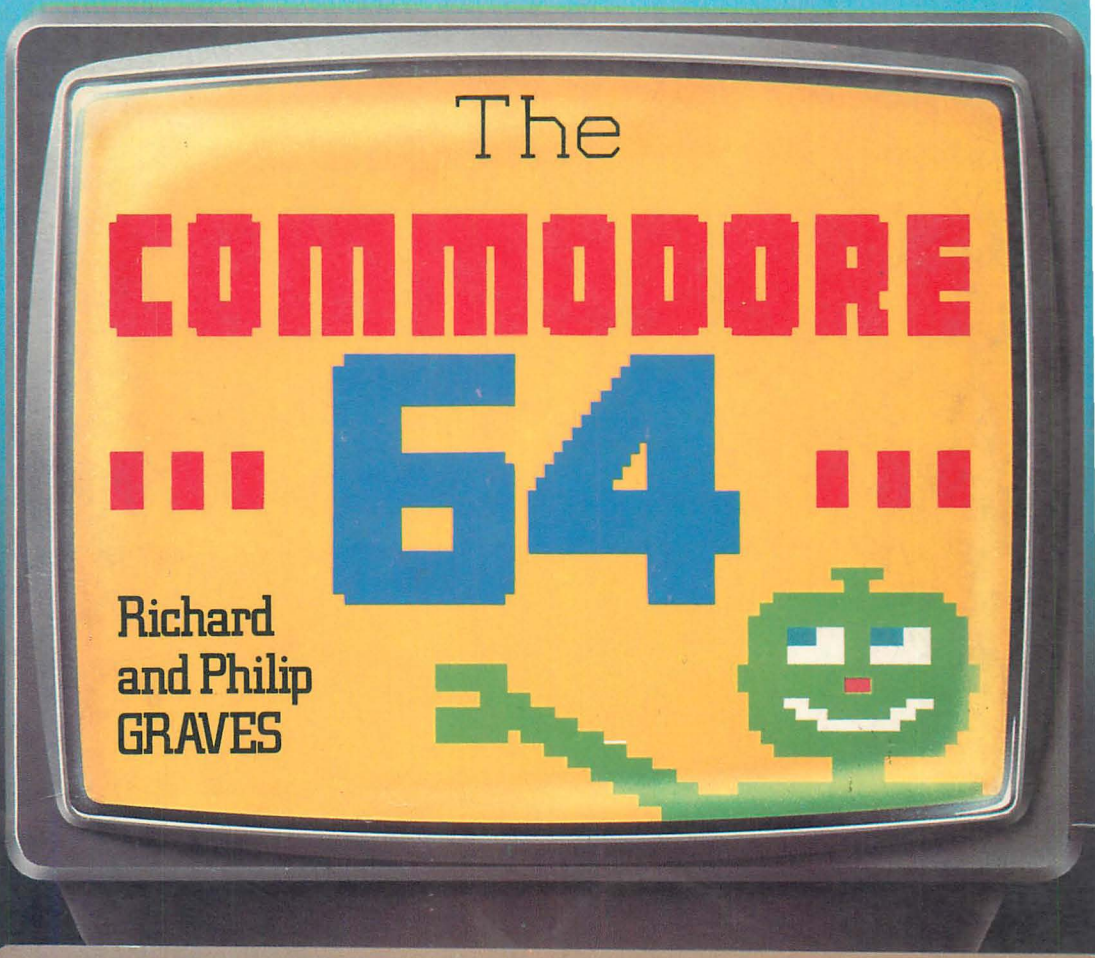
A BEGINNER'S GUIDE TO



- USING THE KEYBOARD
- SIMPLE PROGRAMMING
- GRAPHICS
- SOUND
- GAMES

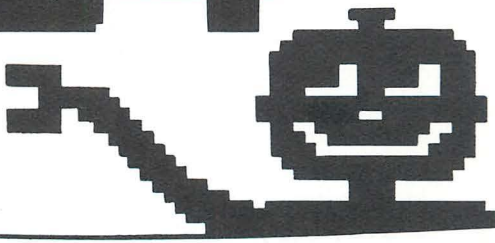
First published in 1985 by Kingfisher Books Limited,
Elsley Court, 20–22 Great Titchfield Street,
London W1P 7AD
A Grisewood & Dempsey Company

BRITISH LIBRARY CATALOGUING IN PUBLICATION DATA
Graves, Richard
A beginner's guide to the Commodore 64.
1. Commodore 64 (Computer) – Juvenile literature
I. Title II. Graves, David
001.64'04 QA76.8.C64

ISBN 0-86272-160-1

Edited by Jacqui Bailey
Design by David Jefferis
Illustrations by Eagle Artists Hayward and Martin
Typeset by Southern Positives and Negatives (SPAN)
Printed in Portugal by Printer Portuguesa, Sintra

The COMMODORE 64



Richard and Philip
GRAVES

Kingfisher Books

Contents

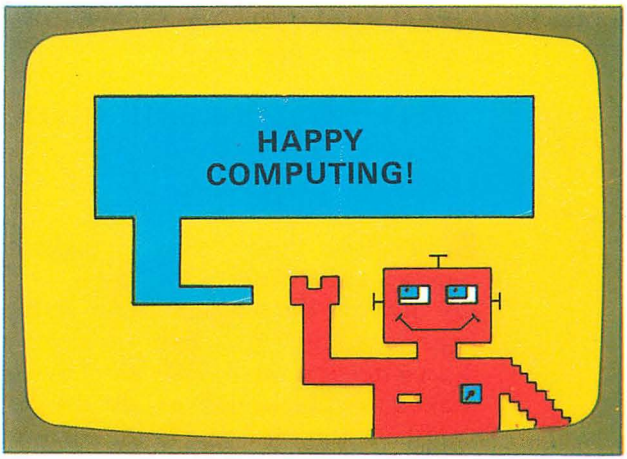
Introduction

Some months ago, our family was given the chance of trying out a Commodore 64 microcomputer. For various reasons we were all keen on the idea. One of us hoped to see how it could be used for business purposes; another looked forward to making use of its superb games-playing facilities.

We had already discovered that many of the standard books on BASIC were not only far too complicated for the younger members of the family, but fairly tough going for the older ones as well. And we soon found that the Commodore 64 *User Manual* was one of the most complicated we had seen. However, we persevered; and having learned the hard way we are keen to share our newly acquired knowledge with other beginners.

By the time you have completed this book, you will be able to write simple programs using number and string variables, sound, colour and simple graphics. We hope you will find that owning a microcomputer opens up new horizons for you, as it has done for us. We should also like to thank David Graves for his help: several of the programs in this book are based upon his ideas.

Richard and
Philip Graves



1 Making a Start

How to give a simple instruction

▲ *When you switch your computer on, you will see the word 'READY' followed by a flashing square known as the cursor.*

The computer can only do what you tell it to do. You tell it to do something by typing instructions on the keyboard. Everything that you type appears at the place on the screen where you can see a flashing white square which is called the *cursor*, and the cursor at once moves one space to the right. The cursor starts just below the word 'READY'. You will often see 'READY' appear on the screen. It simply tells you that the computer is ready for your next instruction.

Now, try typing an instruction. 'Tell me' is an instruction. How about:

TELL ME "LOOK BEHIND YOU"

? Problem 1: When you get to the " (inverted commas), you will find it on the key marked 2. But when you press the key 2 it will only type 2.

✓ Hold one finger on the SHIFT key at the bottom left (or bottom right) of the keyboard. Keep holding it down while you press 2 and the " will appear. This works for all the symbols shown on top of the other number keys, and for the symbols on top of the other symbol keys. Now type in your instruction. Be careful, you might type:

TELL MO" LOOKBE HINDYOU"

by mistake.

? Problem 2: It is easy to make mistakes when typing an instruction.

✓ Look at the top right of the keyboard. Find the key marked INST/DEL. Every time you press it, it rubs out a letter or symbol to the left of the cursor. So if you make a mistake, press INST/DEL until the mistake has vanished, and then type in the correct instruction.

When you have typed in an instruction nothing will happen until you press the key marked RETURN,

which is near the bottom right of the keyboard. So, press RETURN. Now look at the screen. Why does it say (above the word 'READY'):

```
?SYNTAX ERROR
```

? Problem 3: The computer only understands instructions when they are typed in a special language called BASIC. It does not know what 'Tell me' means.

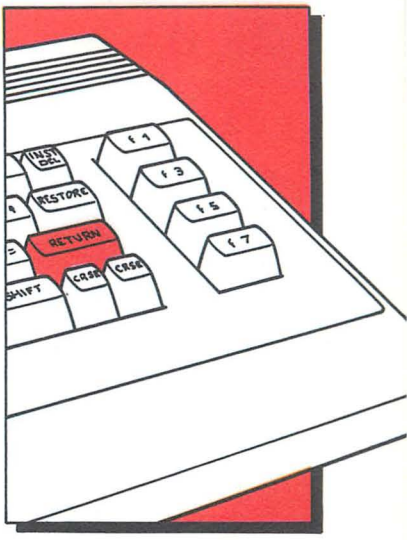
✓ Luckily, instructions in BASIC are not very difficult to learn. PRINT is a useful instruction in BASIC. If you want the computer to print a message on the screen, you type PRINT. Then you type " followed by the message, and then another " . So type:

```
PRINT  "LOOK BEHIND YOU"
```

and press RETURN. Now you can read on the screen:

```
LOOK BEHIND YOU
```

You have now learned how to give the computer a simple instruction. Try some more instructions of your own using the PRINT instruction. For example, tell the computer to PRINT the name of a country which you would like to visit.



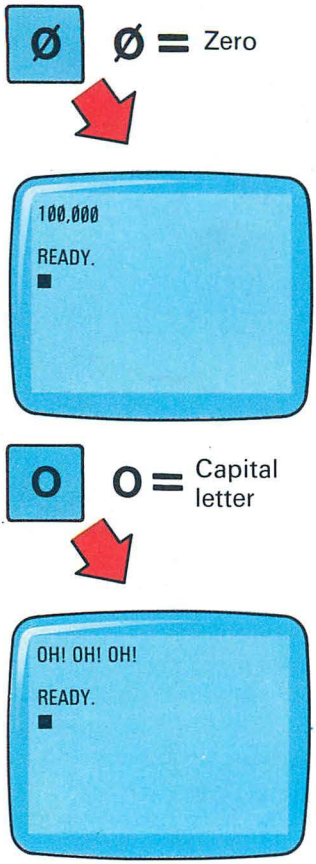
▲ *Each time you type in an instruction you must press the key marked 'RETURN' before the computer will do anything.*

How to give a list of instructions

One instruction on its own is rather boring. If you want the computer to do something more interesting you need to give it a list of instructions.

? Problem 4: The computer obeys every instruction as soon as you press RETURN. How can you make it wait until you have given it a list of instructions?

✓ Type a number before each instruction, or line. Then, when you press RETURN, the computer will store each numbered line in its memory. Start with 10 for the first line, 20 for the second, and so on. You will find the 0 (zero) figure to the right of the 9. It is



marked Ø. Do not muddle it up with the O, which is just the capital of the letter o. From now on, all zeros in this book will be shown as Ø to help you to remember.

Now, type in the list of instructions shown below *and remember to press* RETURN *after each completed instruction line*. Do not worry if you are typing an instruction which will not all fit onto one line of the screen. *Do not* press RETURN when there is no room left on the line – just carry on typing. Some of the words will go over onto the next line on the screen, but the computer will treat it all as one line because you have not yet pressed RETURN.

The ! (exclamation mark), (single inverted comma) and ? (question mark) are on top of the 1, 7 and / keys, so you will need to use the SHIFT key when you type them.

```
1Ø  PRINT  "I AM A COMMODORE 64
MICROCOMPUTER."
2Ø  PRINT  "I HOPE YOU ENJOY USING
ME."
3Ø  PRINT  "DO YOU KNOW WHAT
BASIC MEANS?"
4Ø  PRINT  "BEGINNERS' ALL-PURPOSE"
5Ø  PRINT  "SYMBOLIC INSTRUCTION
CODE!"
6Ø  END
```

The instruction END at line 6Ø tells the computer that the list of instructions has come to an end.

? Problem 5: Now that you have typed in a list of instructions, how do you make the computer obey the instructions?

✓ Type the instruction RUN and press RETURN. You should now see on the screen all the words that you typed between the inverted commas. However, something may have gone wrong. Where one of the lines, such as:

```
I HOPE YOU ENJOY USING ME
```

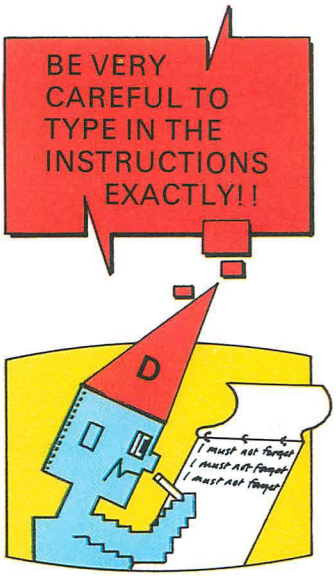
should have appeared, you may see only:

```
Ø
```

followed by a message saying:

```
?SYNTAX ERROR IN
```

followed by the number of a line. Or, you may get the 'SYNTAX ERROR' message on its own.



? Problem 6: The computer is not obeying your instructions.

✓ Look very carefully at the list of instructions which you have typed in. If you had the 'SYNTAX ERROR' message on its own you have made a typing mistake in the line the computer is pointing out to you. If you had 'Ø' where a line should have been, you probably left out the first set of inverted commas. Whenever you type out lists of instructions you must be very careful to copy them out exactly. Make sure that even the spaces between words and symbols are in the right place.

If you do find that you have made a mistake, simply type the whole line out again, including the line number, and press RETURN. Then type RUN again. The computer will automatically replace the old line with the new line that you have typed, and will obey your instructions properly. A list of instructions is called a *program*. Now you have entered your first program and run it. To run it again, simply type RUN again and press RETURN.

? Problem 7: How can you remove a program from the computer's memory without switching off the computer?

✓ Type NEW and press RETURN. Do this, and write a program of your own using the PRINT instruction. For example, you could write a program which will print the names of five wild animals.

Three programs to show what the computer can do

Type out these programs and see what happens.

They are to show you what the computer can do. Remember to be very careful to copy them exactly.

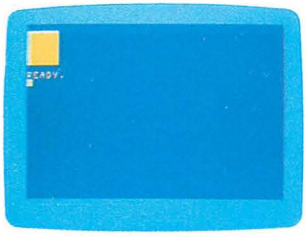
? Problem 8: In line 1Ø of the first two programs you will see the instruction:

```
PRINT  "(CLR/HOME)"
```

This instruction does not remove anything from the computer's memory, but it rubs out everything on the screen, or CLeaRs it, so that when the program is run it has a blank screen on which to begin. But how do you type it?

✓ After typing the first " (inverted commas) you should keep one finger on the SHIFT key while pressing the CLR/HOME key, which is near the top right of the keyboard. A heart shape will appear on the screen. Then you type the second " and go on as normal. (And if you want to clear the screen at any time while you are using the computer, simply keep one finger on the SHIFT key while pressing the CLR/HOME key.)

In line 3Ø of the first program, the words "(5 spaces)" simply mean that you should type in five blank spaces between the inverted commas.



▲ *When you type RUN your screen should look like this. If the square does not appear then you have made a typing error.*

ONE

```
1Ø  PRINT  "(CLR/HOME)"
2Ø  FOR  A=1 TO 5
3Ø  PRINT  CHR$(18);CHR$(158);"(5
spaces)"
4Ø  NEXT  A
5Ø  END
```

Now run the program. If you have typed in the lines correctly, you will see a yellow square near the top left of the screen.

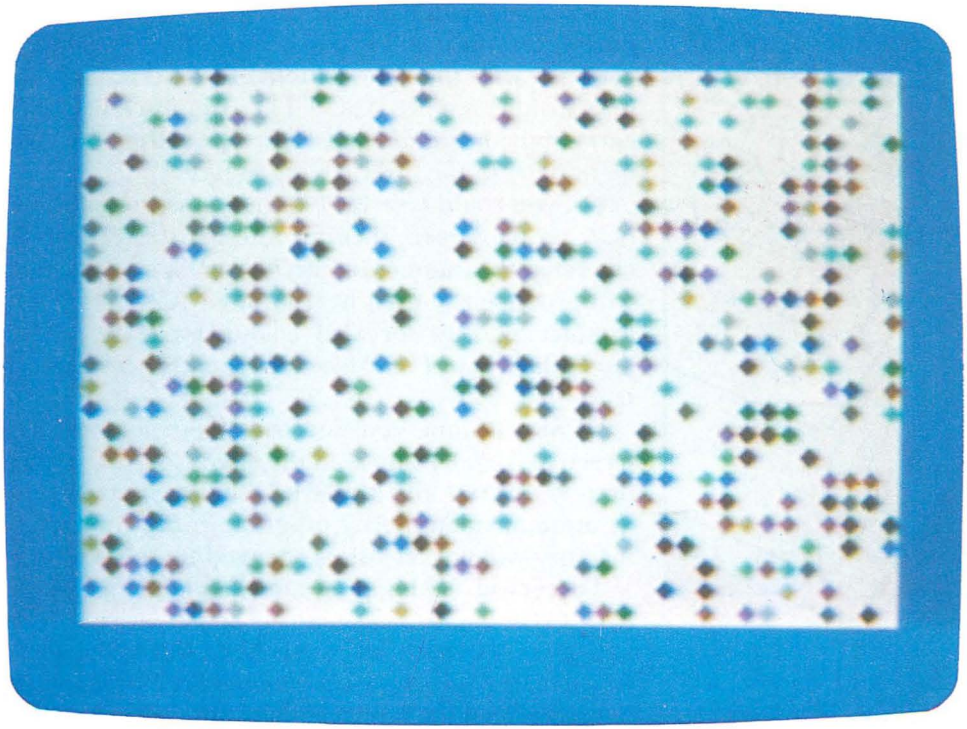
? Problem 9: When you run the program, if the yellow square does not appear then you have probably typed in some program lines wrongly. But your list of instructions has disappeared. How can you get the list back again?

☑ Type the instruction LIST and press RETURN. This shows on the screen all the numbered lines at present in the computer's memory, in numerical order. Now you can check to see whether you have made any mistakes and, if so, type in the correct lines in the normal way.

