A MARSHALL CAVENDISH 2 COMPUTER COURSE IN WEEKLY PARTS

INPUT

LEARN PROGRAMMING - FOR FUN AND THE FUTURE

FREE WITH PART 1

~~UK £1.00~~
~~Republic of Ireland £1.25~~
~~Malta 85c~~
~~Australia $2.25~~
~~New Zealand $2.95~~

INPUT

Vol 1 No 2

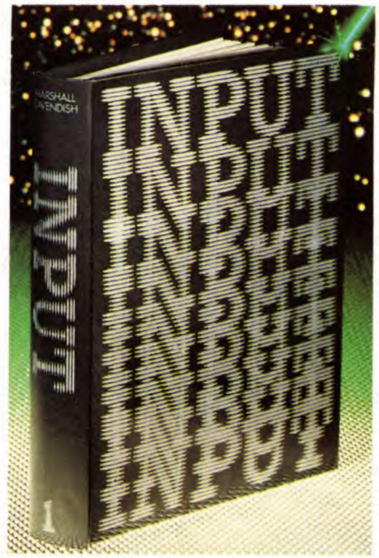
INDEX

The last part of INPUT, Part 52, will contain a complete, cross-referenced index. For easy access to your growing collection, a cumulative index to the contents of each issue is contained on the inside back cover.

PICTURE CREDITS

Front cover: Pete Seaward Pages 34, 37, Chen Ling Pages 38, 45, Phil Dobson Pages 46-53, Pete Seaward Pages 54-59, Dick Ward Pages 60-63, Andrew MacConville

Published by Marshall Cavendish Partworks Ltd, 58 Old Compton Street, London W1V 5PA, England. Printed by Artisan Presss, Leicester and Howard Hunt Litho, London.



There are four binders each holding 13 issues.

HOW TO ORDER YOUR BINDERS

UK and Republic of Ireland: Send £4.95 (inc p & p) (IR£5.45) for each binder to the address below:
Marshall Cavendish Services Ltd, Department 980, Newtown Road, Hove, Sussex BN3 7DN

Australia: See inserts for details, or write to INPUT, Gordon and Gotch Ltd, PO Box 213, Alexandria, NSW 2015

New Zealand: See inserts for details, or write to INPUT, Gordon and Gotch (NZ) Ltd, PO Box 1595, Wellington

Malta: Binders are available from local newsagents.

BACK NUMBERS

Copies of any part of INPUT can be obtained from the following addresses at the regular cover price, with no extra charge for postage and packing:

UK and Republic of Ireland:
INPUT, Dept AN, Marshall Cavendish Services, Newtown Road, Hove BN3 7DN

Australia, New Zealand and Malta:
Back numbers are available through your local newsagent

COPIES BY POST

Our Subscription Department can supply your copies direct to you regularly at £1.00 each. For example the cost of 26 issues is £26.00; for any other quantity simply multiply the number of issues required by £1.00. These rates apply anywhere in the world. Send your order, with payment to:
Subscription Department, Marshall Cavendish Services Ltd, Newtown Road, Hove, Sussex BN3 7DN
Please state the title of the publication and the part from which you wish to start.

HOW TO PAY: Readers in UK and Republic of Ireland: All cheques or postal orders for binders, back numbers and copies by post should be made payable to:
Marshall Cavendish Partworks Ltd.

QUERIES: When writing in, please give the make and model of your computer, as well as the Part No., page and line where the program is rejected or where it does not work. We can only answer specific queries – and please do not telephone. Send your queries to INPUT Queries, Marshall Cavendish Partworks Ltd, 58 Old Compton Street, London W1V 5PA.

INPUT IS SPECIALLY DESIGNED FOR:

The SINCLAIR ZX SPECTRUM (16K, 48K, 128 and +), COMMODORE 64 and 128, ACORN ELECTRON, BBC B and B+, and the DRAGON 32 and 64.

In addition, many of the programs and explanations are also suitable for the SINCLAIR ZX81, COMMODORE VIC 20, and TANDY COLOUR COMPUTER in 32K with extended BASIC. Programs and text which are specifically for particular machines are indicated by the following symbols:

SPECTRUM 16K, 48K, 128, and +
COMMODORE 64 and 128
ACORN ELECTRON, BBC B and B+
DRAGON 32 and 64
ZX81
VIC 20
TANDY TRS80 COLOUR COMPUTER

THE COMPUTER AS DECISION-MAKER

- CHOOSING THE WAY TO GO
- WORKING OUT AVERAGES
- MORE COMPLICATED DECISIONS
- A FRUIT MACHINE GAME
- USING **IF . . . THEN . . . ELSE**

Brainless though it might be, your computer can still make logical decisions—if you program it the right way. Here's how to use IF . . . THEN to turn your computer into a decision-maker.

One of the things that makes a computer superior to an ordinary calculator is its ability to make decisions.

This useful feature allows a program to branch off in different directions—and hence to carry out different instructions—depending on the outcome of a particular test.

One of the ways the computer does this is by means of the IF . . . THEN statement. It acts on such a statement in much the same way that a human being would do: IF so-and-so is true, THEN it will do such-and-such.

One example of this statement has been seen on page 3. Another example is:

IF A<18 (in other words, if A is less than 18) THEN PRINT "underage"

When the computer meets the keyword IF, it checks whether the next statement is true. If it is, the computer carries on and does whatever comes after the word THEN. If, on the other hand, it is not true, the computer ignores that line and goes on to the next line of the program instead.

