O$(C) (TO LB) THEN LET O=C
     :LET C=99
1150 IF RND < .8 THEN PRINT "IT
     MISSES YOU !":RETURN
1340 READ NV:FOR K=1 TO NV:READ
     V$(K),G(K):NEXT K
1380 DATA 19
1390 DATA "NORTH",0,"EAST",0,"SOUTH"
     ,0,"WEST",0,"DOWN",0,"UP",0,
     "LIST",360,"SCORE",540,"LOOK"
     ,1190,"TAKE",210
1400 DATA "DROP",300,"WAVE",420,
     "THROW",590,"FIGHT",710,"SAY"
     ,790,"EXTINGUISH",830,"LIGHT"
     ,830,"UNLOCK",860,"LOCK",860
```

Replace INPUT X by INPUT X:PRINT X in lines 930 and 1000



COMMODORE
ALTERATIONS
==========

```
730 GET G$:IF G$="" THEN GOTO 730
```

Replace CLS by PRINT CHR$(147) in line 1190

Glossary

Array A special type of variable or string variable, which is used to store a list of numbers, letters or words in a program. An array may also be referred to as a 'table'.

ASCII Code (American Standard Code for Information Interchange) An internationally accepted code, built into most computers, which gives a code number to each capital and lowercase letter, number, punctuation mark, or symbol that can be typed into the computer keyboard.

BASIC (Beginners All-purpose Symbolic Instruction Code) The most widely used computer language for programming microcomputers.

Constant A number that expresses a fixed (unchanging) value, such as the figures 38 or 7

Data Any information which is fed into a computer to be used or processed in some way by the program. It can consist of numbers, letters, words or symbols.

Formatting To arrange for information to be displayed on the computer screen in a particular way. Most computer languages have specific commands which deal with formatting.

Menu A list of options contained within a program, which allows the user to choose which part of the program they wish to operate. The menu can be displayed on the screen for use at certain points during the program.

String ($) A specific space in the computer's memory which is set up to store letters, words, or 'strings' of words (such as a sentence). The string, or space, is given a code name so that the computer can find it again whenever it needs to put information into the string, remove the information, or replace it with new information. (See also Variable.)

Variable A specific space in the computer's memory which is set up to store a particular piece of information, which can then be changed later on in the program. A variable is given a code name and a value. The code name does not change,

but the content or value of the variable can be changed. Variables are usually given numerical values; letters and words have to be stored in a string variable.

Word Processing A system used in offices for writing, storing, and editing documents. Paper copies are produced using a printer attached to the computer.

Index

Piccolo Explorer Books

The Explorer Encyclopedia
Superbly illustrated in colour, and packed with hundreds of facts about the amazing world we live in (224 pages).

The Explorer Book of Nature
A passport to the diverse and fascinating world of nature, with beautiful colour illustrations (128 pages).

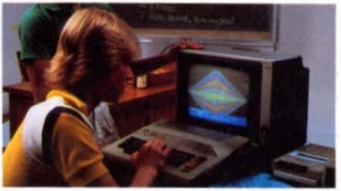
The Explorer Book of Mysteries
Some of the greatest unsolved mysteries of all time, presented in a colourful bumper volume (128 pages).

The Explorer Book of Warfare
The story of military power, from the earliest battles right up to the nuclear age. Fully illustrated (96 pages).

A PICCOLO FACTBOOK

COMPUTER PROJECTS

A range of activities to increase your programming skills.



Titles in the Factbook Series:

- DINOSAUR WORLD
- ANIMAL LIFE
- SPACE FLIGHT
- EARLY MAN
- ASTRONOMY
- AIRCRAFT
- WEAPONS AND WARFARE
- ANCIENT CIVILIZATIONS
- BIRDS
- MUSIC
- CARS AND TRUCKS
- PLANT LIFE
- PLANET EARTH
- COMPUTER WORLD
- HOME SCIENTIST
- SHIPS AND OTHER SEACRAFT
- DRAWING AND PAINTING
- ROBOT WORLD
- BIKES
- COMPUTER PROJECTS

ISBN 0-330-28517-3

9 780330 285179 90000

£1.75

COMPUTER PROJECTS

Piccolo