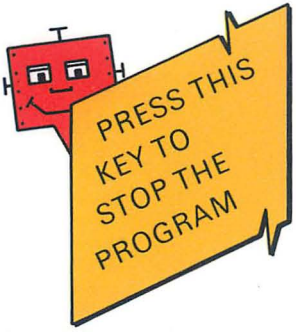
TWO

```
10  PRINT  "(CLR/HOME)"
20  POKE  53281,1
30  LET  A=INT(2023*RND(1))+1
40  IF  A<1024 THEN GOTO 30
50  LET  B=A+54272
60  LET  C=INT(15*RND(1))+1
70  POKE  A,90
80  POKE  B,C
90  GOTO  30
100 END
```

▼ *Because of the instruction GOTO at line 90, this program will go round in a loop forever.*



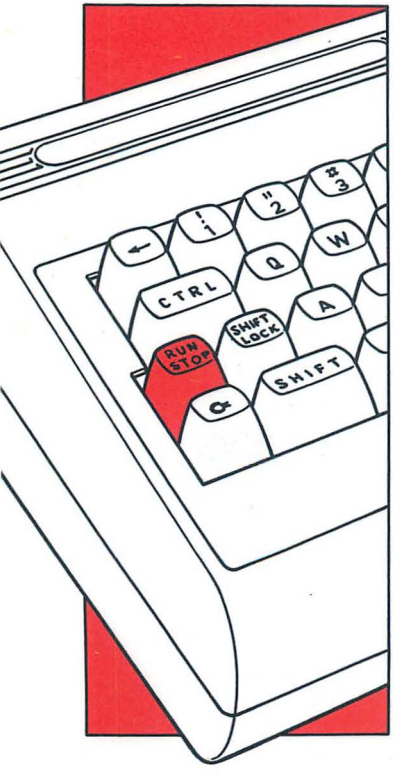
This program will print an endless series of coloured diamonds on a white background, surrounded by a blue border. The program goes on and on because when it reaches line 9Ø the computer reads the instruction GOTO 3Ø. This tells it to jump at once to line 3Ø, and so it goes round and round in circles. It never reaches line 1ØØ. This is known as making a *loop*.



? Problem 1Ø: How can you stop the program running?

✓ Press the RUN/STOP key which is near the bottom left of the computer. At once, the program will stop running and a message will appear above the word 'READY', saying something like:

```
BREAK IN 7Ø
```



The number 7Ø in this message is the number of the line which the computer had read and obeyed just before you broke into the program. It could have been one of the other line numbers.

Now you can run the program again, and a different pattern will gradually appear. Or, you can continue with the same pattern. To do this, you type CONT and press RETURN. The computer will then carry on, beginning at the line after the one mentioned in the 'BREAK' message.

Or, you could type LIST to get your program lines back on the screen, then try altering the pattern by changing the number in line 7Ø from 9Ø to 83 (which will produce coloured hearts) or to 1Ø2 (which will produce slightly speckled coloured squares). (See page 133 of your *User Manual* for some other shapes to try.)

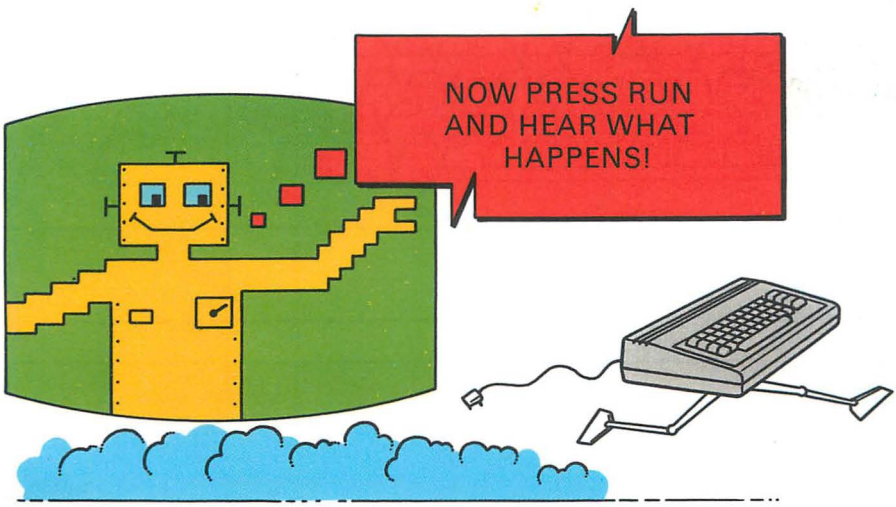
(If at any time you find that the colours on the screen make it hard to read the words (or 'text'), then press RUN/STOP and RESTORE to return the screen to its original colours. Any program in the computer's memory will not be lost or changed by this action.)

To get rid of this program altogether, press the RUN/STOP key to break into it and then type NEW and press RETURN as usual. You will still have a white central screen with a blue border, but if you are happy to keep the screen like this simply press SHIFT

and CLR/HOME to rub out any remaining text or graphics (pictures) from the previous program. If you want to return to the original screen colours, then press RUN/STOP and RESTORE instead.

THREE

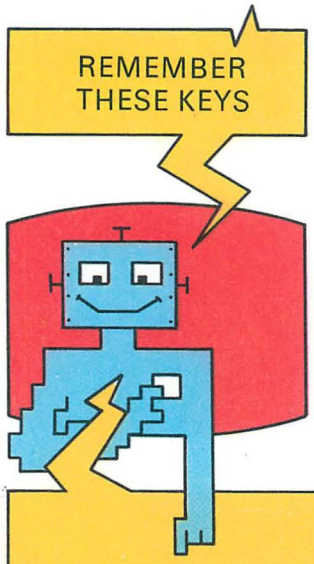
```
 1Ø  POKE  54296,12
 2Ø  POKE  54277,15Ø
 3Ø  POKE  54278,128
 4Ø  FOR  S=1 TO 15
 5Ø  READ  A,B
 6Ø  POKE  54273,A
 7Ø  POKE  54272,B
 8Ø  POKE  54276,17
 9Ø  FOR  T=1 TO 3ØØ
1ØØ  NEXT  T
11Ø  NEXT  S
12Ø  POKE  54276,Ø
13Ø  POKE  54277,Ø
14Ø  POKE  54278,Ø
15Ø  END
16Ø  DATA  25,177,21,154,17,37,28,214,25,
177,28,214,25,177
17Ø  DATA  28,214,22,227,19,63,16,47,28,214,
25,177,28,214,25,177
```

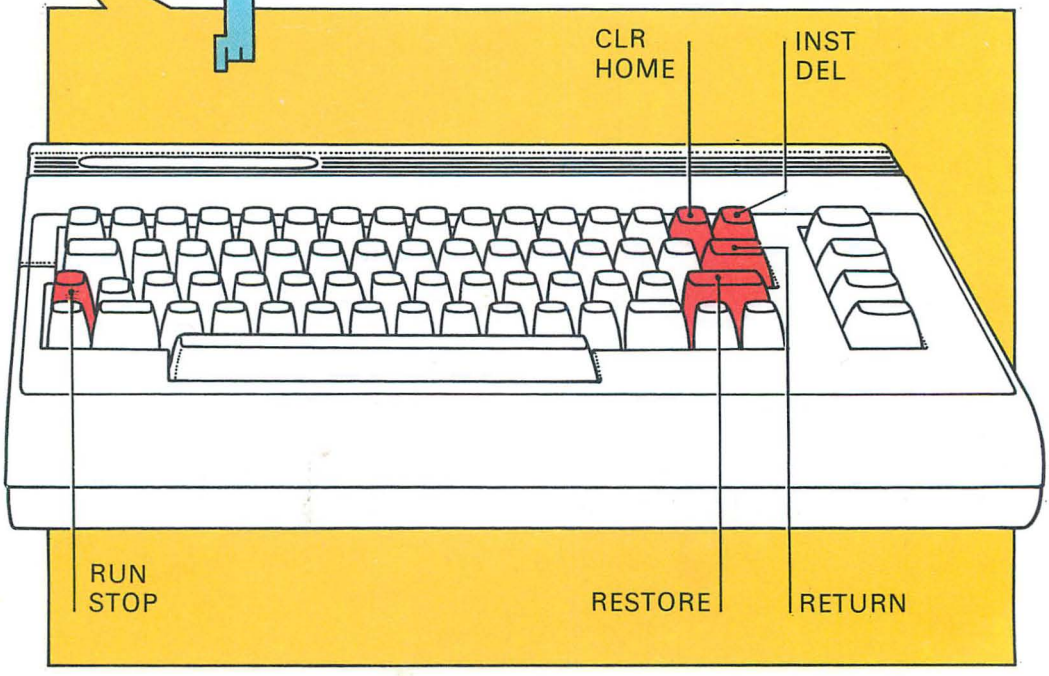


As you will hear when you run this program, the computer can also make sounds!

A reminder

You have now learned seven instructions which have to be typed in:
CONT, END, GOTO, LIST, NEW, PRINT, and RUN.
You have also learned five instructions which can be keyed in:
CLR/HOME, INST/DEL, RETURN, RUN/STOP, and RUN/STOP with RESTORE.
You have also learned how to write a simple program using the PRINT instruction, and how to make the computer RUN it.
(You will learn more about the instructions CHR$, DATA, FOR, IF, INT, LET, NEXT, POKE, READ, RND, THEN and TO later on in the book.)





2 Making the Computer Work for You

Simple mathematics

First of all, find the + (plus) and − (minus) signs. They are on the top row of the keyboard, to the right of the number keys.

Now, here is a simple sum: 3+4. If you type this on the computer and press RETURN, the computer will not understand what you mean. But if you give the correct instruction, the computer will do the sum very quickly and print the answer on the screen. Try typing:

```
PRINT  3+4
```

and press RETURN. The answer appeared at once. But then you could have worked it out in your head just as quickly! Here is a more difficult sum:

4×89898989

? Problem 11: It was easy to find the + and − signs, but where are the × (times) and ÷ (divided by) signs?

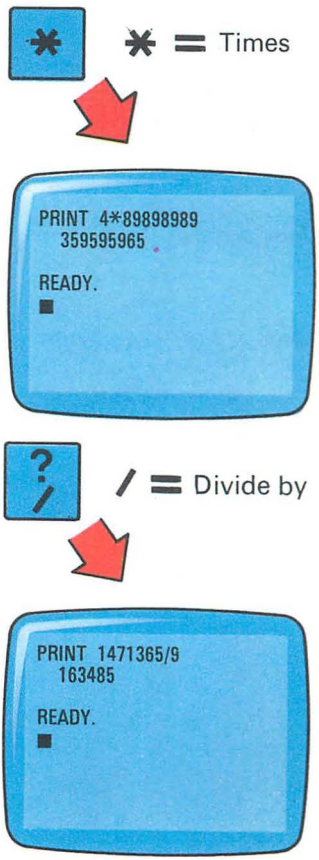
✓ The sign for × (times) is * and it is on a key near the top right of the keyboard. The sign for ÷ (divided by) is / and it is on the bottom right of the keyboard. Now type:

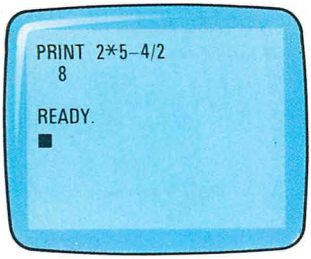
```
PRINT  4*89898989
```

and press RETURN. In a flash, the answer appears: 359595956. Try out these sums:

5×12345678
$1471365 \div 9$

If you have a sum with more than one part, such as





▲ *If you type in a sum with more than one part, the computer will work out the times and divided bys first.*

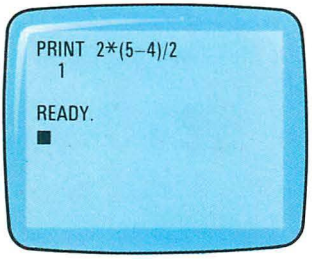
$2 \times 5 - 4 \div 2$, the computer will work out the times and divided bys first. So it would work out that $2 \times 5 = 10$, then $4 \div 2 = 2$, then $10 - 2 = 8$, so the answer is 8. But if you place parts of a long sum inside brackets, which are at the top of keys 8 and 9, it will work out the parts inside the brackets first. So $2 \times (5 - 4) \div 2$ would be worked out as $5 - 4 = 1$, then $2 \times 1 = 2$, then $2 \div 2 = 1$, so the answer is 1. Try checking this out by typing the following statements and pressing RETURN after each one:

```
PRINT  2*5-4/2
PRINT  2*(5-4)/2
```

To put a decimal point in a sum, you can use the full-stop key (two keys to the right of the M key). Try:

```
PRINT  35*7.25
```

and press RETURN. The answer should be 253.75. Now try out some sums of your own. For example, divide the number of pupils in your school by the number of teachers to find out how many pupils there are to each teacher.



▲ *But if you put part of the sum in brackets, it will work out whatever is inside the brackets first.*

Number variables

(Doing the same type of sum, with different numbers each time.)

Imagine that it will soon be your birthday. You have asked everyone for money. You have decided to spend a quarter of what you are given on your stamp collection; to save a quarter; and to spend half on some pop records. These amounts are *fixed*; but you have no idea how much money you will be given. It might be anything from £1 to £50, or even more. Because the number of pounds is *not fixed* it is called a *variable*.

In the program which follows, you may call this *number variable* by the letter A. (And you keep it *outside* the inverted commas of the PRINT statements.) You will find the £ sign near the top right of the keyboard:

```
1Ø  PRINT  "I AM GIVEN £"A
2Ø  PRINT  "FOR STAMPS I HAVE £"A/4
3Ø  PRINT  "FOR SAVING I HAVE £"A/4
4Ø  PRINT  "FOR RECORDS I HAVE
£"A/2
5Ø  END
```

But if you run this program, the computer will print on the screen:

```
I AM GIVEN £ Ø
```

and so on. You have put in the variable A, but you have not told the computer what A is! Or, to put it another way, you have not defined A. You could guess that you will get £2Ø for your birthday, and add the line:

```
7  LET  A=2Ø
```

(Now that you want to add an extra line, you can see why you have become used to numbering lines in tens. There is lots of space left to put in new lines.)

If you run the program now, the computer will print:

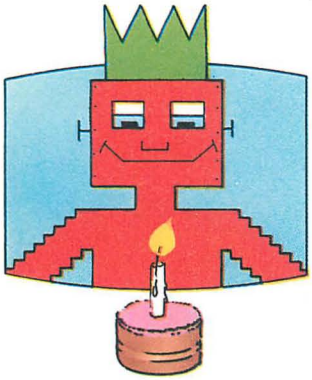
```
I AM GIVEN £ 2Ø
FOR STAMPS I HAVE £ 5
FOR SAVING I HAVE £ 5
FOR RECORDS I HAVE £ 1Ø
```

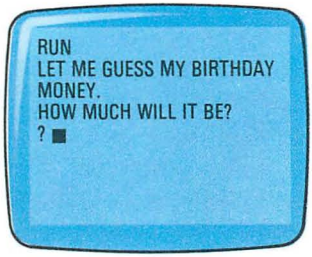
But with this method, you will have to type a new line 7 each time you guess a new amount of money! So you had better delete (rub out) line 7 and do something else.

▲ *To make the computer tell you how many pounds you have for each item, you must first define what the variable (the letter A) at line 1Ø stands for.*

? Problem 12: How can you delete a line in a program when it has already been typed in, and you have pressed RETURN?

✓ Supposing that you wish to delete line 7. If you wish to delete it and put nothing in its place simply type 7 and press RETURN. If you wish to put something in its place, type 7, and the rest of the new line, and press RETURN. The computer will





▲ *By putting an INPUT instruction into the program, you make the computer stop and wait until you have typed in the amount of money you think you will have. It will then work out how much you have to spend on each item.*

immediately delete the old line 7 and put the new one in its place.

Now, try deleting line 7, and check to see that it has disappeared by typing LIST and pressing RETURN. Here are three new lines to enter:

```
5  PRINT  "LET ME GUESS MY
BIRTHDAY MONEY."
6  PRINT  "HOW MUCH WILL IT BE?"
7  INPUT  A
```

Try running the program. The computer will print:

```
LET ME GUESS MY BIRTHDAY MONEY.
HOW MUCH WILL IT BE?
?
```

Then the computer will wait and do nothing, because of your instruction INPUT A. The question-mark on the third line is telling you that the computer is waiting for you to put in, or *input*, whatever number you have decided to place in the store labelled A. The computer will not continue with the program until you have made an input. Try inputting 2Ø (simply type 2Ø and press RETURN). The computer will at once go on with the rest of the program and will give you the answers that you had before. (However, the computer is waiting for a number to be input, so if by mistake you type a letter or symbol it will give you the error message:

```
?REDO FROM START
```

and you must type your guess again.)

Now type RUN again and press RETURN. This time, input the number 32. The answers will, of course, be different. In fact, you can now run this program again and again, guessing a different sum of money each time, and seeing how much you will have for spending and how much for saving.

❓ Problem 13: Your program does not have a title.
☑ To give any program a title you enter a line begining with the word REM, followed by " followed

by a title of no more than 16 letters, numbers or symbols, followed by ". For this program, the title could be BIRTHDAY, in which case you should add:

```
3 REM "BIRTHDAY"
```

The computer takes no action when it reads the word REM; so REM lines are useful for putting lines into programs to remind yourself what is happening when you come to look at a program days or months after first writing it. This program is now complete and working well. But before you type NEW and move on to something else, you can use the program to find out the answer to another problem.

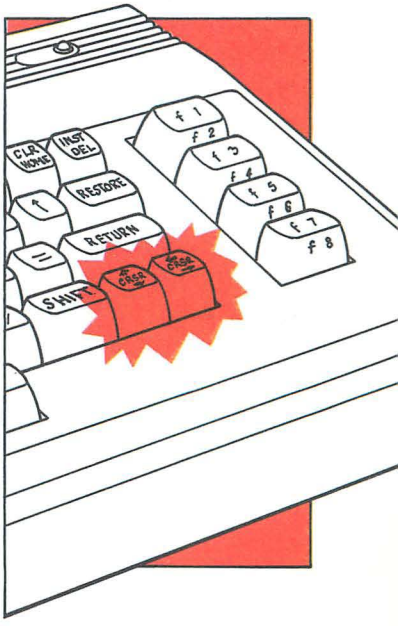
? Problem 14: How can you make a small change to a line when you have already pressed RETURN, and you do not want to have to type the whole line again? Suppose, for example, that you want to see what will happen if you insert a comma into line 10 just before the variable A.

✓ Type LIST and press RETURN so that you can see the "BIRTHDAY" program on the screen in front of you. Now look at the bottom right of the keyboard, where you will see two CRSR keys. You can use these keys to move the flashing CuRSoR around the screen.

As you can see, the left-hand CRSR key has arrows pointing up and down. If you press it on its own, the cursor will move down one space; if you press it while keeping one finger on the SHIFT key, the cursor will go up one space. In the same way, the right-hand CRSR key has arrows pointing to left and right. If you press this key on its own, the cursor will move one space to the right; if you press it while keeping one finger on the SHIFT key, the cursor will move one space to the left.

Practise moving the cursor around the screen. Now, move the cursor until it is in the same place as the letter A at the end of line 10. At the point where you place the flashing cursor in a line, you can replace what is there with another letter or symbol simply by typing it. Or, you can DELete anything to the left of it in the normal way, using the INST/DEL key on its

▼ *To help you to make changes or corrections to a program you can use these two keys, with the SHIFT key, to move the cursor around on the screen.*



own. Or, you can INSerT something new into the line, using the INST/DEL key together with the SHIFT key.