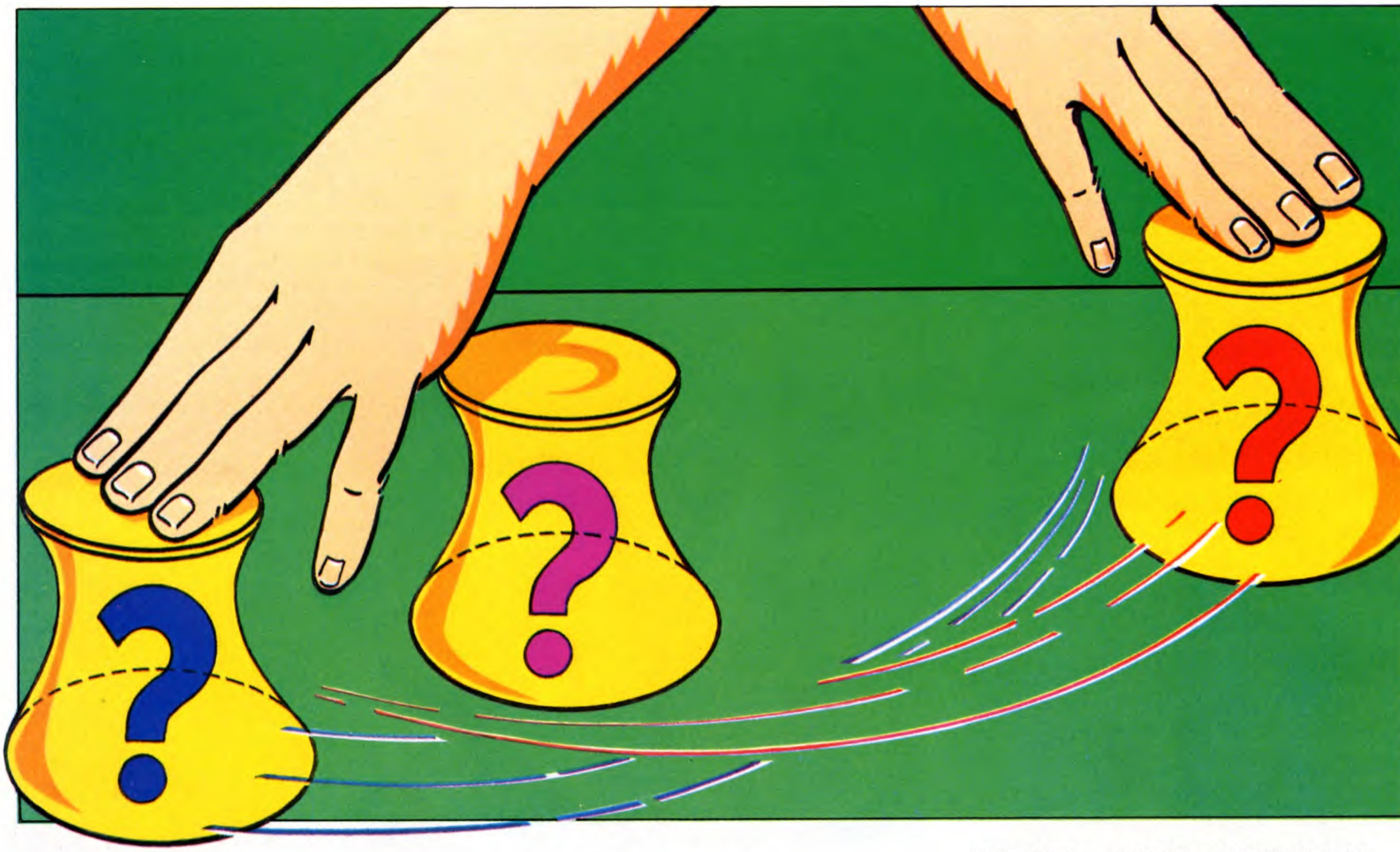
JUST ABOUT AVERAGE

You can see how it works in this next program which works out the average of a list of marks:

On the ZX81, omit :STOP in Line 40 and add

```
45 IF N=-99 THEN GOTO 80
80 STOP
```

```
10 PRINT "ENTER LIST OF MARKS"
20 PRINT "TYPE -99 TO END THE LIST"
25 LET T=0
26 LET C=0
30 INPUT N
40 IF N=-99 THEN PRINT "AVERAGE
   MARK=";T/C: STOP
50 LET T=T+N
60 LET C=C+1
70 GOTO 30
```



```
1Ø PRINT "ENTER LIST OF MARKS"
2Ø PRINT "TYPE –99 TO END THE LIST"
3Ø INPUT N
4Ø IF N= –99 THEN PRINT "AVERAGE
   MARK=";T/C: END
5Ø LET T=T+N
6Ø LET C=C+1
7Ø GOTO 3Ø
```

The instructions tell you to type in a list of marks and then to type –99 to end the list. You'll see the reason for this in a moment. Lines 25 and 26 (necessary only on the Spectrum) set the initial values of the total and the counter to zero. Line 3Ø takes the number that you type in, Line 5Ø adds it to the running total and Line 6Ø keeps track of how many numbers you have entered—by adding 1 for each mark that is input.

As long as you type in real marks, the computer will ignore Line 4Ø and will go back to Line 3Ø for another number. But when you type –99 the condition in Line 4Ø, N= –99, is true, so the computer prints out the average mark (T/C, or total divided by count) and the program ends.

Numbers like –99 are called *dummy* or *terminating numbers* and they are a useful way of controlling what happens in a program.

THREE-WAY CHOICE

What if you want to choose between three or more alternatives in order to send the computer on different courses of action? This is just as easy as choosing between two as you can see in this elaboration of the guessing game in the earlier article on page 3:

```
5 CLS
1Ø LET N=RND(2Ø)
2Ø PRINT "I'VE JUST THOUGHT OF A
   NUMBER"
3Ø PRINT "... CAN YOU GUESS WHAT IT
   IS?"
4Ø INPUT G
5Ø IF G=N THEN PRINT "CORRECT, WELL
   DONE" : FOR D=1 TO 2ØØØ : NEXT D:
   GOTO 1Ø
6Ø IF G<N THEN PRINT " TOO LOW, TRY
   AGAIN"
7Ø IF G>N THEN PRINT " TOO HIGH, TRY
   AGAIN"
8Ø GOTO 4Ø
```

```
5 CLS
1Ø LET N=INT (RND*2Ø)+1
2Ø PRINT "I'VE JUST THOUGHT OF A
   NUMBER"
3Ø PRINT "... CAN YOU GUESS WHAT IT
   IS?"
4Ø INPUT G
5Ø IF G=N THEN PRINT "CORRECT, WELL
   DONE": PAUSE 1ØØ: GOTO 1Ø
6Ø IF G<N THEN PRINT "TOO LOW, TRY
   AGAIN"
7Ø IF G>N THEN PRINT "TOO HIGH, TRY
   AGAIN"
8Ø GOTO 4Ø
```

```
5 PRINT "♥"
1Ø LET N=INT(RND(1)*2Ø+1)
2Ø PRINT "I'VE JUST THOUGHT OF A
   NUMBER"
3Ø PRINT ".... CAN YOU GUESS WHAT IT
   IS?"
4Ø INPUT G
5Ø IF G=N THEN PRINT "CORRECT, WELL
   DONE!":FOR D=1 TO 1ØØØ:NEXT D:
   GOTO 5
6Ø IF G<N THEN PRINT "TOO LOW, TRY
   AGAIN"
7Ø IF G>N THEN PRINT "TOO HIGH, TRY
   AGAIN"
8Ø GOTO 4Ø
```

Line 1Ø chooses a random number between 1 and 2Ø, then Lines 2Ø to 4Ø ask you to guess what it was. Whatever you guess, the computer will check through Lines 5Ø, 6Ø and 7Ø looking for a condition that is true.

Suppose, for example, your guess is too low. In this case the computer will look at Line 5Ø but as G = N is false it ignores that line and goes to Line 6Ø. Here the condition is true, since G is less than N. So it prints out the message 'TOO LOW, TRY AGAIN'. It then naturally goes on to the next line but this condition is false so it ignores the line and goes to Line 8Ø. Line 8Ø simply takes you back for another guess.

What if your guess was too high, or just right? Study the program until you follow exactly what is going on.

This program works very well but it has one disadvantage—it keeps on asking you to guess a number whether you want to keep playing or not. A better way would be to get the computer to ask if you wanted another go.

The next few lines do just that. Again they use IF ... THEN and in this case the computer is comparing letters rather than numbers to see if they are the same.

Microtip

Problems with operators?

If you are not used to the 'greater than' and 'less than' symbols—the *operators*—you may find them confusing at first.

So think of them as wedges. In >, the first, wide, end is *greater than* the pointed end. In <, the first, narrow, end is *less than* the wider end. So A>B reads 'A is greater than B'. Adding = just means A can also be equal to B.