You want to insert a comma into line 1Ø just before the A. So keep one finger on the SHIFT key while you press the INST/DEL key once. Immediately, the letter A moves one space to the right. You have inserted a space into line 1Ø and you can now fill this space with a comma.

Now you have changed or 'edited' the line. However, the flashing cursor is still in the line, and before you type another instruction (or move the cursor to edit another line) you should press RETURN to tell the computer to store the new line 1Ø in its memory. Now press RETURN a few more times, until the cursor is back on an empty line again. (And do not worry if an error message appears in the meantime. This often happens when you are moving the cursor back to an empty line.) When you have done that, run the program again.

▲ *By inserting a comma just before the variable in line 1Ø, you have made a large gap appear between the pound sign and the amount of money in that line. When used in this way, some punctuation marks can affect the appearance of a program when it is run on screen.*

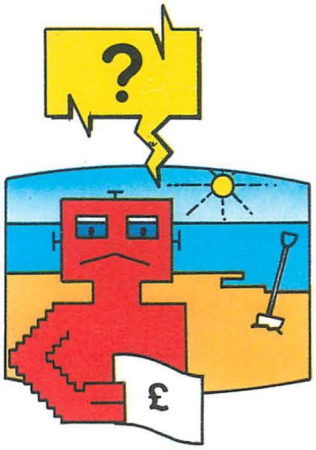
Suppose that the number you enter this time is 35. When you have entered this number you will see that in the line which begins 'I AM GIVEN', the figure 35 appears a long way to the right of the £ sign. When you use a comma as part of an instruction line it will space things out. Putting in no punctuation at all worked quite well with this particular program, but this is not always the case. It is much safer to get into the habit of using punctuation to position the information that you want to see in your PRINT statements so that it can be read clearly when the program is run.

You will find that if you use a semi-colon instead of a comma, the figure 35 will appear alongside the £ sign again. Now practise editing by changing the comma in line 1Ø to a semi-colon, and inserting semi-colons before the number variables A/4 and A/2 in lines 2Ø to 4Ø.

(It is worth remembering that the CRSR keys can also be used to edit a line before it has been entered. This can save a lot of time spent deleting and retyping if you make a mistake very early in a line, which you do not notice until you have typed in the rest of the line.)

Other number variables

Now you can try writing a more complicated program using number variables. Although you usually see programs written out in full, with explanations afterwards, programs do not suddenly come into your head fully worked out! Most programs begin with just a few lines, and then more and more lines are gradually added on.



The first thing to do is to decide what sort of game you want to invent, or what sort of problem you want to solve. For example, perhaps you would like a program which helps you to plan your holiday spending. You might want to know how much money you will have for your holiday, and how much you can afford to spend each day. To begin with, you must decide what information needs to be input into the computer. In this case, it will need to know:

a) how much money (if any) you have already;
b) how much money (if any) you will get during the holiday, as pocket-money;
c) how much money (if any) you want to have left at the end of the holiday;
d) how many days you will be on holiday.

Next, you must decide how to input this into the computer. Try these lines, and remember to copy them carefully, typing in all the inverted commas exactly as shown:

```
 1Ø  PRINT  "ENTER THE NUMBER OF
£S YOU ALREADY HAVE"
 2Ø  INPUT  A
 3Ø  PRINT  "ENTER THE POCKET-
MONEY YOU WILL GET"
 4Ø  INPUT  B
 5Ø  PRINT  "HOW MUCH MONEY DO
YOU WANT TO HAVE LEFT"
 6Ø  PRINT  "AT THE END OF THE
HOLIDAY?"
 7Ø  INPUT  C
 8Ø  PRINT  "ENTER THE NUMBER OF
DAYS YOU WILL BE"
 9Ø  PRINT  "ON HOLIDAY"
1ØØ  INPUT  D
```

The computer now has all the information it needs. But you must tell it what to do with that information. To find out how much you have to spend, you must instruct it to add the money you already have (A) to the money you will get (B), and then take away from A+B the money you want to have left (C). So you could add the line:

```
11Ø  PRINT  "YOU HAVE £";A+B-C;
" TO SPEND"
```

Notice that there is one semi-colon between the second inverted commas and the number variable A+B−C, and a second semi-colon after the number variable and before the third inverted commas. You could have used commas in both these places instead, but this would have left large gaps when the program was run.

However, you do want a gap between the number of pounds and the word 'TO' in the second part of the PRINT statement. So you should have pressed the Space Bar once, immediately after typing the third inverted commas.

Always remember that when you are typing program lines it is important to copy them exactly, and particularly to put in all the spaces correctly. If you miss a space you will soon see your mistake when you run the program and you will then have to edit the line which was wrong. From now on, to help you to type in the lines correctly, the number of spaces that must be put into an instruction line will be shown as a dash, like this _ . One dash simply means put in one space.

Now you need to add some more lines to this program. To find out how much you can spend each day, you must instruct the computer to divide the amount you have to spend (A+B−C) by the total number of days you are on holiday (D). So add the lines:

```
12Ø  PRINT_"YOU_WILL_HAVE_
£";(A+B-C)/D
13Ø  PRINT_"TO_SPEND_EACH_
DAY."
```

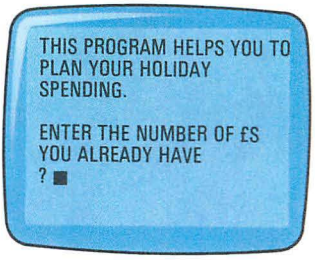
(Line 13Ø does not need a space between the first inverted commas and the word 'TO' because it is a new program line. So this line will automatically appear as a separate program line on the screen, below the words given in line 12Ø.) This program is now ready to be run; but it has no beginning or end. You can give it a title and an end with the lines:

```
  3 REM _"HOLIDAY£S"
14Ø END
```

Perhaps it would also be a good idea to explain to whoever is running the program exactly what you are trying to do. So add the lines:

```
6 PRINT _"THIS _PROGRAM _HELPS _
YOU _TO"
7 PRINT _"PLAN _YOUR _HOLIDAY _
SPENDING."
```

and run the program.



▲ *You can use the PRINT instruction on its own to put blank lines on the screen, so that your program will look better when it is being run.*

? Problem 15: The program is working very well, but you could improve its appearance. To begin with, it does not look very good on the screen because when the program is run it follows on at the end of the list of instructions.

✓ Use the CLR/HOME instruction that you learned in chapter one, and add the line:

```
5 PRINT _"(CLR/HOME)"
```

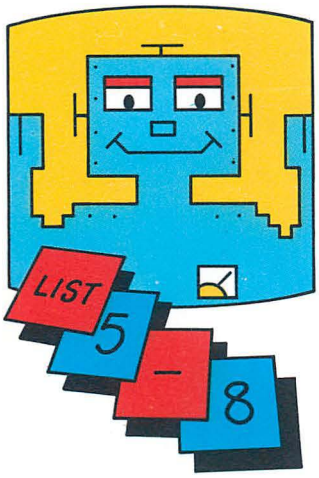
Remember that (CLR/HOME) means simply that you press the CLR/HOME key once, while keeping your finger pressed on the SHIFT key.

You could do still more. A gap between the two lines telling you what the program is about, and the two lines in which the computer asks for input, would look clearer. So would a small gap between the lines in which the computer asks for input, and the lines in which it gives answers; and also a small gap between the two answers.

To leave a blank line on the screen you simply type PRINT with nothing after it. So add:

```
 8 PRINT
 9 PRINT
1Ø5 PRINT
115 PRINT
```

Now, look very carefully at your list of instructions, and try to follow what each line will do when you run the program.



? Problem 16: The program is now so long that you cannot see all of it on the screen! When you type LIST and press RETURN, the first lines whizz past and disappear off the top of the screen.

✓ You can also use the LIST instruction to show you certain parts of a program. If you want to see just one of the lines that are hidden simply type LIST followed by the number of that line. For example:

```
LIST _6
```

If you want to see a group of lines that are hidden, type LIST followed by the numbers of the first and last lines that you wish to see on the screen, separated by a hyphen (use the minus symbol for this). For example:

```
LIST _5–8
```

You can then add to or edit these lines in the usual way.

Now, back to the program. It looks quite good when you run it, but programmers are hardly ever satisfied!

? Problem 17: How could you make this program more interesting and useful?

✓ By using two IF...THEN statements. Why? Because at present, if someone enters, for example, 25 at INPUT A (line 2Ø) and 27 at INPUT B (line 4Ø), they might also enter 8Ø at INPUT C (line 7Ø). Then the computer would print at line 11Ø:

```
YOU HAVE £–28 TO SPEND
```

That would be nonsense! By using IF...THEN

statements you can prevent something like that from happening. Add these two lines:

```
74  IF _A+B-C <=Ø _THEN _PRINT _
THAT'S _SILLY! _YOU _CAN'T _KEEP _
AS _MUCH _AS _THAT."
75  IF _A+B-C <=Ø _THEN _GOTO _5Ø
```

You will find the < (is less than) sign on the top of the comma key; and the = (equals) sign is next to the RETURN key.

Line 74 tells the computer that IF the amount of money left at the end of the holiday is £Ø, or less than £Ø, THEN it must print a message explaining to the person running the program that they have entered too large a number at INPUT C – in other words, he or she has asked for too much money to be left at the end of the holiday.

Line 75 tells the computer that IF the amount of money left at the end of the holiday is £Ø, or less than £Ø, THEN it must jump back to line 5Ø. This will give the person running the program another chance to enter a sensible number at INPUT C.

After typing in these new lines, type LIST and look at the program carefully so that you can see what is happening at every line. Then run it.

String variables

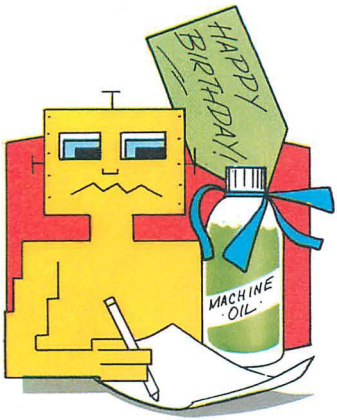
(Giving the same set of instructions, with some different words input each time.)

For variable numbers, you used A, B, C and D. You could also have used A1, A2, A3 and so on, up to A9; and all the other letters of the alphabet up to Z9. For variable words or groups of letters *(string variables)* you can do just the same, except that after, for example, the letter A or the letter and number A1, you must also use the $ (string) sign. This symbol is on the top row of the keyboard, on the 4 key.

Here is a program which you could use to write thank-you letters. It uses the string variables A$, B$, C$, D$, E$, F$ and G$.

At line 22Ø of the program you should type in your

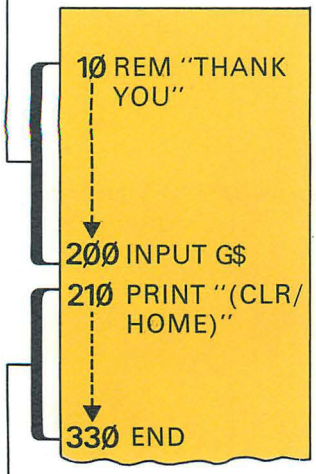
▲ *String variables can be used to enter different words or letters each time you run the program.*



own address, and not 3, POWYS AVENUE.)

```
 1Ø  REM "THANK YOU"
 2Ø  PRINT "(CLR/HOME)"
 3Ø  PRINT "THIS PROGRAM
HELPS YOU TO WRITE"
 4Ø  PRINT "A THANK-YOU
LETTER."
 5Ø  PRINT
 6Ø  PRINT
 7Ø  PRINT "ENTER YOUR
PRESENT. IT WAS A"
 8Ø  INPUT A$
 9Ø  PRINT "CHOOSE A WORD
TO DESCRIBE YOUR ";A$
1ØØ  INPUT B$
11Ø  PRINT "WHO GAVE YOU
THE ";A$
12Ø  INPUT C$
13Ø  PRINT "CHOOSE A WORD
OR WORDS TO DESCRIBE THE
WEATHER."
14Ø  INPUT D$
15Ø  PRINT "CHOOSE A WORD
OR WORDS TO DESCRIBE YOUR
FAMILY."
16Ø  INPUT E$
17Ø  PRINT "TYPE YOUR
SIGNATURE"
18Ø  INPUT F$
19Ø  PRINT "WHAT IS THE DATE?"
2ØØ  INPUT  G$
21Ø  PRINT "(CLR/HOME)"
22Ø  PRINT "(2Ø spaces)3, POWYS
AVENUE."
23Ø  PRINT "(2Ø spaces)";G$
24Ø  PRINT
25Ø  PRINT "(2 spaces)DEAR ";C$
26Ø  PRINT "(4 spaces)THANK YOU
VERY MUCH FOR THE ";A$;" IT
IS REALLY ";B$;".";
27Ø  PRINT " IN FACT IT IS
THE BEST ";A$;" I HAVE EVER
HAD."
```

These lines gather information for the letter



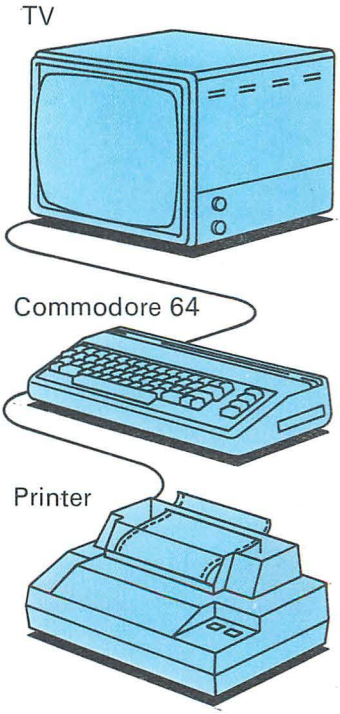
These lines print the letter on the screen

```
28Ø  PRINT _"(4 spaces)I _HOPE _YOU _
HAVE _BEEN _HAVING _GOOD _
WEATHER. _THE _WEATHER _HERE _";
29Ø  PRINT _"HAS _BEEN _";D$;". _
YOU _WILL _BE _GLAD _TO _HEAR _
THAT _THE _FAMILY _ARE _";
3ØØ  PRINT _E$;". _THANK _YOU _
AGAIN _FOR _THE _";A$;"."
31Ø  PRINT _"(6 spaces)WITH _LOVE"
32Ø  PRINT _"(6 spaces)FROM _";F$
33Ø  END
```

As you can see, the lines up to 2ØØ are gathering information, and the lines from 21Ø to 33Ø are printing the letter on the screen. The spaces inside the inverted commas are put in to make your letter look better when it is on the screen. If you had a printer, you could run the program, then print the letter onto a sheet of paper, and then run it again, inputting new string variables to write your next letter. Even without a printer you can enjoy running this program with different variables. They can be sensible or silly! The computer will print whatever you tell it to print. Try running the program a few times. Then make any changes which you feel will improve it.

Perhaps you could make what appears on the screen look better. For example, you will notice that when you are asked to choose a word or words to describe the weather, the 'W' of the word 'WEATHER' appears on one line, and the rest of the word on the next line. You could make it all appear on one line by editing line 13Ø and putting in another space before WEATHER.

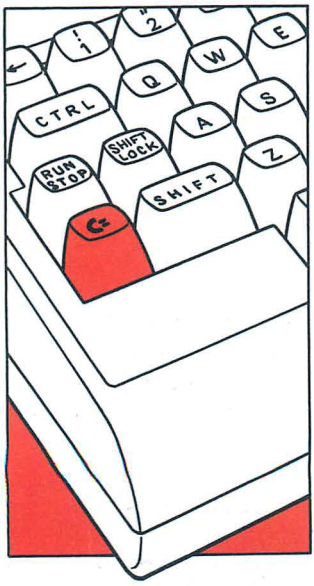
Also, it is rather a short letter. Perhaps you would like to include a paragraph in which you mention a book that you have been reading, or a television programme which you have been watching, and say what you think of it. To do this, you must work out the form of the sentences, such as 'I have been watching a television programme called ... It stars ... and I am finding it extremely ... I wonder what you think of it?' Then you must add the lines to INPUT the information that will be needed; and a line or lines to PRINT the extra sentences in the letter.



▲ *By connecting a printer to your computer you can print out on paper what appears on your screen.*

However, you will need to be careful about the length of your PRINT statements. The Commodore 64 can only take up to 80 characters (letters, numbers, symbols or spaces) in any one program line. This means that if a program line runs over more than two lines on your screen, it will certainly not work, and you will probably get a 'SYNTAX ERROR' message when you press RETURN.

To be completely safe, you should leave a gap of at least one space at the end of the second line (that is, use a maximum of 79 characters). This is why, for example, the PRINT statements for the first paragraph of your letter have been split over two lines. But you will notice that there is a semi-colon at the end of lines 260, 280 and 290. This is to make the PRINT statements in lines 270, 290 and 300 follow on from the previous words. Try taking the semi-colons out and see what difference this makes when you run the program again.



▲ *To type in lower-case (small) letters, hold down the SHIFT key and press the C= key.*

Using number and string variables together

Here is a game which uses both number and string variables. So far, you have always used CAPITAL letters. When you switched on the computer, you were in what is called the 'Upper Case/Graphic Mode', in which everything you type is in CAPITAL (upper-case) letters. But you may feel that you would like the words which appear in your PRINT statements to be in a mixture of capitals and lower-case (small) letters.

? Problem 18: How can you type lower-case letters?

✓ If you want to write a program which has both capital and lower-case letters, then before you start writing the program you should keep one finger pressed on the SHIFT key, while pressing the C= (Commodore) key at the bottom left of the keyboard.

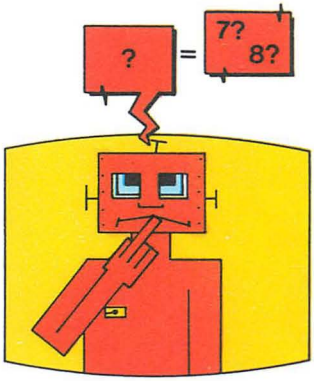
Now you are in what is called the 'Upper and Lower Case Mode'. In this mode, everything, including instructions such as PRINT, will appear in

lower case. If you want any of the letters in your PRINT statements to be capitals, then you need to keep one finger pressed on the SHIFT key while you type them. Or, you can press down the SHIFT LOCK key (above SHIFT), which will allow you to type in capitals until you press SHIFT LOCK again to release it. Do not type any of the actual instruction words, such as PRINT or INPUT in capitals while you are in this mode, however, as the computer will not accept them.

(To get back to the 'Upper Case/Graphic Mode', simply keep one finger on the SHIFT key, and press the Commodore key.)

Now for the game, which you should enter in 'Upper and Lower Case Mode'. You will notice that (CLR/HOME) now appears on the screen as an 'S':

```
10  rem "guess me"
20  print "(CLR/HOME)"
30  print "This is a guessing-game
for TWO players."
40  print
50  print "Please enter the FIRST
player's name."
60  input a$
70  print "Please enter the SECOND
player's name."
80  input b$
90  print
100 print "Thank you, ";a$;" and ";b$
110 print
120 print b$;", please stop looking
at the screen."
130 print a$;", please type a number
between 1 and 20 and then press
RETURN."
140 input a
150 print "Right! Now press
RETURN again to continue."
160 input z$
170 print "(CLR/HOME)"
180 print "Now ask ";b$;" to look
at the screen."
190 rem guessing until right loop
```



▼ *The first part of this program stores the names of the players and the number to be guessed.*

```
This is a guessing-game for
TWO players.

Please enter the FIRST
player's name.
? Jim
Please enter the SECOND
player's name.
? Jenny

Thank you, Jim and Jenny

Jenny, please stop looking at
the screen.
Jim, please type a number
between 1 and 20 and then
press RETURN.
? 13
Right! Now press RETURN
again to continue.
```

```
200  print b$;", guess what the
number between 1 and 20 is:"
210  input b
220  print "(CLR/HOME)"
230  if b <>a then print "Hard
luck! ";b;" was not correct."
240  if b <>a then goto 190
250  print "WELL DONE, ";b$;", YOU
ARE RIGHT!!! The number was ";a
260  print
270  print
280  print "Would anyone like another
turn? Press y for yes or n for no."
290  input c$
300  if c$="y" or c$="Y" then
goto 20
310  if c$ <>"n" and c$ <>"N" then
goto 290
320  print "Goodbye then!"
330  end
```

< is less than

> is more than

<> is less or more than

The first part of this program is very easy to follow:
Line 30 explains that it is a guessing-game.
Lines 50 to 80 get information into the computer about the names of the two players, using the string variables a$ and b$.
At line 140 the first player inputs a number, using the number variable a. (This stores the number at a place in the computer's memory labelled 'a'.)
At line 210 the second player inputs his or her guess, using the number variable b. (This stores the guess at a place labelled 'b'.)

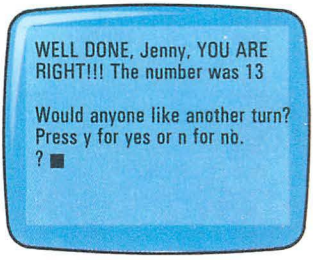
Now the program is a little harder to follow. If the second player has guessed right, the number in the store labelled b will be the same as the number in the store labelled a. But the second player may have guessed wrong. If he has guessed wrong, b will be less or more than a.

You have already used the symbol < meaning 'is less than'. If you put beside it the symbol > meaning 'is more than', you then have <> which means 'is less or more than', or in other words, 'is not equal to'. So now:
At line 240 the computer is told that IF the second

player is wrong (if b < >a) THEN it must GOTO (jump back to) line 19Ø to give the second player another chance to guess the right number. So the lines from 19Ø to 24Ø form a loop, and the computer will go round and round in circles until the right answer has been given at input b.

You have made this loop as clear as possible to understand by putting a REM line at line 19Ø, explaining that the next section of the program is a 'guessing until right loop'. A REM line which is used inside the program to help you remember which part of a program is doing what, and not as a program title, does not need inverted commas and can be as long as you like.

When the correct number has been guessed (that is b=a) the computer will take no notice of lines 23Ø and 24Ø, and will go straight to line 25Ø, where it is told to print a 'Well done' message. When the guess is right, the computer will soon reach line 28Ø. Then you come to some more IF..THEN statements. IF anyone inputs y or Y, THEN the computer will GOTO line 2Ø, where it will clear the screen and a new game will begin.



▲ *The second part of the program compares the second player's guess with the number it has in its store. If the guess is right, the computer prints a 'Well done' message and offers another game.*

? Problem 19: But someone might input another letter.

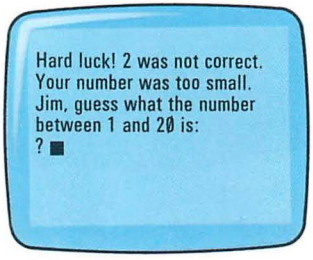
✓ If the letter is n or N, then the computer will ignore lines 3ØØ and 31Ø and will carry on to line 32Ø, a 'Goodbye' message will appear, and the program will end. But if by mistake the player has not typed either a capital or a lower-case y or n, then the computer will ignore line 3ØØ. But at line 31Ø it will jump back to line 28Ø to give the player another chance to press y or n.

? Problem 20: How can you stop the computer moving through a program too quickly?

✓ *Lines 15Ø and 16Ø* deal with this problem. The computer can always be stopped for as long as you like by an INPUT line. When there is an INPUT line the computer has to wait until an input has been made before it can continue. Pressing the RETURN key is one form of input. So at line 15Ø you told the player that if the game was to continue, the RETURN

key must be pressed. Pressing the RETURN key gave the computer the input for which it was looking at line 16Ø and so it went on with the program.

Ways to improve "guess me"



▲ *The program can also be made to give the player clues.*

1) See if you can work out some ways to improve the appearance of the program on the screen. A few more PRINT lines might be useful, for example.
2) At the moment, there is something wrong, not just with the appearance of the game, but with the way it is made up (its structure). When you have made a guess, you have no idea whether your guess is larger or smaller than the right answer. It would make the game more interesting if you added a line instructing the computer to print a message saying either 'Your number was too small' or 'Your number was too big'. To do this, you could use more IF...THEN statements. Try adding these lines:

```
233 if_b <a_then_print_"Your_
number_was_too_small."
235 if_b >a_then_print_"Your_
number_was_too_big."
```

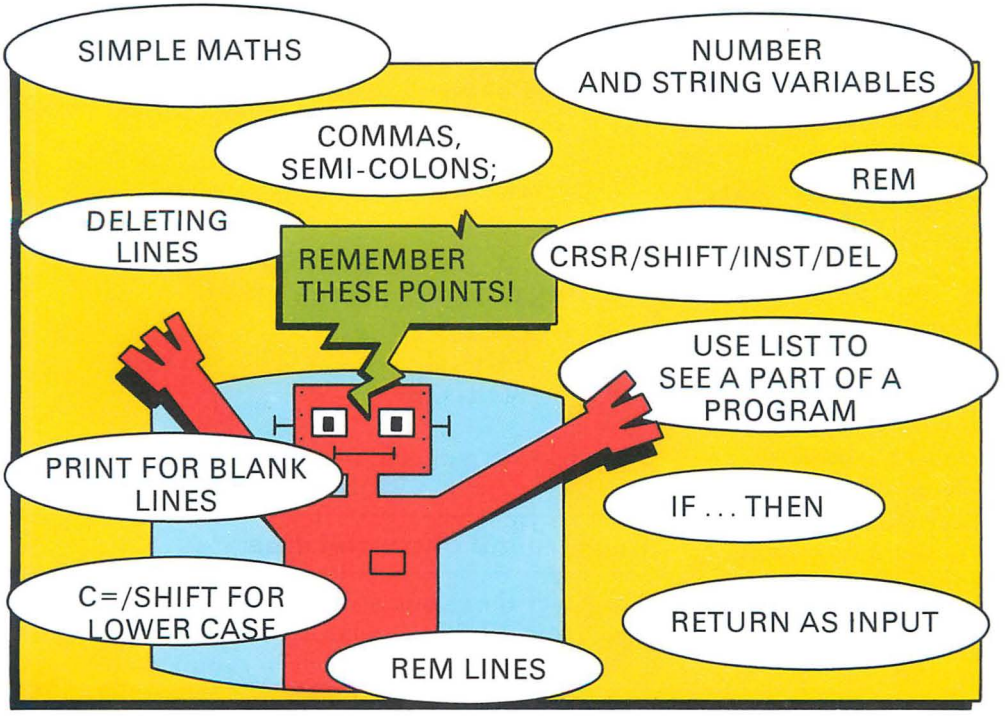
If you like this game and want to SAVE it on tape, so that you can play it at some other time without having to type it all in again, see page 64.

? Problem 21: Now that you have finished with this program, how can you get back into the Upper Case/Graphic Mode in which you began?
✓ Simply press the C= (Commodore) key while holding down the SHIFT key again.

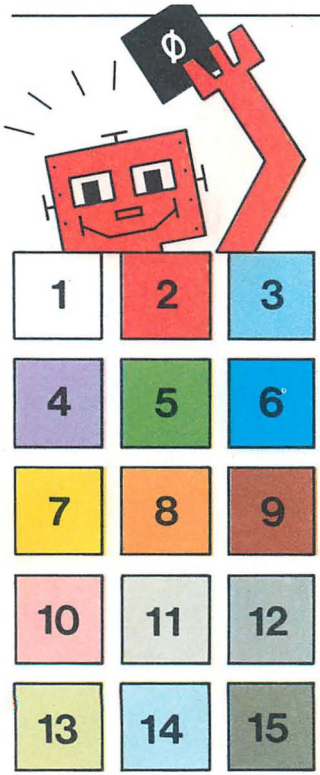
A reminder

You have now learned:
1) how to use the computer for simple maths;
2) how to use number and string variables;
3) how to use commas to create gaps, and semi-colons to close them up;

4) how to delete lines once they have been entered, and put in new ones;
5) how to use REM to give your program a title;
6) how to edit programs using the CRSR keys, and the SHIFT key with the INST/DEL key;
7) how to look at different parts of a program listing when it is too long to fit on a single screen.
8) how to use PRINT to leave blank lines on the screen;
9) how to use IF...THEN statements;
10) how to use the C= (Commodore) key and the SHIFT key to go into Upper Case/Lower Case Mode, in order to enter programs with PRINT statements containing both CAPITALS and small letters; and how to get back into the normal Upper Case/Graphic Mode;
11) how to make it easier to see what is happening when there is a loop in a program by putting in a REM line at the beginning of the loop;
12) how to use the RETURN key as an INPUT.



3 Colouring Words and Backgrounds



▲ *You can use the instruction POKE 53280, followed by a comma, followed by one of these numbers, to put a coloured border on your screen. POKE 53281, used in the same way, will colour the centre of your screen.*

The Commodore 64 computer usually shows light blue text (words) on a dark blue central area of the screen, with a light blue border going all round the edge of the screen. But you can change both the colours of the border and the central area of the screen, and the colour of the text.

The border area

To change the colour of the border is easy. You simply type POKE 53280, followed by a comma, followed by a number. There are fifteen numbers to choose from: 0 gives black, 1 gives white, 2 gives red, 3 gives cyan (a light blue), 4 gives purple, 5 gives green, 6 gives blue, 7 gives yellow, 8 gives orange, 9 gives brown, 10 gives light red, 11, 12 and 15 give different shades of grey, 13 gives light green, and 14 gives another light blue. So POKE 53280,2 would give you a red border to the screen.

Try typing in this instruction as one of the first lines of a program:

```
10  REM _"COLOURING"
20  PRINT _"(CLR/HOME)"
30  POKE _53280,2
40  PRINT _"NOW _YOU _HAVE _A _
RED _BORDER"
```

Run this program, and the red border will appear.

Background of central area

To change the colour of the central area of the screen you type POKE 53281, followed by a comma, followed by a number. There are fifteen numbers to choose from and they are exactly the same as the ones for the

border area. So POKE 53281,1 would give you a white central area of the screen. Try adding these lines to your program:

```
50  PRINT
60  PRINT
70  PRINT "PRESS RETURN TO
GO ON"
80  INPUT Z$
90  PRINT "(CLR/HOME)"
100 POKE 53281,1
110 PRINT "NOW YOU HAVE A
WHITE CENTRAL SCREEN"
```

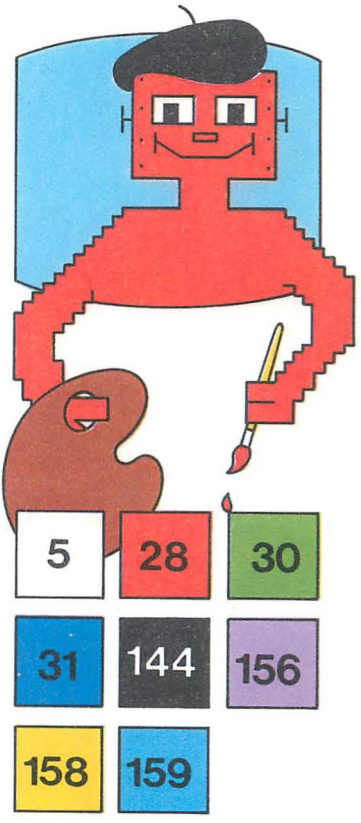
Colour of text using CHR$

To change the colour of the text you type CHR$ and after CHR$ you put a special number in brackets, followed by a semi-colon. Here are the numbers you can use and what they will do:

5 will produce WHITE text
28 will produce RED text
30 will produce GREEN text
31 will produce BLUE text
144 will produce BLACK text
156 will produce PURPLE text
158 will produce YELLOW text
159 will produce CYAN text

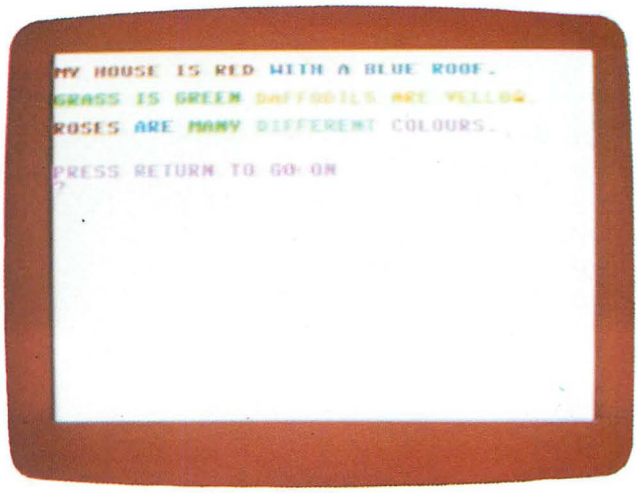
Try adding these lines to your program:

```
120 PRINT
130 PRINT
140 PRINT "PRESS RETURN TO
GO ON"
150 INPUT Z$
160 PRINT "(CLR/HOME)"
170 PRINT CHR$(28);"MY HOUSE
IS RED";CHR$(31);" WITH A BLUE
ROOF."
180 PRINT
190 PRINT CHR$(30);"GRASS IS
GREEN";CHR$(158);" DAFFODILS
ARE YELLOW."
```



▲ *These are the numbers you use with CHR$ to give you coloured text.*

▶ *As well as using the instruction CHR$ to change the colour of text in a program, you can use the CTRL key with the number keys from 1 to 8.*



If you run the program you will see that after each CHR$ instruction the words change colour. There is another method of changing the text colour.

Colour of text using CTRL key

You can also change the colour of the text within a PRINT instruction by adding CTRL instructions. You do this by keeping one finger pressed on the CTRL key (near the top left of the keyboard), while you press one of the keys on the top row of the keyboard which have the names of colours in front of them, beginning with BLK (Black) in front of the 1 key and ending with YEL (Yellow) in front of the 8 key. Try adding these lines to the program:

```
200  PRINT
210  PRINT_"(CTRL 3)ROSES_(CTRL 7)
ARE_(CTRL 6)MANY_(CTRL 4)
DIFFERENT_(CTRL 5)COLOURS."
```

(CTRL 3) simply means that you keep one finger on the CTRL key while pressing the 3 key, which has the word 'RED' in front of it. When you do this, you will see that a £ sign appears on the screen in your program line, the beginning of which will look like this:

```
21Ø PRINT "£ROSES
```

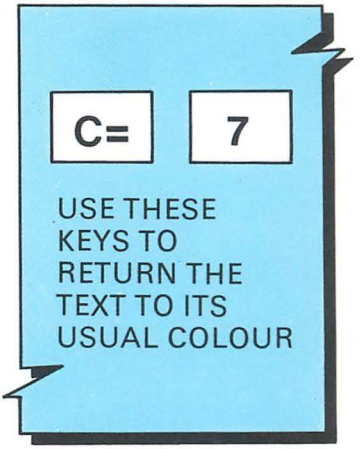
Every CTRL instruction will leave another symbol in the program line – but those symbols don't appear when you run the program, as you can see.

Getting back to normal

To get back to a light blue border, you need to use POKE 5328Ø,14. To get back to a blue central screen, you use POKE 53281,6.

❓ Problem 22: How can you get the text back to light blue, when there is no CHR$ instruction or CTRL instruction for a light blue that matches the original text colour?

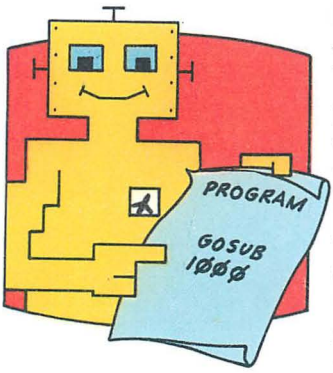
✅ A further range of text colours is available if you use the C= (Commodore) key with the number keys, in exactly the same way as you used the CTRL key. (See page 56 of your *User Manual* for the range of C= colours.) You can get the correct light blue by pressing C= and 7 together (written as (C= 7) in the program). Now add these lines to your program:




```
22Ø  PRINT
23Ø  PRINT
24Ø  PRINT_"PRESS_RETURN_TO_
GO_ON"
25Ø  INPUT_Z$
26Ø  PRINT_"(CLR/HOME)"
27Ø  POKE_5328Ø,14
28Ø  POKE_53281,6
29Ø  PRINT_"(C= 7)BACK_TO_
NORMAL."
3ØØ  END
```

Run the whole program a few times, carefully comparing what happens on the screen with the list of instructions in this book. Now try your own experiments with coloured text and backgrounds. Why not go back to the "guess me" program and put some of the words which appear on the screen into

colour? If you have saved the final version on a tape, you will only need to LOAD it back into the computer (see page 65), and then you can get to work at once.