Here is a full list of all the different combinations:

A=B: A equals B
A>B: A greater than B
A<B: A less than B
A>=B: A greater than or equal to B
A<=B: A less than or equal to B
A<>B: A not equal to B

On Acorn computers, you must type in the operators in the order shown. If, for example, you typed A=<B, the computer would not understand your meaning and would report an error.

On the Spectrum and ZX81, composite operators—like <=, for example—must be entered from a single key. If you tried entering < followed by =, the line would not be accepted.

```
1ØØ PRINT "DO YOU WANT ANOTHER GO?
    (Y/N)"
11Ø LET A$=GET$
12Ø IF A$="Y" THEN RUN
13Ø END
```

```
1ØØ PRINT "DO YOU WANT ANOTHER
    GO?(Y/N)"
11Ø LET A$=INKEY$ : IF A$="" THEN
    GOTO 11Ø
12Ø IF A$="Y" THEN RUN
```

```
1ØØ PRINT "DO YOU WANT ANOTHER GO?
    (Y/N)"
11Ø GET A$: IF A$="" THEN 11Ø
12Ø IF A$="Y" THEN RUN
13Ø END
```

If you want to use these extra lines, you must also change Line 5Ø of the last program to:

```
5Ø IF G=N THEN PRINT "CORRECT, WELL
    DONE": GOTO 1ØØ
```

Here, Line 11Ø waits for you to press a key. If capital Y is pressed the program will automatically RUN, but if any other key (including lower-case y!) is pressed it will stop.

These lines are very handy to add to the end of any games or quiz program to give a neat way out.

DOUBLE CHECKING

Sometimes you want the computer to test whether two or more conditions are true before deciding which way to go. One way it can do this is to use special keywords or symbols called *operators*. Look at this program line:

```
1ØØ IF D$="SATURDAY" AND T=1745
    THEN PRINT "ITS TIME FOR DR WHO"
```

When you use the keyword AND between two conditions then both conditions have to be true for the computer to carry on and do the rest of the line; otherwise it goes on to the next line in the program. In this example, it has to be Saturday AND the time has to be 1745 before the computer will print out the appropriate message.

Another example is:

```
2ØØ IF P$="SAGO" OR P$=
    "TAPIOCA" THEN PRINT "I'M NOT
    HUNGRY TODAY"
```

This line uses the keyword OR and the computer PRINTs out the sentence as long as at least one condition is true.

Q+A

To save program space, can I combine two or more IF ... THEN statements into one line?

Usually, this is a bad idea. The principle on which IF ... THEN works in BASIC is that, if the condition set out in the line is true, then the computer will execute that part of the line that comes after the THEN. But if the condition is not true, the computer ignores the rest of the line. So in this line:

```
7Ø IF X=Y THEN PRINT "OUT OF TIME":
    LET lives=lives-1: GOTO 3Ø
```

no instructions after the letter Y will be carried out unless X does equal Y.

Sometimes, though, compound IF ... THENs are useful. In these lines:

```
7Ø IF X=Y THEN PRINT "OUT OF TIME":
    IF lives>Ø THEN LET lives=
    lives-1
8Ø IF X=Y AND lives=Ø THEN PRINT
    "Game over"
```

the player will lose a life only if he has one left. But because of the way Line 7Ø is structured, the 'out of time' warning will be PRINTed regardless.

The test can get very complicated if there are a lot of conditions to check. If you have several ANDs and ORs together in one line then you should use brackets so the computer knows which to check first.

For example, a line in an adventure game may look like this:

```
2ØØØ IF P=14 AND (C$="SWORD" OR
    C$="KNIFE") THEN PRINT "YOU'VE
    KILLED THE GREMLIN"
```

This condition is only true—and you get to kill the gremlin—if you are at position 14 AND you are carrying either a sword OR a knife. But try changing the brackets to this:

```
2ØØØ IF (P=14 AND C$="SWORD") OR
    C$="KNIFE" THEN PRINT "YOU'VE
    KILLED THE GREMLIN"
```

This is true if you are at position 14 with a sword, OR you are anywhere and just carrying a knife—which is not the same thing at all.

Brackets are essential to make the computer do exactly what you want if certain priorities have to be observed.

WINNING THE JACKPOT

Here is a program to play a fruit machine game which makes good use of AND and OR. See if you can win the jackpot:

```
20 LET M=5Ø
3Ø CLS
4Ø LET M=M-5
5Ø IF M<Ø THEN PRINT "SORRY, YOU'RE
   BROKE": STOP
6Ø LET A=INT (RND*12)+13Ø
7Ø LET B=INT (RND*12)+13Ø
8Ø LET C=INT (RND*12)+13Ø
21Ø PRINT PAPER Ø; INK 4;AT 1Ø,14;CHR$
   A;AT 1Ø,16;CHR$ B;AT 1Ø,18;CHR$ C
22Ø IF A=B AND B=C THEN PRINT AT
   13,2;"YOU'VE HIT THE JACKPOT....
   5Ø": LET M=M+5Ø
23Ø IF (A=B OR B=C) AND A<>C THEN
   PRINT AT 13,9;"YOU'VE WON $1Ø": LET
   M=M+1Ø
24Ø PAUSE 25
25Ø PRINT AT 15,8;"ANOTHER GO? (y/n)":
   PRINT TAB 1Ø;"YOU HAVE $";M
26Ø IF INKEY$="" THEN GOTO 26Ø
27Ø IF INKEY$="n" THEN STOP
28Ø GOTO 3Ø
```

Q+A

Why do I keep getting error reports when I RUN typed-in programs?

There may be bugs in the programs themselves—but a far more common cause is simple typing errors which often creep in when you are copying. Here are some common ones:

- Confusing capital I or lower-case l with the numeral 1
- Confusing capital O with numeral Ø
- Omitting the quotation marks at the end of a PRINT statement
- In a DATA statement, omitting the comma between two numbers (this may well produce a number too big for the computer to accept)
- Omitting a minus sign (in any program which generates graphics, this is likely to tell the computer to print something 'out of screen')
- Omitting the semi-colon at the end of a line (this will create havoc with your screen display)

```
2Ø LET M=5Ø
3Ø CLS
4Ø LET M=M-5
5Ø IF M<Ø THEN PRINT "SORRY, YOU'RE
   BROKE":END
6Ø LET A=RND(12)+192
7Ø LET B=RND(12)+192
8Ø LET C=RND(12)+192
21Ø PRINT@ 237,CHR$(A):PRINT@
239,CHR$(B):PRINT@ 241,CHR$(C)
22Ø IF A=B AND B=C THEN PRINT@258,
   "YOU'VE HIT THE JACKPOT....$5Ø":
   LET M=M+5Ø
23Ø IF (A=B OR B=C) AND A<>C THEN
   PRINT@ 265,"YOU'VE WON $1Ø":LET
   M=M+1Ø
24Ø FOR D=1 TO 5ØØ:NEXT
25Ø PRINT@ 327,"ANOTHER GO? (Y/N)":
   PRINT@ 361,"YOU HAVE $";M
26Ø LET K$=INKEY$:IF K$="" THEN GOTO
   26Ø
27Ø IF K$="Y" THEN GOTO 3Ø
28Ø END
```

```
2Ø LET M=5Ø
3Ø CLS
4Ø LET M=M-5
5Ø IF M<Ø THEN PRINT "SORRY, YOU'RE
   BROKE":END
6Ø LET A=RND(12)+224 (32 for Electron)
7Ø LET B=RND(12)+224 (32 for Electron)
8Ø LET C=RND(12)+224 (32 for Electron)
21Ø PRINT TAB(17,1Ø);CHR$147;CHR$A;
   "□";CHR$B;"□";CHR$C
22Ø IF A=B AND B=C THEN PRINT
   TAB(7,12) "YOU'VE HIT THE
   JACKPOT....$5Ø": LET M=M+5Ø
23Ø IF (A=B OR B=C) AND A<>C THEN
   PRINT TAB(14,12) "YOU'VE WON $1Ø":
   LET M=M+1Ø
24Ø FOR D=1 TO 15ØØ: NEXT
25Ø PRINT TAB(13,16) "ANOTHER GO?
   (Y/N)";TAB(15,17) "YOU HAVE $";M
26Ø LET K$=GET$
27Ø IF K$="Y" THEN GOTO 3Ø
28Ø END
```