Using a group of lines more than once

You may have noticed that in the "COLOURING" program, lines 5Ø to 9Ø, 12Ø to 16Ø and 22Ø to 26Ø are exactly the same. Whenever you are writing a program and have a group of lines which you want to use more than once, it is a good idea to write them only once after the END of the program. Then put a special instruction into the program which will call them up whenever you want them. The special instruction is GOSUB, followed by the number of the first line in the group of lines which you wish to call up. Try writing the "COLOURING" program again, putting in GOSUB instead of lines 5Ø to 9Ø, lines 12Ø to 16Ø, and lines 22Ø to 26Ø. The program will look like this:

```
 1Ø  REM _"COLOURING"
 2Ø  PRINT _"(CLR/HOME)"
 3Ø  POKE _53280,2
 4Ø  PRINT _"NOW _YOU _HAVE _A _
RED _BORDER"
 5Ø  GOSUB _1ØØØ
 6Ø  POKE _53281,1
 7Ø  PRINT _"NOW _YOU _HAVE _A _
WHITE _CENTRAL _SCREEN"
 8Ø  GOSUB _1ØØØ
 9Ø  PRINT _CHR$(28);"MY _HOUSE _
IS _RED";CHR$(31);" _WITH _A _BLUE _
ROOF."
1ØØ  PRINT
11Ø  PRINT _CHR$(3Ø);"GRASS _IS _
GREEN";CHR$(158);" _DAFFODILS _
ARE _YELLOW."
12Ø  PRINT
13Ø  PRINT _"(CTRL 3)ROSES _(CTRL 7)
ARE _(CTRL 6)MANY _(CTRL 4)
DIFFERENT _(CTRL 5)COLOURS."
14Ø  GOSUB _1ØØØ
```

```
15Ø  POKE_53280,14
16Ø  POKE_53281,6
17Ø  PRINT_"(C= 7)BACK_TO_
NORMAL."
18Ø  END
```

Now, after the END of the program, you must enter the lines which your GOSUB will call up. To make these lines easy to see and understand in a program, it is a good idea to give them a high number which sets them apart from other lines. That is why you have given the instruction GOSUB 1ØØØ instead of GOSUB 19Ø. It also helps if you make the first of the GOSUB lines a REM line, to help you remember what is happening in the lines which follow. The last of the GOSUB lines must always be the instruction RETURN, which makes the computer jump back into the main program again. So add:

```
1ØØØ  REM_PAUSE_AND_GO_ON
1Ø1Ø  PRINT
1Ø2Ø  PRINT
1Ø3Ø  PRINT_"PRESS_RETURN_TO_
GO_ON"
1Ø4Ø  INPUT_Z$
1Ø5Ø  PRINT_"(CLR/HOME)"
1Ø6Ø  RETURN
```

In this case, using a GOSUB has only saved you five lines. But it often saves more; and, in any case, the use of GOSUBs can make a program very much easier to follow. When a program is easy to follow, it is said to be 'well structured'. Good structure is the mark of a good programmer!

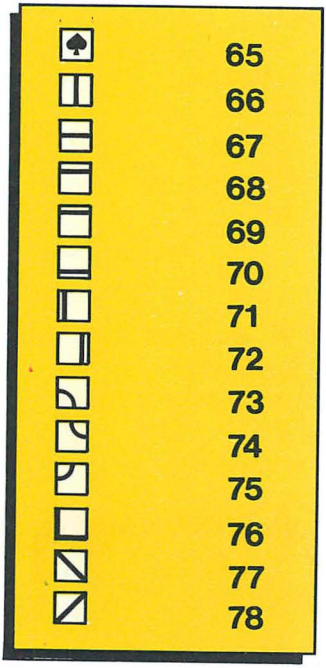
A reminder

You have now learned how to colour the border area of the screen using POKE 53280 instructions, and the central area using POKE 53281 instructions. You have learned to change the colour of the text using CHR$, CTRL and C= instructions. You have also learned how to use GOSUBs.

▼ *Remember, POKE 53280 with a number from 1 to 15 will colour the border area.*

▼ *And POKE 53281 with a number from 1 to 15 will colour the centre.*

4 Drawing with Blocks of Colour



▲ *Each of the shapes shown here and on the next few pages can be typed on your screen by using the code number given alongside it. These shapes are known as graphic characters.*

In order to make pictures (graphics) on the screen, you want to be able to fill an area with colour. To make this easier, the central area of the screen is divided up by the Commodore 64 computer into 25 rows; and each row has 40 squares.

Each of these 1000 squares can be given its own character, such as a letter, a number, a punctuation symbol, or a shape (known as a *graphic character*); and each character can be given its own colour. (More than 200 of these characters are shown in sets 1 and 2 on pages 132 to 134 of the *User Manual*. And note that all of these characters can also be shown as white letters, numbers or shapes reversed out of a coloured background, by simply renumbering the POKE codes given on these pages. This is done by adding 128 to each code number, so that the numbers run from 128 to 255, instead of 0 to 127.)

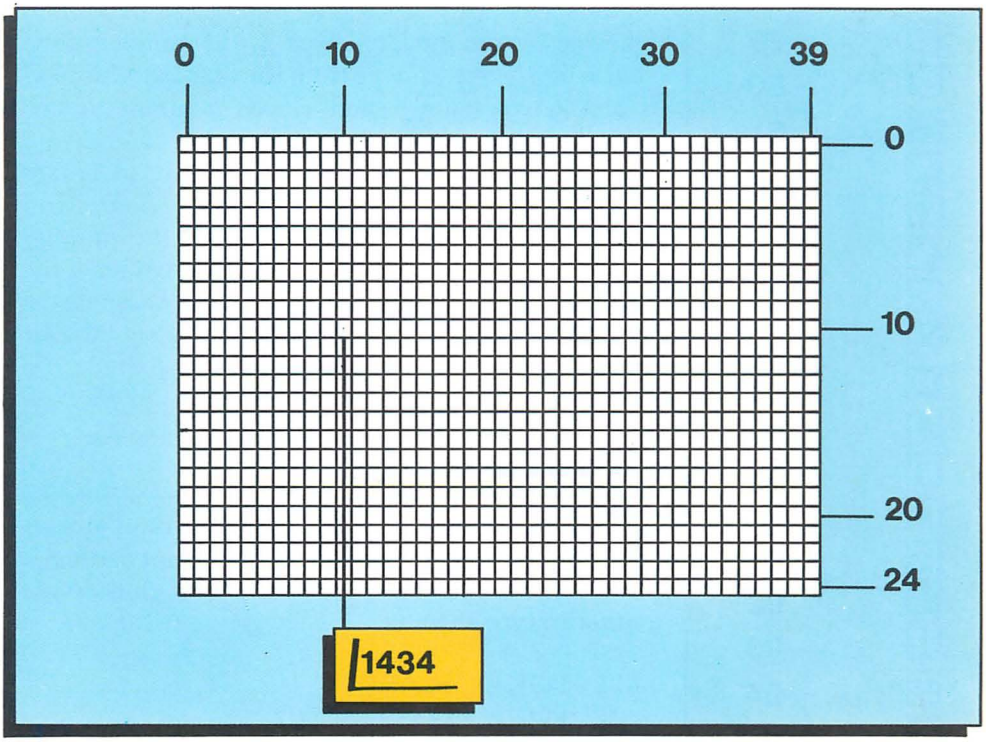
If you want a quick idea of what the characters look like on the screen, run this program:

```
10 PRINT "(CLR/HOME)"
20 POKE 53281,1
30 LET A=1104
40 FOR X=0 TO 255
50 POKE A,X
60 LET A=A+3
70 NEXT X
80 END
```

Before you begin drawing on your screen, alter the background colour of the central area of the screen to white, by typing:

```
POKE 53281,1
```

and pressing RETURN. Now you will be able to see what you are doing much more clearly.



▲ *The centre area of your screen is divided up by the computer into small squares, which are exactly the same size as the graphic characters. To put a shape on the screen you must first tell the computer exactly which square you want to be filled with which character.*

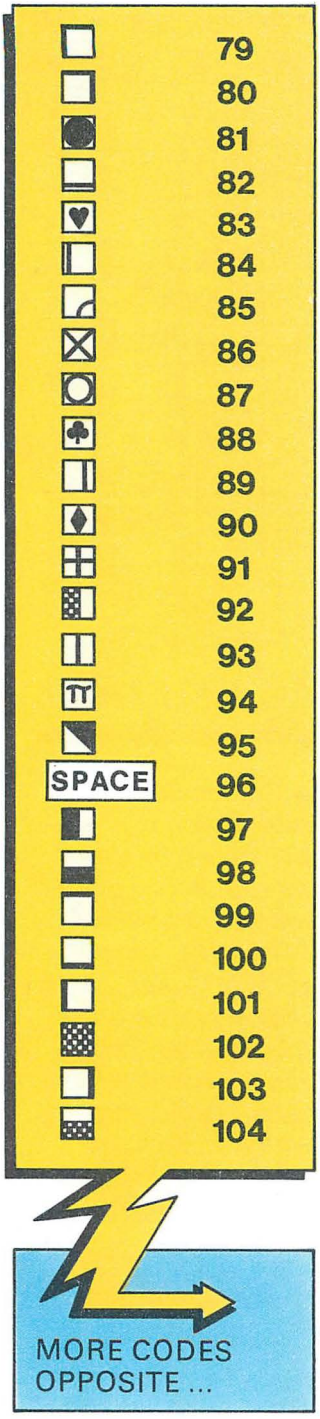
Also, study the map of the screen squares on page 63 of the *User Manual*. You will see that the top row begins at position 1Ø24 in the top left-hand corner, and runs to 1Ø63 in the top right-hand corner. The next row runs from 1Ø64 to 11Ø3, and so on, until you reach the bottom row which runs from 1984 to 2Ø23. Whenever you are drawing on your screen you should have this map in front of you, so that you can give the computer the correct co-ordinates of the squares you want to colour in. Now you are ready to begin.

Colouring a single square

Type in these lines and then run them:

```
1Ø  POKE_1434,1Ø2
2Ø  POKE_1434+54272,2
```

In line 1Ø you are placing character 1Ø2 in square 1434



on the screen, by POKEing the number of the square, followed by a comma, followed by the number of the character. As you can see from the map, square 1434 is eleven rows down and eleven columns across. Character 1Ø2 is a ▒.

In line 2Ø you are colouring the character in square 1434 in red. To change the colour in a square, you POKE the number of the square plus the number 54272, followed by a comma, followed by one of the numbers from Ø to 15 that you use for changing the border or central screen colour. In line 2Ø, the number 2 changes the character to red.

Colouring a row of squares

Now, imagine that you wanted to fill a row of squares with colour, starting with square 1434 and ending at 145Ø. The first lines could look like this – but do not bother to type them in!

```
1Ø  POKE_1434,1Ø2
2Ø  POKE_1434+54272,2
3Ø  POKE_1435,1Ø2
4Ø  POKE_1435+54272,2
5Ø  POKE_1436,1Ø2
6Ø  POKE_1436+54272,2
```

and so on. At this rate, it would take another 28 instructions to finish the row. Luckily, there is a way to make this much faster and easier, by using a FOR...NEXT loop. Try typing these lines:

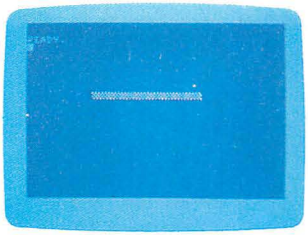
```
1Ø  REM_"LINE"
2Ø  PRINT_"(CLR/HOME)"
3Ø  FOR_X=Ø_TO_16
4Ø  POKE_X+1434,1Ø2
5Ø  POKE_X+1434+54272,7
6Ø  NEXT_X
7Ø  END
```

When you run this program, the row of colour appears very quickly. This is what is happening:
The lines from 3Ø to 6Ø are a FOR...NEXT loop. The first

time the computer reaches line 3Ø it is told that X has been given values all the way from Ø to 16 (for the 17 squares that you want to colour in). Because it is the first time it has reached the line it has to choose the first value, which is Ø. So at line 4Ø, it POKEs the square number 1434, and fills it with character 1Ø2; and at line 5Ø, it POKEs the colour yellow into square number 1434. Then, when it reaches line 6Ø it is told to go back to find the NEXT value of X so it returns to line 3Ø.

This time, it discovers that the next value of X is 1. So at line 4Ø it POKEs the next square, number 1435, with character 1Ø2; and at line 5Ø it POKEs the colour yellow into square number 1435 . . . and so on, until it has POKEd yellow into square number 145Ø. At this point, X = 16 so the computer finds no more values of X and it goes on to line 7Ø at last, and the program comes to an END.

By using a FOR . . . NEXT loop in this way, you are making the computer carry out a certain task a particular number of times without having to repeat all the instructions.

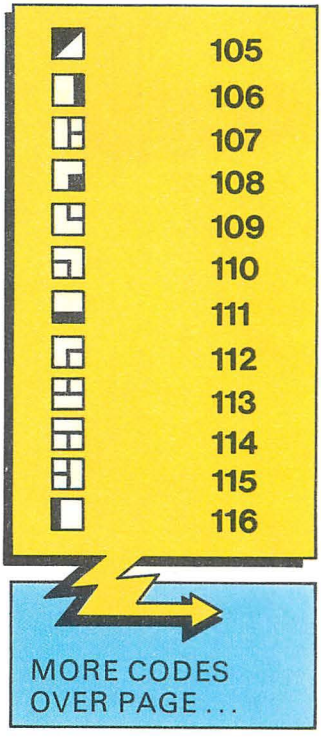


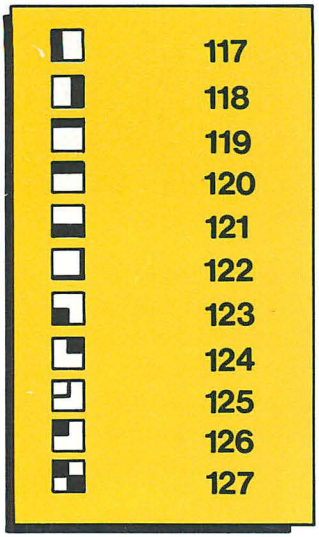
▲ *By using a FOR . . . NEXT loop, you can make the computer fill in a row of squares without having to give it the co-ordinates for each square.*

Colouring several rows of squares

Supposing that you wanted to colour not only the row of squares from 1434 to 145Ø, but also the nine rows of squares beneath it (from 1474 to 149Ø; from 1514 to 153Ø; from 1554 to 157Ø; and so on up to the row 1794 to 181Ø). You could do this by using six lots of FOR . . . NEXT loops of three lines each, beginning like this – but do not type these lines in:

```
 1Ø  REM _"BLOCK"
 2Ø  PRINT _"(CLR/HOME)"
 3Ø  FOR _X=Ø _TO _16
 4Ø  POKE _X+1434,1Ø2
 5Ø  POKE _X+1434+54272,7
 6Ø  NEXT _X
 7Ø  FOR _X=Ø _TO _16
 8Ø  POKE _X+1474,1Ø2
 9Ø  POKE _X+1474+54272,7
1ØØ  NEXT _X
```





and so on. At this rate, it would take another 32 instructions to finish the block of colour. Again, there is a way to make this much faster and easier by using one FOR...NEXT loop inside another FOR...NEXT loop. First, notice that the number of a square is always 40 numbers more than the square above it. Now, try typing these lines:

```
10 REM "BLOCK"
20 PRINT "(CLR/HOME)"
30 FOR Y=0 TO 9
40 FOR X=0+Y*40 TO 16+Y*40
50 POKE X+1434,102
60 POKE X+1434+54272,7
70 NEXT X
80 NEXT Y
90 END
```

When you run this program, the complete block of colour gradually appears. This is what is happening: *The lines from 30 to 80* are a FOR...NEXT loop. The first time the computer reaches line 30, it is told that Y has been given values all the way from 0 to 9 (for each of the 10 rows that you want to colour in). Because it is the first time it has reached the line, it has to choose the first value, which is 0. So Y now equals 0. But before it can go on with this FOR...NEXT loop it finds at line 40 that it has reached another FOR...NEXT loop, which runs from lines 40 to 70. So with Y at the value of 0, it has to get all the way through this second FOR...NEXT loop which is telling it to colour in the row from square 1434 to square 1450.

The first time that the computer reaches line 40 it finds that the value of X is 0+Y*40. At present, the value of Y is 0, so X is 0+0*40, which is 0. So at line 50 the computer fills the first square, which is 1434, with character 102 and at line 60 it colours this character yellow.

At line 70, the computer is sent back to line 40 to get the next value of X. This time the value of X is 1+0*40, which is 1, so the computer will move to the next square, which is 1435. (Remember that when sums have more than one part, the computer will always carry out the times and divided bys before any

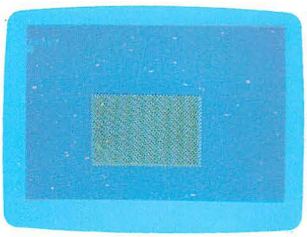
plus or minus sums. So with this sum, the computer is working out that $\emptyset*4\emptyset=\emptyset$, and that $1+\emptyset=1$.)

When X equals $16+\emptyset*4\emptyset$, which is 16, the first row of squares has been filled in. There are no more values of X and so the computer moves on to line $8\emptyset$ where it is told to go back to line $3\emptyset$ to get the next value of Y. So far, the program would have worked just as well if line $4\emptyset$ had read:

```
4Ø  FOR _X=Ø _TO _16
```

However, because you want to position the next row of squares immediately below the first row, each square in the second row must have a value that is $4\emptyset$ units more than the square immediately above it. (Remember that the length of a row on your screen is exactly $4\emptyset$ squares.) So the first square to be coloured in the second row must be square 1474, which is $4\emptyset$ units more than square 1434. This is why line $4\emptyset$ includes the sums $\emptyset+Y*4\emptyset$ TO $16+Y*4\emptyset$. The next time the computer reaches line $4\emptyset$, Y has been given a value of 1; and the first value of X for this row of squares will be $\emptyset+1*4\emptyset$, which is $4\emptyset$. This means that at line $5\emptyset$, the square to be filled in will be square 1474 $(1434+4\emptyset)$.

The computer carries on in this way through both FOR...NEXT loops, until it has reached the last square of the last row. Then it finds no more values of Y and so it goes to line $9\emptyset$ and the program ends.

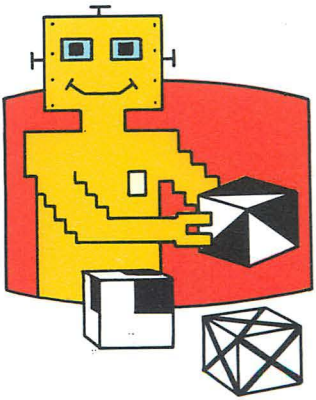


▲ *By using one FOR ... NEXT loop within another FOR ... NEXT loop, you can make the computer print out a whole block of squares.*

Printing more than one block of colour in a program

Imagine that you wanted to use several different blocks of colour in one program. You can use the same set of instructions for every block, by making some of the items in those instructions variable items. You can then use this set of instructions with a GOSUB instruction (as you did with the "COLOURING" program on page 36).

Look at the last program again. Five of the items in this set of instructions can be represented by a number variable. You can LET:



A = the number of rows, less one, on which squares are to be printed;
B = the number of columns, less one, on which squares are to be printed;
C = the number of the top left-hand square in the block to be printed;
D = the number of the character which is to be POKEd into the squares;
E = the number of the colour which is to be POKEd into the squares.

You can then write out the GOSUB set of instructions like this:

```
1Ø1Ø  FOR_Y=Ø_TO_A
1Ø2Ø  FOR_X=Ø+Y*4Ø_TO_B+Y*4Ø
1Ø3Ø  POKE_X+C,D
1Ø4Ø  POKE_X+C+54272,E
1Ø5Ø  NEXT_X
1Ø6Ø  NEXT_Y
1Ø7Ø  RETURN
```

Now, whenever you wanted to put a block of colour in a program, you would simply need some lines to define the variables, and then a line to send a GOSUB to this set of instructions. For the "BLOCK" program, the lines defining the variables could have been:

```
3Ø  LET_A=9
4Ø  LET_B=16
5Ø  LET_C=1434
6Ø  LET_D=1Ø2
7Ø  LET_E=7
```

followed by a GOSUB line, saying:

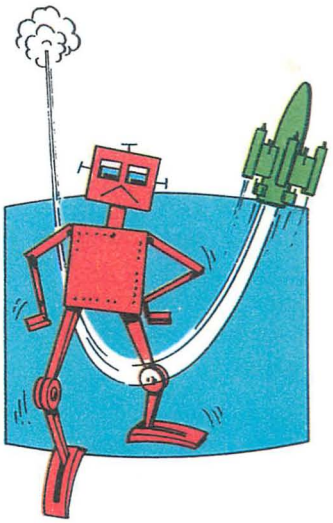
```
8Ø  GOSUB_1Ø1Ø
```

? Problem 23: You have a group of very short lines taking up too much space in the program.