On the Vic, change Line 1Ø to 1Ø POKE 36879,8. Change TAB(15) in Line 21Ø to TAB(3). Omit TAB(5) in Line 22Ø, and change TAB(13) in Line 23Ø to TAB(4).

```
1Ø POKE 5328Ø,Ø: POKE 53281,Ø:PRINT
   CHR$(3Ø)
2Ø LET M=5Ø
3Ø PRINT "♥"
4Ø LET M=M-5
5Ø IF M<Ø THEN PRINT "SORRY YOU'RE
   BROKE": END
6Ø LET A=INT(RND(1)*4)+1
7Ø LET B=INT(RND(1)*4)+1
8Ø LET C=INT(RND(1)*4)+1
9Ø IF A=1 THEN LET A=97
1ØØ IF A=2 THEN LET A=115
11Ø IF A=3 THEN LET A=12Ø
12Ø IF A=4 THEN LET A=122
13Ø IF B=1 THEN LET B=97
14Ø IF B=2 THEN LET B=115
15Ø IF B=3 THEN LET B=12Ø
16Ø IF B=4 THEN LET B=122
17Ø IF C=1 THEN LET C=97
18Ø IF C=2 THEN LET C=115
19Ø IF C=3 THEN LET C=12Ø
2ØØ IF C=4 THEN LET C=122
21Ø PRINT "♥■■■■■
   ■■■■■"TAB(15)CHR$(A)
   SPC(3)CHR$(B)SPC(3)CHR$(C)
22Ø IF A=B AND B=C THEN PRINT
   TAB(5)"■■■■YOU'VE HIT THE
   JACKPOT....$5Ø":LET M=M+5Ø
23Ø IF (A=B OR B=C) AND A<>C THEN
```

Microtip

Using REPEAT ... UNTIL

Acorn computers have two extra statements called REPEAT ... UNTIL which you can often use in place of IF ... THEN ... GOTO. This is useful when you want to repeat a section of program over and over again, only stopping when a certain condition is true. In a games program this might be when you have run out of bombs, for instance.

Using IF ... THEN, the program can be written like this:

```
5Ø (main program starts here)
2ØØ IF bombs=Ø THEN PRINT "You've
   lost" :END
21Ø GOTO 5Ø
```

And with REPEAT ... UNTIL, it looks like this:

```
45 REPEAT
5Ø (main program starts here)
2ØØ UNTIL bombs=Ø
21Ø PRINT "You've lost" :END
```

The two versions are equivalent, but the second is easier to use. In general, it's faster and the whole program is better structured.

```
    PRINT TAB(13) "■■■■YOU'VE WON
    $10":LET M=M+10
240 FOR D=1TO1500:NEXT
250 PRINT "■■■■ANOTHER GO?
    (Y/N)...YOU HAVE $";M;"LEFT"
260 GET K$:IF K$<>"Y" AND K$<>"N"
    THEN GOTO 260
270 IF K$="Y" THEN GOTO 30
280 END
```

This program uses several IF . . . THEN lines. The first one in Line 50 simply checks to see if you have enough money to play. If you do it ignores the line, but if you don't it PRINTs out a message and ends the game.

Lines 60 to 80 choose three random numbers and Line 210 converts these numbers into characters and PRINTs them out at the centre of the screen.

The Sinclair, Dragon, Tandy and Acorn machines convert these numbers straight into characters. The Commodore needs twelve extra lines to convert each random number from 1 to 4 into the code for one of the four suits—hearts, clubs, diamonds and spades—conveniently available as part of this machine's ROM graphics. Line 210 then PRINTs these out on the screen.

At Line 220, if all three characters are the same you win the jackpot and your money is increased by $50. At Line 230 you win $10 if two adjacent characters are the same (either A=B or B=C will do), but you don't win if only the outer characters are the same.

If you don't have a winning line then the computer ignores Lines 220 and 230 and goes on to Line 240. This line causes a slight delay, then the next few lines are another version of the Yes/No routine which asks you if you want another go.

IF . . . THEN . . . ELSE

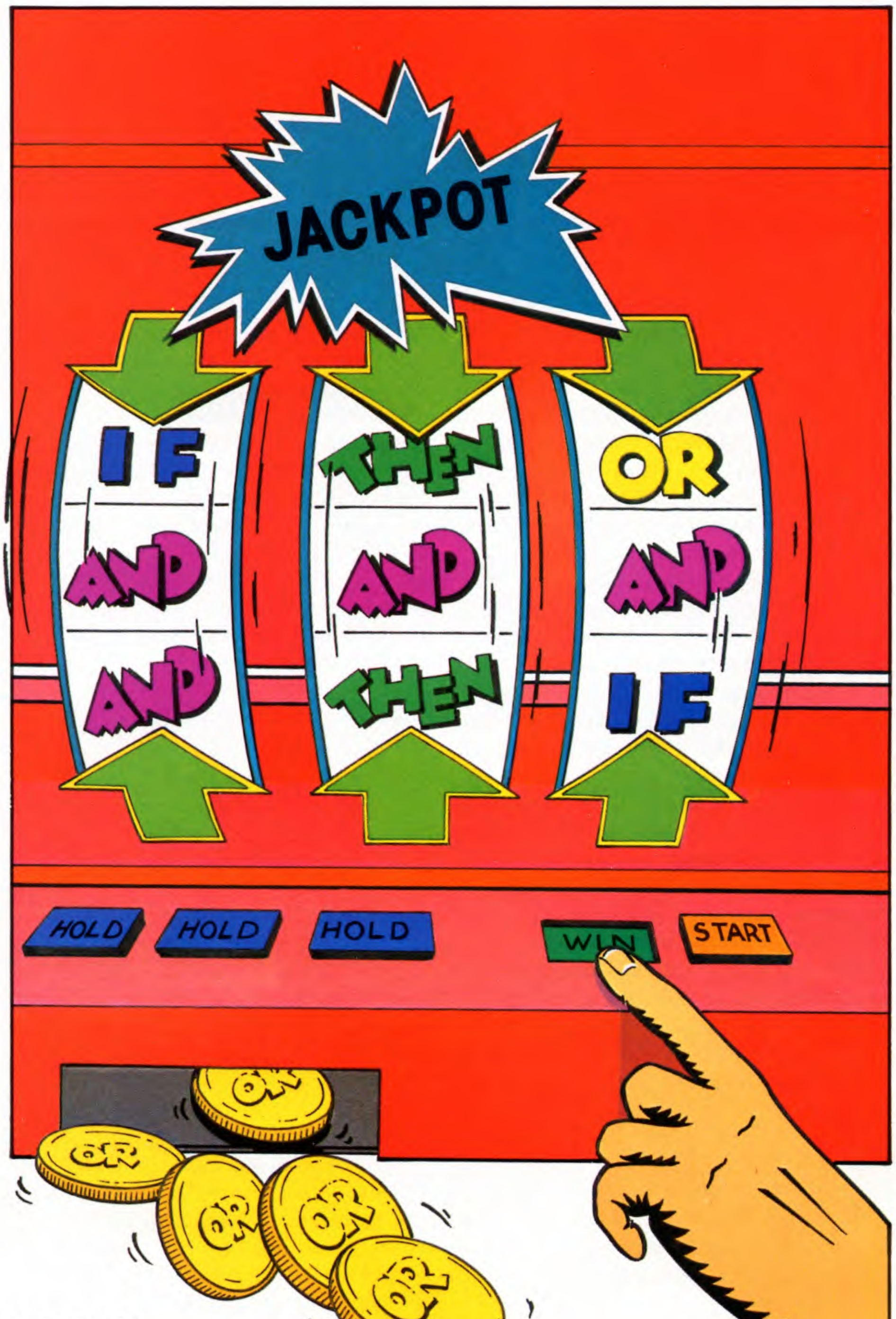
On some computers (but not the Spectrum, ZX81, Commodore 64 or Vic 20) you can write IF . . . THEN . . . ELSE. Here's an example:

```
10 INPUT AGE
20 IF AGE <18 THEN PRINT "UNDERAGE"
   ELSE PRINT "ELIGIBLE"
```

This does exactly what it says—if you are under 18 years old the computer will print "UNDERAGE", but if you are 18 or over it will print "ELIGIBLE".

IF . . . THEN . . . ELSE is useful as it makes the program easier to write and understand. It is more like an ordinary sentence.

But it is not essential, and you can write programs without it if your computer just has IF . . . THEN. In fact, there are two ways round the problem. The first uses IF . . . THEN followed by GOTO to jump to the correct part of the program—making the last program:




```
10 INPUT AGE
20 IF AGE<18 THEN PRINT
   "UNDERAGE": GOTO 30
25 PRINT "ELIGIBLE"
30 ... rest of program
```

The second method uses an extra IF . . . THEN statement to make sure every possible condition is covered:

```
10 INPUT AGE
20 IF AGE<18 THEN PRINT "UNDERAGE"
25 IF AGE>=18 THEN PRINT "ELIGIBLE"
30 ... rest of program
```

10-MINUTE GAMES GRAPHICS

You don't need to know machine code—or even understand binary—to produce simple games graphics. Here are a couple of dozen that will RUN on your computer

Anyone with a dozen hours' computer experience can create original graphics characters for use in games. All you need is a pencil and paper to sketch out your ideas, plus two—or at most three—simple routines to turn these ideas into computer pictures.

Each home computer has its own method of creating new graphics from its user defined graphics characters, or UDGs. The size of the characters you can create easily also varies. The Commodore, for example, has a standard 'sprite' which is 24 pixels (dots) by 21, whereas the best that the Spectrum can offer is a UDG only eight pixels by eight. The ZX81 and Vic 2Ø are not covered here, but a later article will give some routines.

Whatever your computer offers, however, the best way to start creating your own graphics is in the 8 × 8 size—about the size of an 'enemy' in a space game. Once you have the knack of doing this, it is easy to create a bigger graphic if your computer allows it. Or, if it doesn't, to string two or three small graphics together to make a larger one.

FROM DRAWING TO BINARY

Eventually, your computer will store the information you give it in *binary* (base 2) arithmetic. But you do not have to understand binary—or even know what it is—to turn your graph-paper character into rows of binary numbers. All you need to know is:

1 Every time you want a *dot*, you use the number 1.

2 Every time you want a *space*, you use Ø.

Take the cross of Lorraine below, for example. Its top line consists of three spaces, one dot, and four more spaces. In binary, that's ØØØ1ØØØØ.

The second line is two spaces, three dots and three more spaces—ØØ111ØØØ. And the whole pattern can be represented like this:

Ø	Ø	Ø	1	Ø	Ø	Ø	Ø
Ø	Ø	1	1	1	Ø	Ø	Ø
Ø	Ø	Ø	1	Ø	Ø	Ø	Ø
Ø	1	1	1	1	1	Ø	Ø
Ø	Ø	Ø	1	Ø	Ø	Ø	Ø
Ø	Ø	Ø	1	Ø	Ø	Ø	Ø
Ø	Ø	Ø	1	Ø	Ø	Ø	Ø
Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø

On some computers the DATA you need to set up a user defined graphic can be entered directly in this binary form. On others, you first have to convert it into *decimal* (base 1Ø) or *hexadecimal* (base 16) arithmetic. So read the section for your own machine (pages 4Ø to 44) before you do any conversions.

BINARY TO DECIMAL

The quickest way of converting binary into decimal—that is, the ordinary units, tens, hundreds, and thousands we use every day—is to use a little chart eight rows wide by nine rows deep. In the top row you write these numbers: 128, 64, 32, 16, 8, 4, 2, 1. In the other eight rows you write the binary for the graphic that you want to reproduce. Here, for example, is the chart for the cross of Lorraine:

128	64	32	16	8	4	2	1
Ø	Ø	Ø	1	Ø	Ø	Ø	Ø
Ø	Ø	1	1	1	Ø	Ø	Ø
Ø	Ø	Ø	1	Ø	Ø	Ø	Ø
Ø	1	1	1	1	1	Ø	Ø
Ø	Ø	Ø	1	Ø	Ø	Ø	Ø
Ø	Ø	Ø	1	Ø	Ø	Ø	Ø
Ø	Ø	Ø	1	Ø	Ø	Ø	Ø
Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø

To do the conversion, you ignore the Øs altogether. First you compare each of the 1s in the binary with the number at the top of its column. Then you take all the numbers for the first *horizontal* row and add them up, repeating the process for each row.