✓ You are allowed to put more than one instruction on the same line, provided that you separate the instructions by a : (colon), and you keep the line to less than 8Ø characters in length.

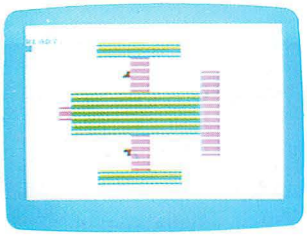
So lines 3Ø to 7Ø could read:

```
3Ø  LET _A=9:LET _B=16:LET _
C=1434;LET _D=1Ø2:LET _E=7
```



Now try typing out this program, which has seven blocks of colour and two single squares with coloured characters in them.

```
  1Ø  REM _"SPACESHIP"
  2Ø  PRINT _"(CLR/HOME)":POKE _
53281,1
  3Ø  REM _COLOUR _MAIN _HULL
  4Ø  LET _A=5:LET _B=19:LET _
C=1431:LET _D=192:LET _E=5
  5Ø  GOSUB _1ØØØ
  6Ø  REM _STABILIZERS _AT _WING _
ENDS
  7Ø  LET _A=1:LET _B=12:LET _
C=1155
  8Ø  GOSUB _1ØØØ
  9Ø  LET _C=1875
 1ØØ  GOSUB _1ØØØ
 11Ø  REM _WINGS
 12Ø  LET _A=4:LET _B=2:LET _
C=124Ø:LET _E=4
 13Ø  GOSUB _1ØØØ
 14Ø  LET _C=168Ø
 15Ø  GOSUB _1ØØØ
 16Ø  REM _TAIL _SECTION
 17Ø  LET _A=11:LET _C=1331
 18Ø  GOSUB _1ØØØ
 19Ø  REM _THE _NOSE
 2ØØ  LET _A=1:LET _B=1:LET _
C=15Ø9:LET _E=1Ø
 21Ø  GOSUB _1ØØØ
 22Ø  REM _THE _GUNS
 23Ø  POKE _1319,254:POKE _
1319+54272,9
 24Ø  POKE _1759,251:POKE _
1759+54272,9
 25Ø  END
1ØØØ  REM _A _BLOCK _OF _COLOUR
1Ø1Ø  FOR _Y=Ø _TO _A
```



▲ *Your screen should look like this. If it doesn't, check through your program lines very carefully.*

```
1Ø2Ø  FOR_X=Ø+Y*4Ø_TO_B+Y*4Ø
1Ø3Ø  POKE_X+C,D
1Ø4Ø  POKE_X+C+54272,E
1Ø5Ø  NEXT_X
1Ø6Ø  NEXT_Y
1Ø7Ø  RETURN
```

Run the program to check that it is working properly. If not, then LIST the program and check the lines carefully.

Now look at the program again. You will see that in line 2Ø, as well as clearing the screen, there is a POKE 53281,1 instruction, which alters the central area of the screen to white so that the drawing will show up more clearly. In line 4Ø, you define all the variables. But you will notice that in line 7Ø you define only some of them. This is because variables D and E are the same as in line 4Ø, and do not need defining again.

Now try experimenting with blocks of colour to make your own pictures. How about a castle? Or a soldier with a square helmet? Remember that although it is fun just to try things out, you will get better results if you plot out your picture on graph paper first so that you can work out all your co-ordinates. The diagram on page 63 of the *User Manual* is a very useful guide for this.

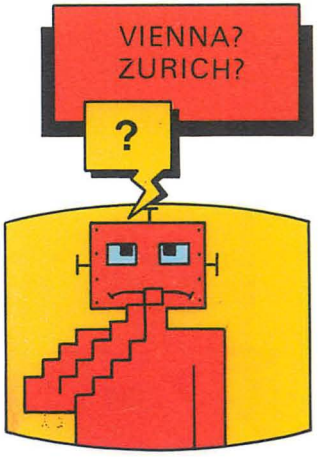
A reminder

You have now learned how to position a square of colour on the screen, and how to use FOR...NEXT loops to draw blocks of colour. You have also learned how to use a : (colon) to place more than one instruction on the same program line.

5 Inventing a Quiz

One of the useful things a computer can do is to test either your general knowledge, or your knowledge of a particular subject. You can also make up quizzes to amuse your family and friends.

Two problems are involved. First you must work out how the quiz is to appear on the screen; and then you must work out how the necessary information for the questions and answers is to be stored.



Screen appearance

Supposing that you decide to write a quiz on the Capital cities of the world. You will want a question to appear on the screen, saying something like:

```
WHAT IS THE CAPITAL OF AUSTRIA?
ENTER YOUR GUESS:
```

Then, if the answer is right, you could print a message saying:

```
WELL DONE! THE CAPITAL
OF AUSTRIA IS VIENNA
```

But if the answer is wrong then you could print:

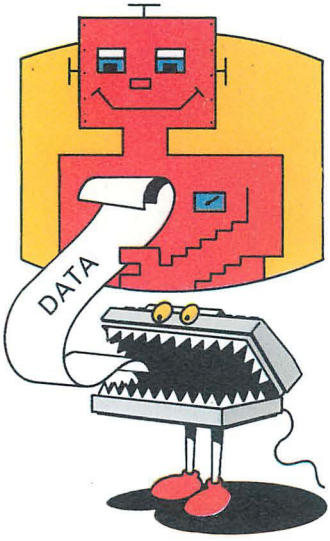
```
HARD LUCK! THE CAPITAL
OF AUSTRIA IS VIENNA
```

If you were only asking one question, it would be easy to write a program to do this, such as:

```
 10  REM "QUIZ"
 20  PRINT "(CLR/HOME)":POKE
53281,1
 30  LET C$="VIENNA"
 40  PRINT CHR$(28);"WHAT IS
THE CAPITAL OF AUSTRIA?"
 50  PRINT "ENTER YOUR GUESS"
```

```
60  INPUT _G$
70  PRINT:PRINT
80  IF _G$=C$ _THEN _PRINT _
CHR$(28);"WELL _DONE! _THE _
CAPITAL _OF _AUSTRIA _IS _";C$
90  IF _G$ < >C$ _THEN _PRINT _
CHR$(31);"HARD _LUCK! _THE _
CAPITAL _OF _AUSTRIA _IS _";C$
100  END
```

However, you will want more than one question; and for each question there will be not only a different country (so the word 'AUSTRIA' in lines 40, 80 and 90 will have to be different each time), but there will be a different Capital (or C$) also.



How to store information using READ . . . DATA

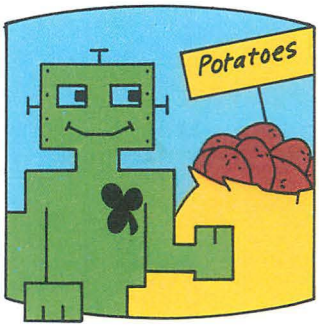
To make the computer go through a similar series of actions using certain different items of information each time, you can use READ . . . DATA statements. The DATA line contains the information that you want the computer to use; and the READ line tells the computer to read and store that information so it can be used a bit at a time. Here is an example:

```
10  REM _"CROPS"
20  PRINT _"(CLR/HOME)"
30  FOR _X=1 _TO _3
40  READ _A$,B$,C$
50  PRINT _A$;" _GROWS _";B$;" _IN _
";C$
60  NEXT _X
70  END
80  DATA _MICHAEL,POTATOES,
IRELAND
90  DATA _IVAN,WHEAT,RUSSIA
100  DATA _JOHN,CABBAGE,ENGLAND
```

When the computer reaches line 40, the READ instruction sends it to the first line of DATA, which is at line 80. It has been told to READ three string

variables, A$, B$ and C$; and it finds (separated by commas) the three words, MICHAEL, POTATOES and IRELAND. So it will store MICHAEL in A$, POTATOES in B$ and IRELAND in C$. In line 5Ø the computer is told to print the information stored in these strings, so it will print:

```
MICHAEL GROWS POTATOES
IN IRELAND
```



At line 6Ø, the FOR...NEXT loop sends it back to line 3Ø again. The computer knows that it has already READ the DATA at line 8Ø, so this time the READ instruction at line 4Ø sends it to line 9Ø, where A$ becomes IVAN, B$ becomes WHEAT and C$ becomes RUSSIA. In this way, the computer will work through the lines of DATA, and cover as many lines as the FOR...NEXT loop allows.

A complete quiz

Now that you have decided how your quiz should appear on the screen, and have also learned how to store information using READ...DATA statements, your "QUIZ" program could look like this:

```
 1Ø  REM _"QUIZ"
 2Ø  PRINT _"(CLR/HOME)":POKE _
53281,1
 3Ø  FOR _A=1 _TO _4
 4Ø  READ _C$,P$
 5Ø  PRINT _CHR$(28);"WHAT _IS _
THE _CAPITAL _OF _";P$;"?"
 6Ø  PRINT _"ENTER _YOUR _GUESS:"
 7Ø  INPUT _G$
 8Ø  IF _G$=C$ _THEN _PRINT _
"WELL _DONE! _THE _CAPITAL _OF _
";P$;" _IS _";C$
 9Ø  IF _G$ < >C$ _THEN _PRINT _
CHR$(31);"HARD _LUCK! _THE _
CAPITAL _OF _";P$;" _IS _";C$
1ØØ  NEXT _A
11Ø  END
```

```
12Ø  DATA _VIENNA,AUSTRIA
13Ø  DATA _MADRID,SPAIN
14Ø  DATA _PARIS,FRANCE
15Ø  DATA _ROME,ITALY
```

Run this a few times. Then try a quiz of your own. Perhaps you could do one which tested you on French words; the first line of the question might be:

```
5Ø  PRINT _CHR$(28);"WHAT _IS _
THE _FRENCH _FOR _";C$
```

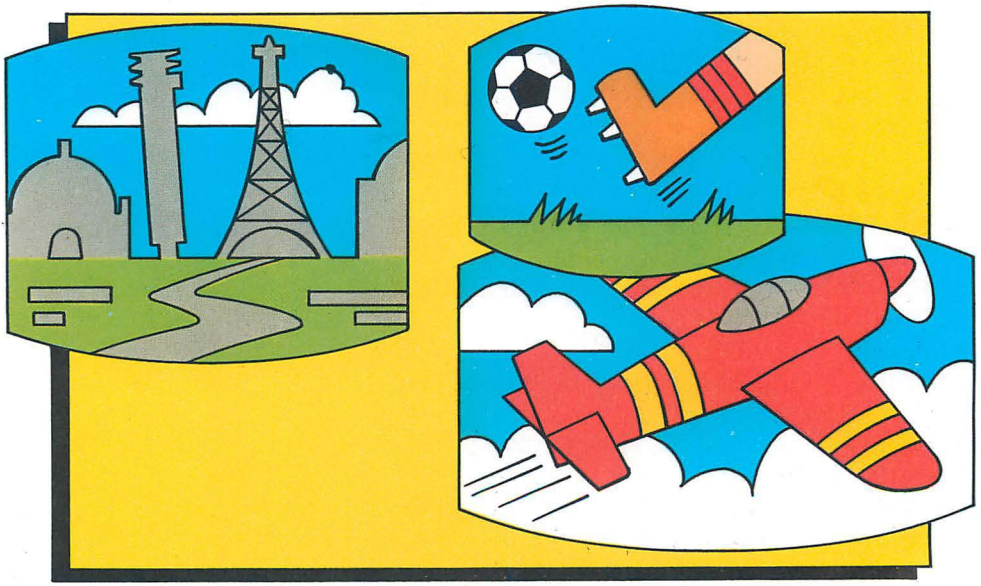
and the first line of DATA could be:

```
12Ø  DATA _THE _CHILD,L'ENFANT
```

You can type in as many READ ... DATA statements as you want, just remember to adjust the FOR ... NEXT loop at line 3Ø.

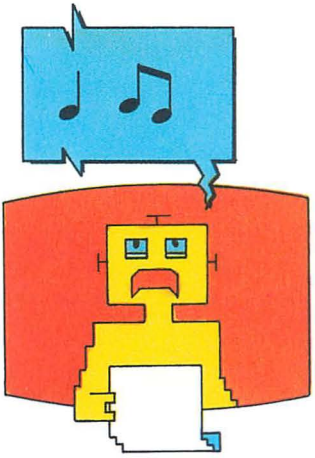
A reminder

You have now learned how to make up quizzes using READ ... DATA statements.



6 Making Sounds

You can instruct the Commodore 64 computer to make strange noises and to produce musical sounds!



Producing a single note

When you are putting sound into a program, the lines to produce a single note might look like this:

```
10  POKE 54296,12
20  POKE 54277,150
30  POKE 54278,128
40  POKE 54273,25:POKE 54272,177
50  POKE 54276,17
60  FOR T=1 TO 300:NEXT T
70  POKE 54276,0:POKE 54277,0:POKE
54278,0
80  END
```

At first, this looks like a very complicated way of producing just one note; but it is worthwhile finding out how it is done because programs without any sound can be very dull, and in any case, excellent sound is one of the best features of the Commodore 64 computer. For one thing (and unlike most other home computers), it plays the notes through the loudspeaker of your television set. This is what you have been telling the computer to do:

Line 10: Setting the Volume. The instruction POKE 54296, followed by a comma, followed by a number from 0 to 15, sets the volume. The number 15 gives the loudest sound, so:

```
POKE 54296,12
```

is quite loud, but not as loud as it could be.

Line 20: Setting the Attack/Decay level. The instruction POKE 54277, followed by a comma, followed by a number from 0 to 255, sets the speed at which the note reaches and falls from its greatest

volume. You can experiment with this if you like, but for the purposes of this book you can always use the same instruction:

```
POKE 54277,150
```

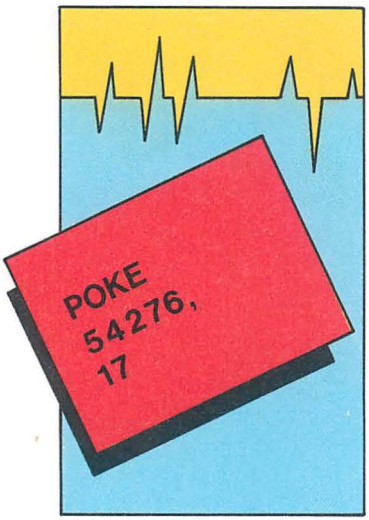
Line 30: Setting the Sustain/Release rate. The instruction POKE 54278, followed by a comma, followed by a number from 0 to 240, allows you to prolong the volume of a note for a certain time. In this book you can always use the same instruction:

```
POKE 54278,128
```

Line 40: Choosing the Note. On pages 152 to 154 of the *User Manual* you will find a complete list of the notes which the Commodore 64 can play. For each note you need to write two POKEs into the program: POKE 54273 followed by the number for the High Frequency of the note; and POKE 54272 followed by the number for the Low Frequency.

For example, you can see that the note G in the fourth octave (note 55 on page 153) has a 'Hi Freq' of 25, and a 'Low Freq' of 177. So to play that note you need the line:

```
POKE 54273,25:POKE 54272,177
```



▲ *Use this POKE instruction to set the waveform, the 'shape' of the sound.*

Line 50: Setting the Waveform. This is to do with the 'shape' of the sound. In this book you can always use:

```
POKE 54276,17
```

Line 60: Setting the Length of the Note. A FOR...NEXT loop which goes round in circles without doing anything can always be used to make the computer wait, or pause in a program. In this case, as long as the program pauses, the note goes on being played. So by increasing or decreasing the number 300 in line 60, you can have a longer or shorter note.

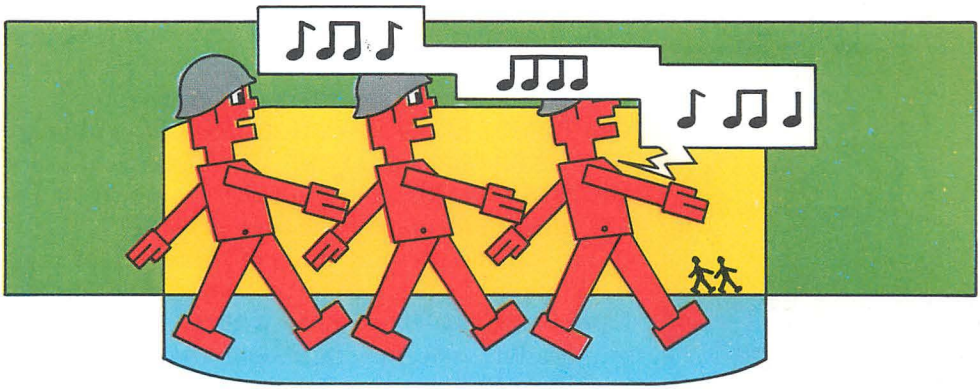
Line 70: Turning off Settings. By giving the special POKE numbers followed by a comma, followed by 0, you can turn off the waveform, attack/decay and

sustain/release settings. As you will see below, when producing a tune, you need to turn off the waveform setting (54276,Ø) after each separate note.

Producing a tune

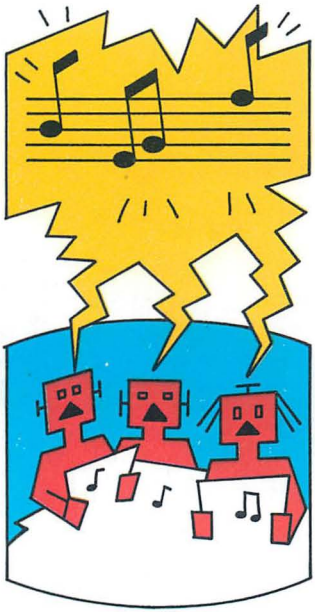
Seeing how difficult it is to produce a single note, you will be pleased to hear that a tune with many notes in it can be produced fairly easily if you can remember how to use FOR...NEXT loops and READ...DATA statements.

In the lines above, you produced the first note for the tune which follows:



```
1Ø  REM _"SOLDIER"
2Ø  POKE _54296,12
3Ø  POKE _54277,15Ø
4Ø  POKE _54278,128
5Ø  FOR _N=1 _TO _29
6Ø  READ _A,B
7Ø  POKE _54273,A:POKE _54272,B
8Ø  POKE _54276,17
9Ø  FOR _T=1 _TO _3ØØ
1ØØ NEXT _T
11Ø POKE _54276,Ø
12Ø NEXT _N
13Ø POKE _54277,Ø:POKE _54278,Ø
14Ø END
```

```
15Ø  DATA _25,177,21,154,17,37,28,214,25,
177,25,177,25,177
16Ø  DATA _22,227,22,227,19,63,16,47,28,214,
25,177,28,214,25,177
17Ø  DATA _22,227,25,177,21,154,17,37,28,
214,25,177,25,177,25,177
18Ø  DATA _22,227,22,227,19,63,16,47,28,214,
25,177
```



Lines 6Ø to 11Ø produce each single note, with READ A,B taking two numbers at a time from the lines of DATA. So the first time the computer reaches line 7Ø, it will produce G in the fourth octave, as A will be 25 and B will be 177.

Lines 5Ø to 12Ø are a FOR...NEXT loop which means that 29 notes will be played.

As the waveform setting must be switched off after each note, but the other settings only need to be switched off when the program is over, you will see that POKE 54276,Ø appears at line 11Ø, within the FOR...NEXT loop. But POKE 54277,Ø (attack/decay) and POKE 54278,Ø (sustain/release) appear at line 13Ø, after all the notes have been played.

Try making up some tunes of your own, or make the computer play a tune that you know already. Later, you can experiment with various different settings. But to begin with, keep to the settings in the "SOLDIER" program and simply change the note numbers in the DATA lines. Remember that the second number in line 5Ø must be equal to the number of notes you wish to play, whose high and low frequency numbers you have included in the DATA lines.

At the moment, each note is played for the same length of time because of the length of the pause in line 9Ø. They can all be played for a longer or shorter time by making the number 3ØØ larger or smaller. However, if you want to write anything more than the very simplest of tunes, you will need to make some notes play for different times within the same program. To do this, you must change lines 6Ø and 9Ø to:

```
6Ø  READ _A,B,C
```

```
9Ø  FOR_T=1_TO_C
```

Then you must give three numbers for each note in the DATA lines, instead of two; the third number being the length of the note. So the first note to be READ in line 15Ø could be:

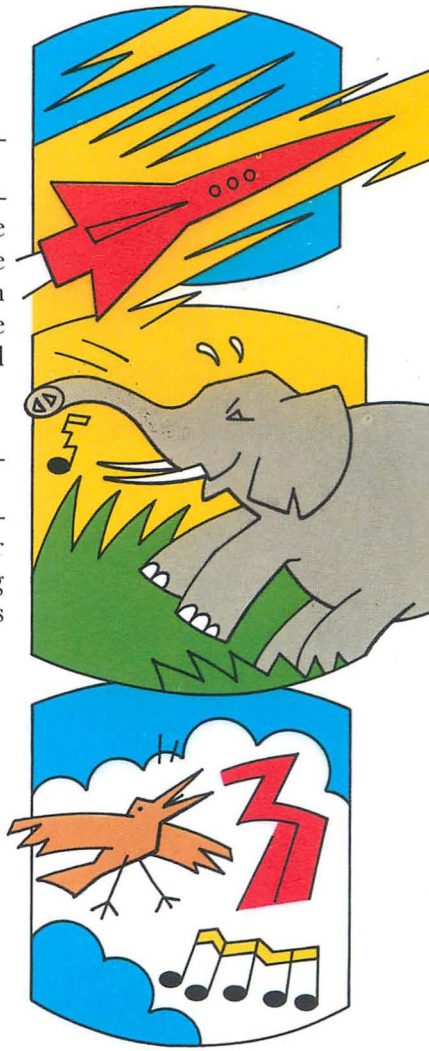
```
15Ø  DATA_25,177,3ØØ
```

Making strange noises

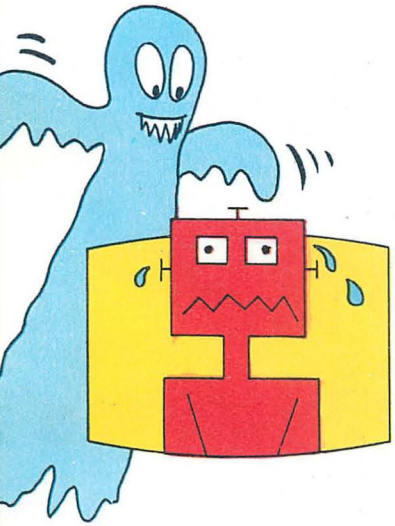
In line 8Ø of the "SOLDIER" program, you set the waveform setting to POKE 54276,17. By changing the last number of that setting from 17 to 129, you can make the computer choose sounds from a 'noise channel', which you can use to create odd sound effects for your programs.

A reminder

You have now learned how to make your computer play single notes, tunes, and strange noises, using POKE instructions together with FOR...NEXT loops and READ...DATA statements.



7 "GHOSTHUNT"



This is a very long program for what is basically a guessing-game. There is a haunted house with a ghost in one of the rooms, and you have to guess which room it is in. The trick is that it is in a different room each time! You will find a few new CHR$ colours and new ways of doing things in this program.

Use of long variable names

Until now, you have used single letters in statements such as:

```
LET A=20
```

But you can use names instead, which sometimes make it easier to see what is happening. For example, you could type:

```
LET WEALTH=20,000
```

In "GHOSTHUNT" you will find on line 30, the variable GHOSTROOM, and on line 40, the variable WINNINGS. There are only two rules to bear in mind:

1) You must not include in the same program different variables which begin with the same two letters, such as WELL and WEALTH. The computer remembers name variables by their first two letters only, so it would not be able to tell the difference between these two variables.

2) You must not include any programming words in the name. SCORE would be no good, for example, because it contains the programming word OR.

Typing graphic characters directly into programs

As you can see, there are graphic characters shown on the front of some of the keys on the keyboard. These

characters can be typed directly into PRINT statements. In "GHOSTHUNT" you will use some of them for decoration in lines 1Ø3Ø and 2Ø3Ø. To type them in, simply hold down the SHIFT key while pressing the correct character key.

Using RESTORE with READ . . . DATA

? Problem 24: Supposing that your computer has READ and used all the DATA lines in your program. But you want them to be READ and used again. For example, in "GHOSTHUNT", from lines 6ØØØ to 618Ø READ . . . DATA lines are used to create tunes. GOSUB 6ØØØ calls up these lines. But by the time GOSUB 6ØØØ has been used twice, the READ line at 6Ø5Ø has used up all the DATA, and so a new GOSUB 6ØØØ would bring the message 'OUT OF DATA ERROR IN 6Ø5Ø'.

✓ Before the next GOSUB 6ØØØ instruction, you type in the instruction RESTORE. This means that the next time the computer comes to the READ instruction at line 6Ø5Ø, it will be reset or RESTOREd to look at the first line of DATA in the program, so it will be able to READ through all the lines of DATA again.

Use of RND

This is a very important instruction, which means that you can invent games with an element of chance in them. For example, the following statement would make the computer print, at RaNDom, any whole number between 1 and 1Ø:

```
PRINT_INT(1Ø*RND(1))+1
```

Or

```
PRINT_INT(8*RND(1))+1
```

will make the computer choose a number between 1 and 8; and so on. In this program, there is a house with eight rooms; and the room with the ghost in it is

```
1Ø REM "GHOSTHUNT"
2Ø GOSUB 1ØØØ:REM
INTRODUCTION
3Ø LET GHOSTROOM =
INT(8*RND(1)) +1
```

▲ *The instruction INT(8*RND(1))+1 will make the computer choose a different number 'at random' between 1 and 8 each time you run the program.*

called the GHOSTROOM. At line 3Ø:

```
LET _GHOSTROOM=INT(8*RND(1))+1
```

means that the computer chooses a number between 1 and 8; and, until a new game begins, that number is the number of the GHOSTROOM. Later in the program, at line 229Ø, the player has to INPUT a guess for that number. If the player guesses correctly, the game soon comes to an end and the computer will choose another number between 1 and 8 for the GHOSTROOM in the next game.

If the guess is wrong, the player has a chance to guess again. If the player guesses right first time, the WINNINGS will be 6. Every wrong guess puts the WINNINGS down by 1 point (see line 3Ø2Ø of the program). If you get down to Ø points, the ghost will eat you! So you only have six chances to guess the number of the GHOSTROOM.

Now, study the program carefully to work out the order in which things are going to happen. You will see that many GOSUBs have been used, which has split the program into chunks, each of which should be reasonably easy to understand. The main body of the program ends at line 13Ø, and then there are all the GOSUBs.

(As this is a very long program, you might want to type it in stages. It's worth remembering that you can SAVE part of your program and then LOAD it back into the computer later, when you want to continue. How to SAVE and LOAD programs is explained in the next chapter.)

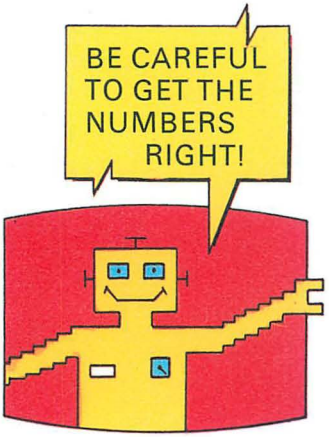
Now, type in the program, and run it:

HOLD A RULER OR A PENCIL UNDER EACH LINE AS YOU COPY IT. IT WILL HELP YOU NOT TO MISS ANY BY MISTAKE!

```
  1Ø  REM _"GHOSTHUNT"
  2Ø  GOSUB _1ØØØ:REM _
INTRODUCTION
  3Ø  LET _GHOSTROOM=INT(8*RND
(1))+1
  4Ø  LET _WINNINGS=6
  5Ø  GOSUB _2ØØØ:REM _
GHOSTHOUSE _AND _GUESS
```

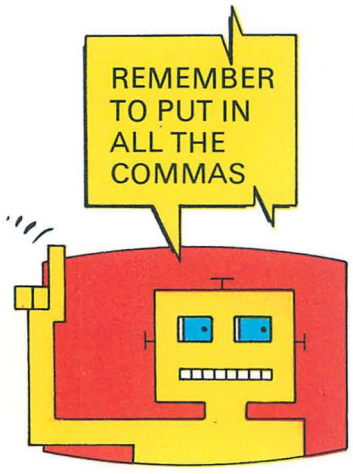
```
  6Ø  IF _G < >GHOSTROOM _THEN _
GOSUB _3ØØØ:REM _SCORE _IF _
WRONG
  7Ø  IF _WINNINGS=Ø _THEN _
GOTO _1ØØ
  8Ø  IF _G < >GHOSTROOM _THEN _
GOTO _5Ø
  9Ø  GOSUB _4ØØØ:REM _SCORE _IF _
RIGHT
 1ØØ  GOSUB _5ØØØ:REM _NEW _GAME _
OR _END
 11Ø  IF _B$="Y" _THEN _GOTO _2Ø
 12Ø  IF _B$ < >"N" _THEN _GOTO _1ØØ
 13Ø  END

1ØØØ  REM _INTRODUCTION
1Ø1Ø  POKE _53280,8:POKE _53281,7
1Ø2Ø  PRINT _"(CLR/HOME)":PRINT:
PRINT:PRINT
1Ø3Ø  PRINT _CHR$(156);"(5 spaces)
♠♥♦♣(5 spaces)GHOSTHUNT(5 spaces)
♣♦♥♠"
1Ø4Ø  RESTORE:GOSUB _6ØØØ:REM _
TUNE
1Ø5Ø  PRINT:PRINT:PRINT
1Ø6Ø  PRINT _CHR$(31);"(18 spaces)BY"
1Ø7Ø  PRINT:PRINT _"(7 spaces)THE _
GRAVES _FAMILY _(1985)"
1Ø8Ø  PRINT:PRINT:PRINT _CHR$(3Ø);
" _WHAT _IS _YOUR _NAME?"
1Ø9Ø  INPUT _A$:GOSUB _6ØØØ
11ØØ  PRINT:PRINT:PRINT _CHR$(28);
" _SOON _";A$;", _YOU _WILL _SEE"
111Ø  PRINT:PRINT:PRINT _CHR$(156);
"(7 spaces)A _HAUNTED _HOUSE"
112Ø  PRINT:PRINT:PRINT:PRINT:
RESTORE:GOSUB _6ØØØ
113Ø  PRINT _CHR$(3Ø);" _PRESS _
RETURN _TO _CONTINUE"
114Ø  INPUT _Z$:PRINT _
"(CLR/HOME)": RETURN

2ØØØ  REM _GHOSTHOUSE _AND _
GUESS
```

```
2Ø1Ø  REM _MAIN _HOUSE
2Ø2Ø  PRINT:PRINT _CHR$(31);"(8 spaces)
HERE _IS _THE _HAUNTED _HOUSE!!"
2Ø3Ø  PRINT:PRINT _"(8 spaces)♦♦
(4 spaces)IT _HAS _8 _ROOMS
(5 spaces)♦♦"
2Ø4Ø  LET _A=12:LET _B=27:LET _C=
1351:LET _D=1Ø2:LET _E=4:GOSUB _65ØØ:
REM _COLOUR _BLOCK
2Ø5Ø  REM _THE _ROOF
2Ø6Ø  LET _A=2:LET _B=21:LET _C=
1234:LET _E=9:GOSUB _65ØØ
2Ø7Ø  LET _R=1Ø2:LET _S=54272:POKE _
1273,R:POKE _1273+S,9:POKE _1312,R:
POKE _1312+S,9
2Ø8Ø  POKE _1313,R:POKE _1313+S,9:
POKE _1296,R:POKE _1296+S,9:POKE _
1336,R
2Ø9Ø  POKE _1336+S,9:POKE _1337,R:
POKE _1337+S,9
21ØØ  POKE _1311,233:POKE _1311+S,9:
POKE _1272,233:POKE _1272+S,9:POKE _
1233,233
211Ø  POKE _1233+S,9:POKE _1256,223:
POKE _1256+S,9:POKE _1297,223:POKE _
1297+S,9
212Ø  POKE _1338,223:POKE _1338+S,9
213Ø  REM _THE _WINDOWS
214Ø  LET _A=3:LET _B=4:LET _C=
1433:LET _D=127:LET _E=5:GOSUB _65ØØ
215Ø  LET _C=1439:GOSUB _65ØØ
216Ø  LET _C=1446:GOSUB _65ØØ
217Ø  LET _C=1452:GOSUB _65ØØ
218Ø  LET _A=2:LET _B=3:LET _C=1672:
LET _D=127:LET _E=5:GOSUB _65ØØ
219Ø  LET _C=1677:GOSUB _65ØØ
22ØØ  LET _C=1689:GOSUB _65ØØ
221Ø  LET _C=1694:GOSUB _65ØØ
222Ø  REM _THE _DOOR
223Ø  LET _A=4:LET _B=3:LET _C=
1683:LET _E=11:GOSUB _65ØØ
224Ø  RESTORE:GOSUB _6ØØØ
225Ø  PRINT: PRINT: PRINT:PRINT:PRINT:
PRINT:PRINT:PRINT:PRINT:PRINT
```



```
2260  PRINT:PRINT:PRINT:PRINT:
PRINT:PRINT:PRINT
2270  PRINT "WHERE IS THE
GHOST ";A$;"?"
2280  PRINT "ENTER YOUR
GUESS FROM 1 TO 8:"
2290  INPUT G:GOSUB 6000
2300  RETURN

3000  REM SCORE IF WRONG
3010  POKE 53280,4:POKE 53281,4:
PRINT "(CLR/HOME)"
3020  LET WINNINGS=WINNINGS-1
3030  PRINT CHR$(5);" BAD
LUCK ";A$
3040  PRINT:PRINT:PRINT:PRINT
3050  PRINT " THE GHOST WAS
NOT IN ROOM ";G
3060  PRINT:PRINT:PRINT:PRINT:FOR
T=1 TO 2000:NEXT T
3070  IF WINNINGS=0 THEN
PRINT " OOPS! YOU HAVE
BEEN EATEN BY THE GHOST."
3080  IF WINNINGS=0 THEN
RESTORE:GOSUB 6000
3090  IF WINNINGS=0 THEN PRINT:
PRINT:PRINT " PLEASE PRESS
RETURN":GOTO 3140
3100  PRINT " YOUR POSSIBLE
SCORE IS NOW DOWN FROM
6 TO ";WINNINGS
3110  PRINT:PRINT:PRINT:PRINT:
RESTORE:GOSUB 6000
3120  PRINT " PRESS RETURN
WHEN YOU ARE BRAVE
ENOUGH"
3130  PRINT " TO HAVE
ANOTHER TRY."
3140  IF WINNINGS=0 THEN LET
G=GHOSTROOM
3150  INPUT Z$:PRINT "(CLR/HOME)":
POKE 53281,7:RETURN

4000  REM SCORE IF RIGHT
```

```
4Ø1Ø  PRINT:PRINT:PRINT_"(CLR/
HOME)":PRINT_"(7 spaces)WELL_
DONE_";A$;"!"
4Ø2Ø  PRINT:PRINT:PRINT_"(4 spaces)
THE_GHOST_WAS_IN_ROOM_";G
4Ø3Ø  RESTORE:GOSUB_6ØØØ
4Ø4Ø  POKE_5328Ø,2:POKE_53281,7
4Ø5Ø  PRINT:PRINT:PRINT_"(4 spaces)
YOUR_SCORE_THIS_TIME_IS_";
WINNINGS
4Ø6Ø  PRINT:PRINT:PRINT:PRINT:
PRINT:PRINT:PRINT:PRINT_"(4 spaces)
PLEASE_PRESS_RETURN"
4Ø7Ø  INPUT_Z$:GOSUB_6ØØØ
4Ø8Ø  PRINT_"(CLR/HOME)":RETURN

5ØØØ  REM_NEW_GAME_OR_END
5Ø1Ø  POKE_5328Ø,4:POKE_53281,14:
PRINT_"(CLR/HOME)":PRINT:PRINT
5Ø2Ø  PRINT_CHR$(144);"_IF_
YOU_WANT_ANOTHER_GAME_OF_
GHOSTHUNT"
5Ø3Ø  PRINT:PRINT_"_THEN_ENTER_
Y_FOR_YES_OR_N_FOR_NO"
5Ø4Ø  INPUT_B$:PRINT_"(CLR/
HOME)":RETURN

6ØØØ  REM_TWO_TUNES
6Ø1Ø  POKE_5328Ø,5
6Ø2Ø  POKE_54296,12:POKE_54277,15Ø
6Ø3Ø  POKE_54278,128
6Ø4Ø  FOR_S=1_TO_1Ø
6Ø5Ø  READ_A,B,C
6Ø6Ø  POKE_54273,A
6Ø7Ø  POKE_54272,B
6Ø8Ø  POKE_54276,17
6Ø9Ø  FOR_T=1_TO_C:NEXT_T
61ØØ  POKE_54276,Ø
611Ø  NEXT_S
612Ø  POKE_54277,Ø:POKE_54278,Ø
613Ø  DATA_25,177,3ØØ,27,56,3ØØ,28,214,
6ØØ,27,56,3ØØ,38,126,15Ø,34,75,15Ø
614Ø  DATA_32,94,15Ø,3Ø,141,6ØØ,34,
75,6ØØ,36,85,6ØØ
```



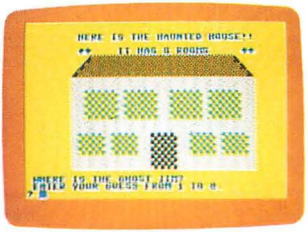
```
615Ø  DATA_17,37,15Ø,21,154,45Ø,25,177,
15Ø,21,154,45Ø,17,37,15Ø,21,154,225
616Ø  DATA_22,227,75,21,154,15Ø,19,63,15Ø,
17,37,45Ø
617Ø  POKE_5328Ø,8
618Ø  RETURN

65ØØ  REM_COLOUR_BLOCK
651Ø  FOR_Y=Ø_TO_A
652Ø  FOR_X=Ø+Y*4Ø_TO_B+Y*4Ø
653Ø  POKE_X+C,D
654Ø  POKE_X+C+54272,E
655Ø  NEXT_X
656Ø  NEXT_Y
657Ø  RETURN
```

Run the program a few times to enjoy playing the game. Now think about whether you can improve it.