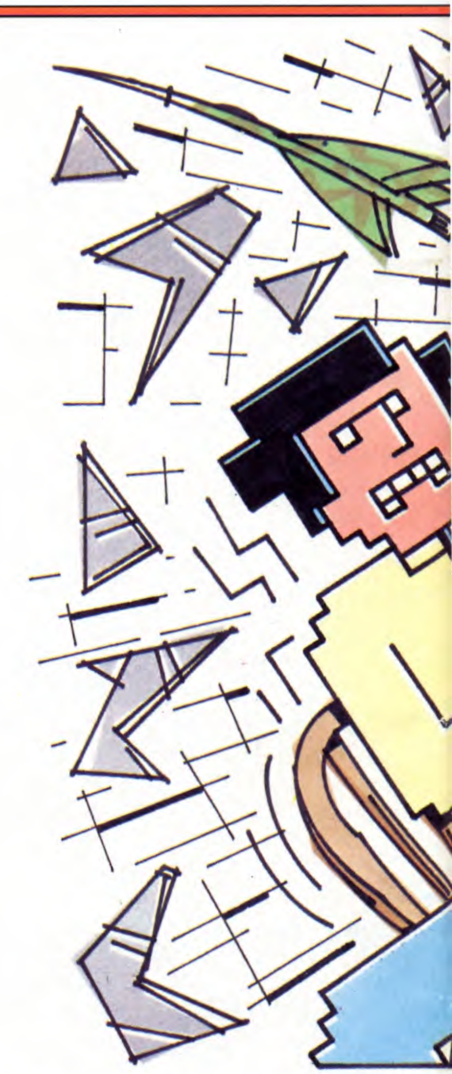
In the example above, the top row consists of nothing, nothing, nothing, 16, nothing, nothing, nothing, nothing. Total: 16.

The second row consists of nothing, nothing, 32, 16, 8, nothing, nothing, nothing. Total for this row (32 + 16 + 8): 56.

By the time you have repeated this process to the bottom of the chart you will have eight decimal numbers. So your DATA statement, to enter into your computer, will look like this:

```
DATA 16, 56, 16, 124, 16, 16, 16, Ø
```

When you are new to it, this seems a long-winded way of doing the job. But after half a dozen attempts you will find yourself doing it very rapidly. And some common combinations—like decimal 255 for binary 11111111—you'll find yourself remembering without bothering to work them out.



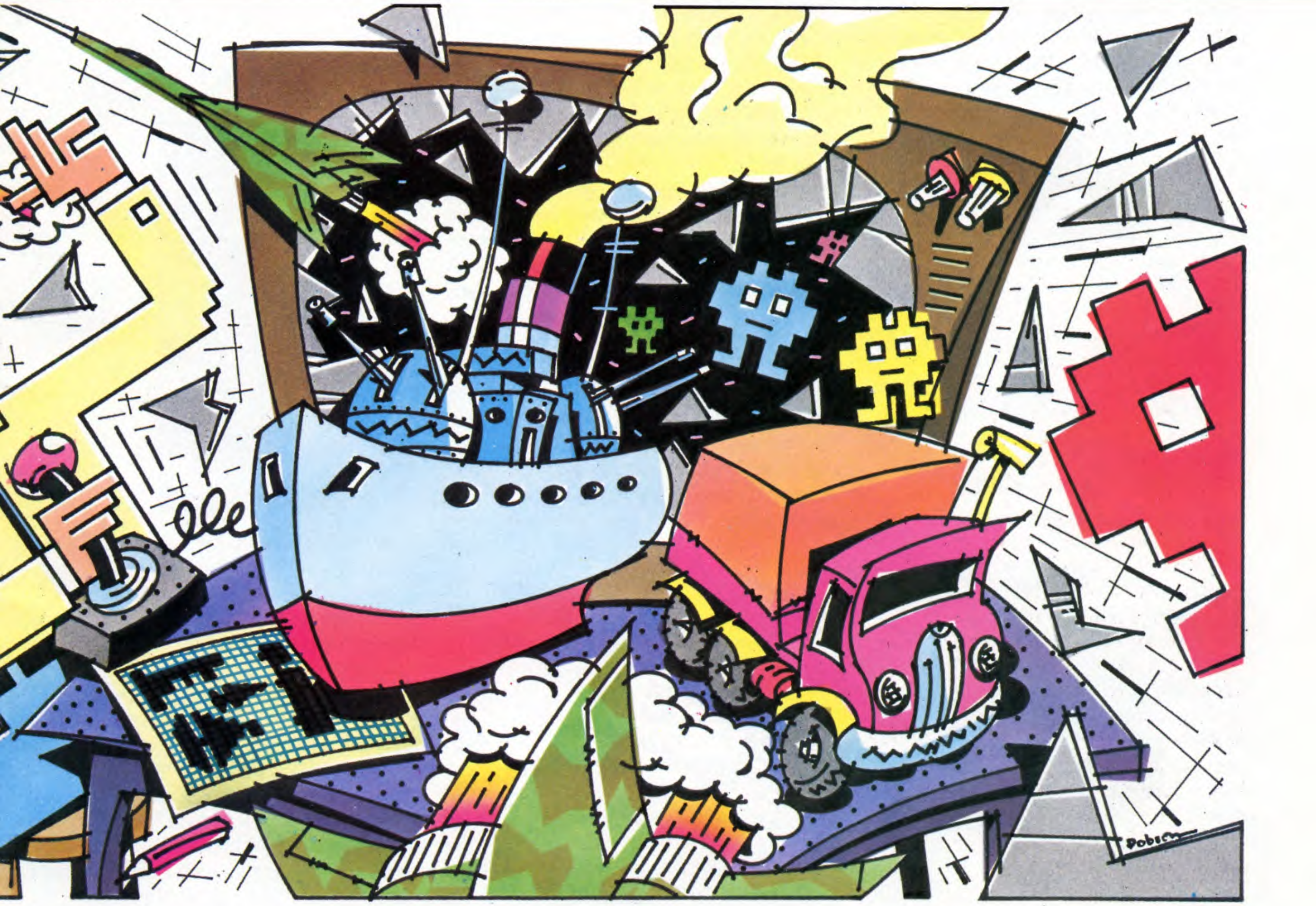
BINARY TO HEX

Converting binary to hexadecimal, if that's what your computer wants, is even easier.

You will need this chart, although there is no need to write out the binary numbers alongside or below it.

Binary	Hexadecimal
ØØØØ	Ø
ØØØ1	1
ØØ1Ø	2
ØØ11	3
Ø1ØØ	4
Ø1Ø1	5
Ø11Ø	6

- FROM DRAWING TO BINARY IN ONE EASY STAGE
- BINARY/DECIMAL AND BINARY/HEX CONVERSIONS
- USING GRAPHICS IN PROGRAMS



Ø111	7
1ØØØ	8
1ØØ1	9
1Ø1Ø	A
1Ø11	B
11ØØ	C
11Ø1	D
111Ø	E
1111	F

This time you do *not* ignore the Øs. The first thing you do is to split each line of binary numbers in half. Then you take the hex equivalent of the first half-row (that is, group of four digits) and write it down. Alongside it you write the hex equivalent of the second half-row. And the two together are the hex number you need.

To return again to that cross of Lorraine. The first half-row of the binary is by now terribly familiar: ØØØ1. In hex (look at the chart) that's 1. The second half-row is ØØØØ. In hex, that's Ø. Write the two numbers together and you have 1Ø—the hex number you want for your DATA statement.

Similarly, if you split the second row you will see that the first half is ØØ11—in hex, 3—and the second half is 1ØØØ—in hex, 8. So the hex number for the whole line is 38.