1) Can you improve the interest of the game? For example, you might want to add lines which will tell you if your guess was more or less than the right answer, as you did for the earlier program "guess me".You could fit this in between lines 3Ø3Ø and 3Ø5Ø, where there are four blank lines on the screen.

2) Can you improve the small details of the program? For example, you might want to change some of the sounds, or the colours of the house.

3) Can you improve the overall structure of the game? For example, you might decide that using a FOR...NEXT loop would be a better way of ensuring that the player is allowed only six guesses.



▲ *When you have run the program a few times, think about how you could improve it.*

A reminder

You have learned how to type graphic characters directly into programs. You have also used long variable names. You have learned how to use RESTORE with READ...DATA instructions. Most importantly, you have learned how to use the instruction PRINT INT(a number)*RND(1))+1. This is one of the most useful instructions when you are inventing games.

8 Saving and Loading Programs

Saving programs

You have learned to give each program a title in the first line, using the statement REM followed by no more than 16 letters and/or numbers and/or spaces typed between sets of inverted commas. For example, the first line of the program for thank-you letters was:

```
1Ø REM _"THANK _YOU"
```

To save the "THANK YOU" program on tape, you must:

1) Type the list of program lines for "THANK YOU" into the computer.

2) Type:

```
SAVE _"THANK _YOU"
```

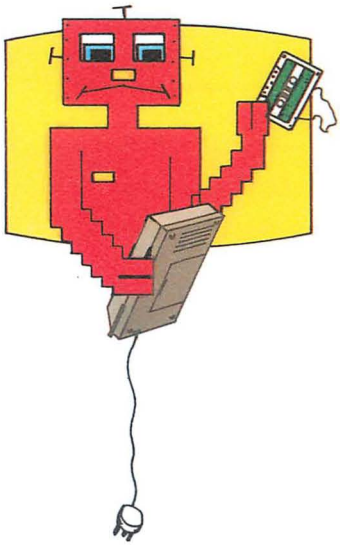
3) Press RETURN, and at once the following message will appear:

```
PRESS RECORD & PLAY ON TAPE
```

4) Press the PLAY and RECORD levers on your tape recorder. At once the central area of the screen will go blank, and will change to the colour of the border area.

▶ Your programs can be saved onto a tape cassette so that you do not lose them when you switch the computer off, or start a new program.

5) After a while, the words on the central area of the screen will reappear, followed by the usual prompt, 'READY', to tell you that the computer has saved your program and is waiting for your next instruction.



Checking that a program has been properly saved

1) Rewind the tape to just before the start of where you saved the "THANK YOU" program.
2) Type:

```
VERIFY_"THANK_YOU"
```

3) Press RETURN, and at once you will see:

```
PRESS PLAY ON TAPE
```

4) Press the PLAY lever on your tape recorder. The central area of the screen will go blank and change to the colour of the border.
5) After a while, the words on the central area of the screen should reappear, followed by the messages:

```
OK

SEARCHING FOR THANK YOU
FOUND THANK YOU
```

Press the RUN/STOP key to stop the verifying procedure. Your program has been saved on tape, and you can now switch off the computer without losing it.

Loading programs

To load your program back into the computer:
1) Rewind the tape to just before the start of where you saved the "THANK YOU" program.
2) Type:

```
LOAD_"THANK_YOU"
```

▲ *When you want to use a program you have saved onto tape, you use these instructions to play it back into your computer.*

3) Press RETURN, and at once the message will appear:

```
PRESS PLAY ON TAPE
```

4) Press the PLAY lever on your tape recorder. The central area of the screen will go blank and change to the colour of the border. After a few seconds the words on the central area of the screen should reappear, followed by the messages:

```
OK

SEARCHING FOR THANK YOU
FOUND THANK YOU
```

5) Press the C= (Commodore) key, which is at the bottom left of the keyboard. The screen will go blank just as before. After a while, the words on the central area of the screen should reappear, followed by the messages:

```
LOADING
READY.
```

The program is now loaded and ready to be run.

Some common problems with verifying or loading

When you try to verify or load programs you may find that after you have pressed PLAY on your tape recorder, the screen stays blank. When you are certain that nothing is going to appear press the RUN/STOP key. If you have been trying to VERIFY then you will see:

```
OK

SEARCHING FOR THANK YOU
FILE NOT FOUND  ERROR
READY.
```

And if you have been trying to LOAD you will see:

```
SEARCHING FOR THANK YOU
?BREAK  ERROR
READY.
```

What could have gone wrong?

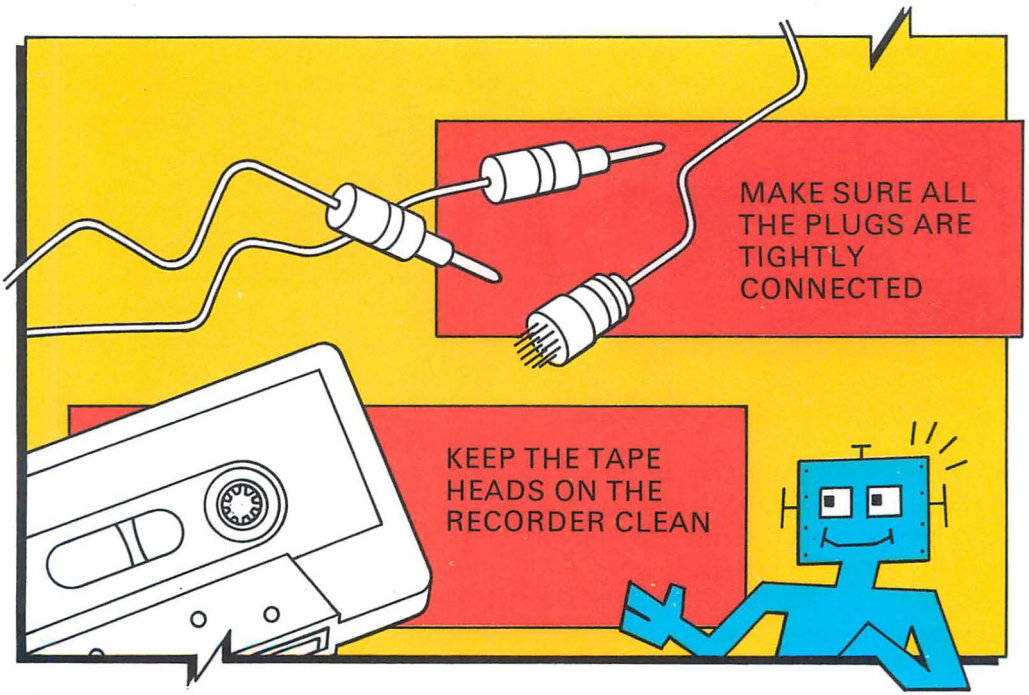
1) The recorder may be dirty, in which case play a cleaning tape.

2) You may have started the tape in the middle of the program, instead of just before the beginning of the program, so rewind the tape and try again.

3) You may have made a mistake in typing the name of the program which you are trying to verify or load. You will find this out, as a message will appear on the screen saying, for example:

```
OK

SEARCHING FOR THANK YOU
FOUND THNK YOU
```

Try again, using the name the computer has found.



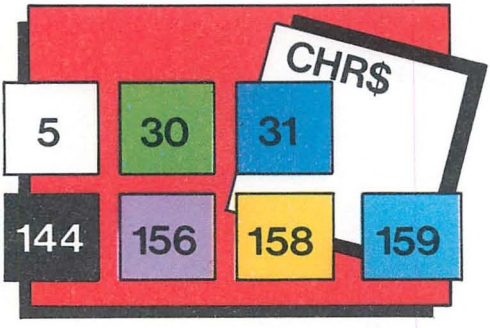
9 A List of Programming Words

Here you will find all the programming words which you have learned. Look at this chapter carefully, as you will find a few new ways of doing things.

**CHR$** This instruction, followed by a number in brackets, is used to produce coloured text. CHR$(5) produces white text; 28 red; 3Ø green; 31 blue; 144 black; 156 purple; 158 yellow; 159 cyan. CHR$(18) turns on 'reverse type'; which means that:

PRINT CHR$(18);"_____"

will not print five blank spaces, but five squares filled with whatever is the current colour of the text. (See also pages 58–6Ø of the *User Manual.*)



CONT This instruction is used to make a program CONTinue, after it has been stopped either by your pressing the RUN/STOP key, or by the computer reaching an END or STOP statement within a program. (See also page 114 of the *User Manual.)*

DATA Lines beginning with DATA hold either numbers or words which can be brought into the program by READ statements. See READ below.

END This statement marks the end of a program. (See also page 119 of the *User Manual.*)

FOR This is the first instruction in a FOR...NEXT loop. It makes the computer carry out a task a particular number of times. Thus:

```
1Ø  FOR A=1 TO 6
2Ø  PRINT A
3Ø  NEXT A
```

will print (on separate lines) the numbers 1, 2, 3, 4, 5 and 6. (See also pages 39–4Ø and 119 of the *User Manual.*)

GOSUB This is a very useful way of keeping a program tidy and easy to follow. For example:

GOSUB 1ØØØ

will send the computer to line 1ØØØ and all the lines which follow, until the instruction RETURN is reached. It will then jump back to the next line in the program immediately after the GOSUB 1ØØØ from which it set out. So a single GOSUB line in the main program can be used to call up the same set of lines again and again. (See also page 12Ø of the *User Manual.*)

GOTO This makes the computer jump to a certain line in the program. It should not be used too often in a program. (See also pages 32–33 and 12Ø of the *User Manual.*)

IF This is used at the start of an IF...THEN statement to say that IF something is true, THEN the computer must do a certain thing. For example:

```
IF A=B THEN PRINT
"RIGHT!"
```

(See also pages 37–39 and 12Ø–121 of the *User Manual.*)

INPUT This instruction is used to put information into the computer:

1) It can enter numbers. For example:

```
5Ø  PRINT "ENTER THE
POCKET MONEY YOU WILL
GET"
6Ø  INPUT B
```

When you type in the amount, for example, 15, then press RETURN, the number 15 will be placed in a store marked B; and if in future you give the instruction:

```
PRINT B
```

then the number 15 will appear on the screen as soon as you have pressed RETURN. The two lines above could also be condensed into a single line, like this:

```
5Ø  INPUT "ENTER THE
POCKET MONEY YOU WILL
GET ";B
```

2) It can also enter words, as in:

```
21Ø  PRINT "TYPE YOUR
SIGNATURE"
22Ø  INPUT F$
```

When you type in your name, for example Philippa, then the word 'Philippa' will be placed in a store marked F$; and if in future you give the instruction:

```
PRINT F$
```

then the word 'Philippa' will appear on the screen as soon as you have pressed RETURN. These two lines could also be condensed to:

```
21Ø  INPUT "TYPE YOUR
SIGNATURE ";F$
```

(See also pages 45–47 and 121 of the *User Manual.*)

INT See RND below.

LET This instruction is used to store information in the computer.

1) It can store numbers. For example, on page 15 you used:

```
LET A=2Ø
```

When you did that, the number 2Ø was stored in the computer under the label A. If you wanted to add 6 to that store, you could either say:

```
LET A=26
```

or, you could add another line to the program saying:

```
LET A=A+6
```

You could also use a statement like:

```
LET BIRTHDATE=1969
```

2) It can store words. For example, you might have a line saying:

```
LET A$="Pocketwatch"
```

This statement enters the word 'Pocketwatch' in a store labelled A$. (See also page 121 of the *User Manual.*)

LIST On its own, this statement will make the computer list out whatever program it has in its memory. If you want to see only one line, type LIST followed by the number of that line. To see a particular group of lines, type LIST followed by the first and last lines that you wish to see, separated by a dash, like this:

```
LIST 4Ø-6Ø
```

(See also page 115 of the *User Manual.*)

LOAD This instruction is used when loading a program into the computer. See pages 65–66. (See also pages 18–21 and 115 of the *User Manual.*)

NEW When you type NEW and press RETURN, any program lines and any variables stored in the computer's

memory will be rubbed out. (See also page 115 of the *User Manual.*)

NEXT See FOR above.

POKE This is a useful instruction which allows you to push or 'POKE' a number into the computer's memory, and this can do several different jobs, such as:

1) Changing the colour of the border of the screen, using POKE 53280 followed by a comma, followed by 0 for black; 1 for white; 2 for red; 3 for cyan; 4 for purple; 5 for green; 6 for blue; 7 for yellow; 8 for orange; 9 for brown; 10 for light red; 11 for grey 1; 12 for grey 2; 13 for light green; 14 for light blue; 15 for grey 3.

2) Changing the colour of the background of the central area of the screen, using POKE 53281 followed by a comma, followed by one of the colour numbers above.

3) Placing a graphic character on the screen, using POKE followed by the number of the place on the screen, followed by a comma, followed by the number of the graphic character. (For a screen map see page 63, and for the graphic characters pages 132–134, of the *User Manual.*)

4) Changing the colour of a graphic character which has been 'POKEd' onto the screen, by using POKE followed by the number of the place on the screen plus 54272, followed by a comma, followed by one of the colour numbers given above.

5) Making sounds, using a number of POKE instructions as explained in Chapter 6.

(See also pages 60–66, 80–90, 123, and 138–139 of the *User Manual.*)

PRINT This instruction is usually used to print something on the screen. For example:

```
PRINT "BOOK"
```

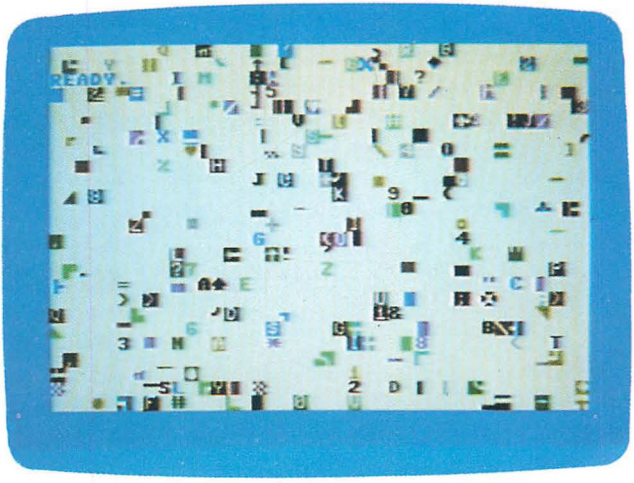
makes the computer print on the screen whatever is typed between the inverted commas, in this case BOOK.

```
PRINT
```

on its own produces an empty line.

```
PRINT A
```

will print the number previously placed in the store labelled A.



▶ *By using the POKE instruction as described in chapter four, you can make coloured shapes and letters appear on the screen.*

PRINT A$

will print the letters previously placed in the store labelled A$. (See also pages 22–29 and 123–124 of the *User Manual.*)

READ This statement, followed by a number or string variable, tells the computer that it must take the next item of information from a line of DATA and give it to the variable. Thus:

40 READ A

will take the first number from a line of DATA beginning:

110 DATA 7,23,45

and A will now be equal to 7. If a FOR...NEXT loop is used, or a GOTO statement is used to create a loop, the next time the computer reaches line 40, A will be equal to 23; and the next time, 45. Using the same line of DATA, but changing the READ statement to:

40 READ A,B,C

would make A=7, B=23 and C=45. (See also pages 92–94 and 124 of the *User Manual.*)

REM The computer ignores everything in a line after a REM statement, so REM statements are very useful for reminding yourself what a program is about. For example, in "GHOSTHUNT" you had:

1000 REM INTRODUCTION
2000 REM GHOSTHOUSE AND GUESS

and so on. REM may also be used to give a program a title, so that you can SAVE and LOAD it. For example:

10 REM "GHOSTHUNT"

(See also page 124 of the *User Manual.*)

RESTORE Normally, a READ statement will be instructed to work its way through all the lines of DATA in a program, in order. But as soon as the instruction RESTORE is given, the READ statement must turn back to the very first item of DATA, and begin again. This means that lines of DATA can be read more than once. (See also page 124 of the *User Manual.*)

RETURN See GOSUB above.

RND This instruction makes the computer choose a number at RaNDom. PRINT RND followed by a number in brackets, such as:

PRINT RND(9)

will print a random number between 0 and 1.

PRINT INT(6*RND(1))

will print a random number between 0 and 6 (but will never reach 6).

PRINT INT(6*RND(1))+1

will print a whole number at random between 1 and 6. (See also pages 48–53 and 126–127 of the *User Manual.*)

RUN This instruction tells the computer to work its way through the program which is in the computer's memory. (See also page 116 of the *User Manual.*)

SAVE This is used when saving a program on tape. See pages 64–65. (See also pages 21–22 and 116 of the *User Manual.*)

VERIFY This is used to check that a program has been properly saved onto tape. See page 65. (See also page 117 of the *User Manual.*)

Index

Some words such as PRINT are on many pages. So as not to waste your time, this Index lists only the most useful examples.

BEGINNER'S GUIDES

Richard Graves, a qualified teacher, and his son Philip, found that many of the standard books on introducing programming were far too complicated. Together, they have produced a guide to simple programming which can be used by anyone — of any age.

Beginner's Guide to the BBC Micro:
'. . . a valuable book for the beginner.' *Personal Computer News*

'. . . commendable attention to detail and programming style that sets the beginner off in the right direction.' *The Times Educational Supplement*

Guide to the ZX Spectrum:
'. . . one of the most colourful, beautifully produced books . . . at an excellent pocket money price.' *The Bookseller*

Guide to the Acorn Electron:
'. . . you can't go wrong for just a simple, plain English introduction to learning the basics on your Electron.' *Personal Computer News*

'. . . well laid-out pages and first class artwork . . . excellent value for money . . .' *Microchamp*

Books in the series:
● BBC Micro ● ZX Spectrum ● Acorn Electron ● Commodore 64

ISBN 0-86272-160-1

£3.50
NET UK ONLY