Repeat the process to the bottom of the binary listing and you'll have eight hex numbers. And your DATA statement for the computer will look something like this:

```
DATA 1Ø, 38, 1Ø, 7C, 1Ø, 1Ø, 1Ø, ØØ
```

The only other thing you must do is to tell your computer, which can't guess, whether the numbers you are entering are in hex or in decimal. How to do this is in the section for your own machine.

Of course it is possible to write a short program to convert binary into decimal, or binary into hex. But such a program is not much help if you want to amend something when you are on the computer keyboard.

The Dragon and Tandy will accept DATA in hex, decimal or binary.

If the DATA is in hex, you have to add an extra line to the program, as shown below, to turn it into decimal. But this is still normally the neatest way of doing the job.

The first program draws a tiny aircraft (below) in the top left-hand corner of the screen—not the best place to view it, but the easiest place to start if you want to write a program to move it around the screen.

```
2Ø PMODE 4,1
3Ø PCLS
4Ø SCREEN 1,1
6Ø FOR L=Ø TO 7
7Ø READ N$
8Ø POKE L*32+1536,VAL("&H"+N$)
9Ø NEXT L
11Ø GOTO 11Ø
5ØØ DATA ØØ,1Ø,18,9C,FF,9C,18,1Ø
```

Type in the program and RUN it.

PMODE 4,1 has been selected in Line 2Ø because only this highest resolution mode allows you to produce UDGs.

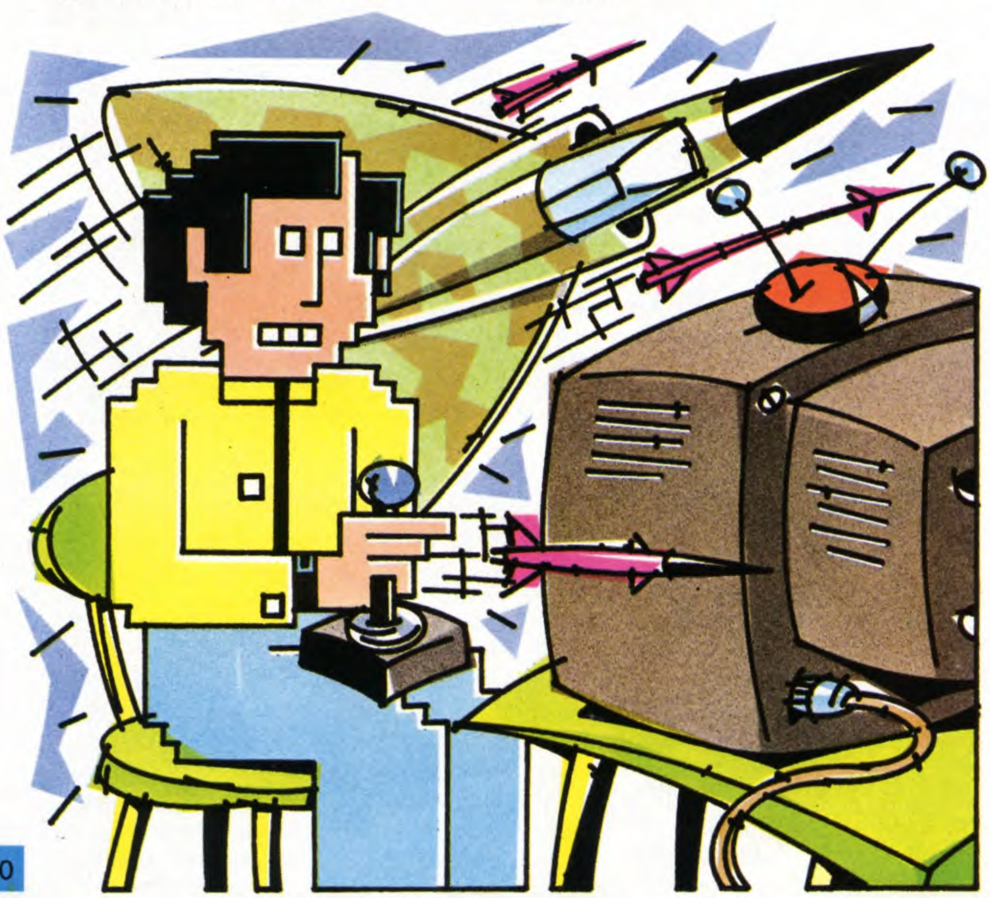
To clear the screen ready for your character, you must use the PCLS command as in Line 3Ø. This applies not just to PMODE 4,1 but to all the high resolution modes.

To turn on the high resolution screen so that the UDG can be displayed you use SCREEN 1,1 as in Line 4Ø. SCREEN 1,1 also chooses the black and white colour set.

The FOR . . . NEXT loop in Lines 6Ø and 9Ø causes Line 7Ø to be executed eight times. Every time the computer reaches READ N$, it reads the next piece of DATA in the DATA line, Line 5ØØ.

Line 8Ø is important for two reasons. First, it makes the pattern of pixels which correspond to the hex numbers in Line 5ØØ appear on the screen. Second, it converts the information from the DATA line into decimal, then puts it directly into the part of the computer's memory which governs the display on your TV screen.

Finally, Line 11Ø is a loop which keeps the high resolution screen switched on. Without this line the program would end, making the computer switch back automatically to the text screen. So you wouldn't see your design on the screen at all!



TALLER UDGs

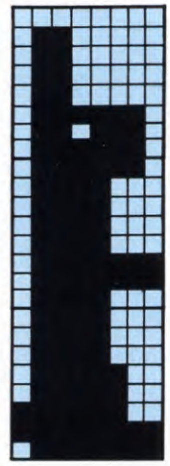
Simply by changing Line 6Ø and altering the DATA in Line 5ØØ, you can create a tall, thin UDG instead of an 8 × 8.

First change Line 6Ø so that it reads:

```
6Ø FOR L=Ø TO 23
```

Next alter the DATA in Line 5ØØ so that it is now:

```
5ØØ DATA ØØ,6Ø,6Ø,6Ø,6Ø,7E,6E,7E,7E,
    78,78,78,78,7F,7F,78,78,78,78,7C,7C,
    FC,FF,7F
```



When you RUN the program you should find a picture of a rabbit on the screen.

Altering Line 6Ø has allowed more DATA to be READ from Line 5ØØ, and the DATA in Line 5ØØ defines the shape of the rabbit using hex numbers.

Using this system, you can create a UDG of any height you wish. Once you have designed the UDG on graph paper, count how many lines of pixels you have used. Next, alter Line 6Ø so that the correct number of lines are read. Then make sure that the number of pieces of information in the DATA statement corresponds to the number of times the FOR . . . NEXT loop is executed.

WIDER UDGs

To accommodate a graphic which is long and low, instead of being tall and thin, is a bit more complicated.

Start by changing Lines 6Ø and 8Ø so that they read:

```
6Ø FOR L=Ø TO 7
8Ø POKE L*32+1536+F,VAL("&H"+N$)
```

Then alter the DATA in Line 5ØØ so that it reads:

```
5ØØ DATA ØF,ØF,EF,EF,EF,EF,FE,44,FF,
    FF,FF,FF,FF,FF,4Ø,ØØ,FF,FF,FF,FF,
    FF,FF,66,66
```

And finally add these lines:

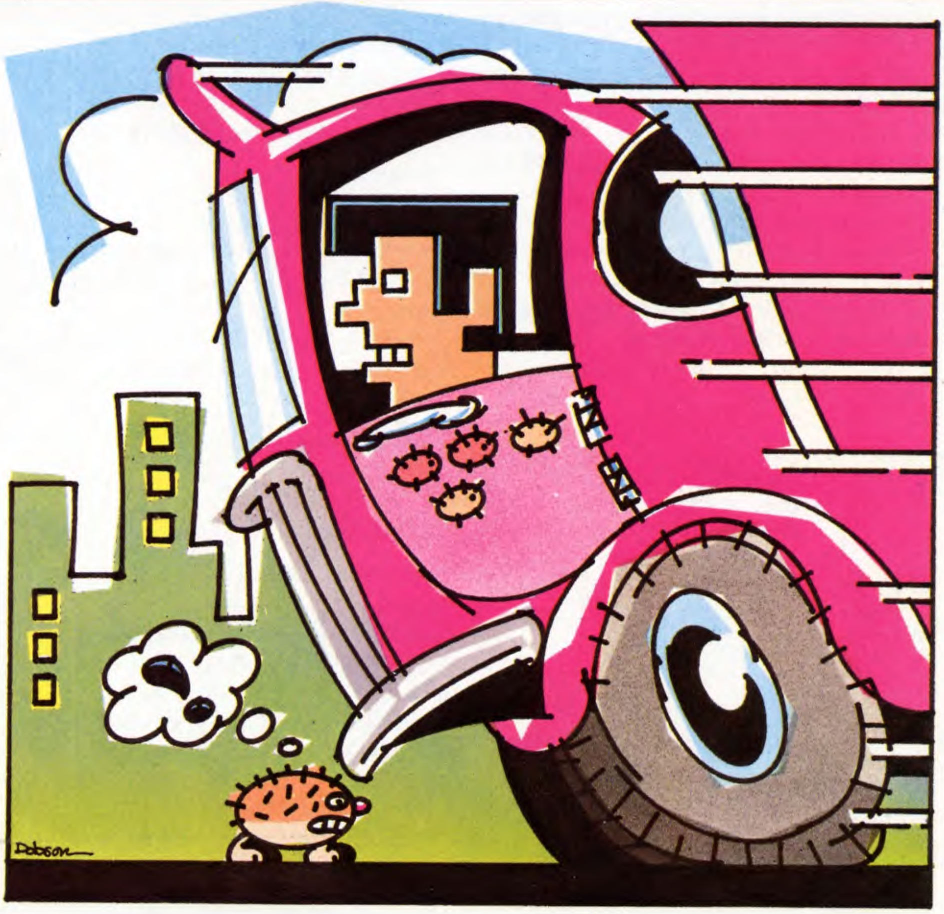
```
5Ø FOR F=Ø TO 2
1ØØ NEXT F
```

When you RUN the altered program you should find a picture of an articulated lorry on the screen.

How does it work? The 24 pieces of DATA in Line 5ØØ make three squares each eight pixels deep. If the DATA were read by a single FOR . . . NEXT loop as before, the lorry would look peculiar, to say the least—it would be cut up into three slices and stacked vertically down the screen.

So the computer has to be told to arrange the blocks side by side. To do this, an extra FOR . . . NEXT loop (Lines 5Ø and 1ØØ) is used, and Line 8Ø is amended so that it now has + F in the POKE statement.

What the program does is to READ the first eight pieces of DATA and to POKE them on to the screen. Then the extra FOR . . . NEXT loop alters the POKE value in Line 8Ø so that the second block of DATA appears on the top line of the screen alongside the first block. After another eight pieces of DATA have been READ, the final block is POKED on to the screen on the top line.



MOVING UDGs

You can move the UDG of the aircraft around the screen by adding these lines:

```
11Ø DIM A(3),B(3)
12Ø GET (Ø,Ø)-(7,7),A,G
13Ø PCLS
14Ø LET X=127
15Ø LET Y=95
16Ø PUT (X,Y)-(X+7,Y+7),A,PSET
17Ø LET LX=X
18Ø LET LY=Y
19Ø IF PEEK(338)=239 AND Y>2 THEN
   Y=Y-2:GOTO 24Ø
2ØØ IF PEEK(342)=247 AND Y<182 THEN
   Y=Y+2:GOTO 24Ø
21Ø IF PEEK(34Ø)=223 AND X>3 THEN
   X=X-3:GOTO 24Ø
22Ø IF PEEK(338)=223 AND X<245 THEN
   X=X+3:GOTO 24Ø
23Ø GOTO 19Ø
24Ø PUT (LX,LY)-(LX+7,LY+7),B,PSET
25Ø GOTO 16Ø
```

On the Tandy, change the 239 in Line 19Ø to 251; change 247 in Line 2ØØ to 253; 223 in Line 21Ø and 22Ø to 247.

When you RUN this, the Z key causes left movement, X right movement, P upwards movement and L downwards movement.

Three new BASIC keywords have been introduced in this program—GET, PUT and DIM. A full explanation is in a later article, but GET allows the computer to remember what is on a particular part of the screen, and PUT allows the computer to place it anywhere on the screen. DIM reserves memory space for GET.

Lines 19Ø to 22Ø are the lines which detect keypresses and move the UDG. Keyboard control is covered on page 59.

If you want to move the rabbit you'll have to make these alterations. On the Tandy, use 253, not 247, in Line 2ØØ.

```
11Ø DIM A(6),B(6)
12Ø GET (Ø,Ø)-(7,23),A,G
16Ø PUT (X,Y)-(X+7,Y+23),A,PSET
2ØØ IF PEEK(342)=247 AND Y<166 THEN
   Y=Y+2:GOTO 24Ø
24Ø PUT (LX,LY)-(LX+7,LY+23),B,PSET
```

And to move the lorry, make these changes. On the Tandy, use 253, not 247 in Line 2ØØ, and 247 in place of 223 in Line 22Ø.

```
11Ø DIM A(6),B(6)
12Ø GET (Ø,Ø)-(23,7),A,G
16Ø PUT (X,Y)-(X+23,Y+7),A,PSET
2ØØ IF PEEK(342)=247 AND Y<182 THEN
   Y=Y+2:GOTO 24Ø
22Ø IF PEEK(338)=223 AND X<229 THEN
   X=X+3:GOTO 24Ø
24Ø PUT (LX,LY)-(LX+23,LY+7),
   B,PSET
```

USING BINARY DATA

You can put binary numbers in the DATA line if you like, but make sure each number consists of an 8-bit byte. You'll also have to add these lines to the program:

```
71 LET N=Ø
72 FOR J=1 TO 8
74 IF MID$(N$,J,1)="1" THEN
   N=N+2↑(8-J)
76 NEXT J
```

And make sure that Line 8Ø reads as follows:

```
8Ø POKE L*32+1536+F,N
```

The new lines examine each bit in turn of the 8-bit number and convert it to a decimal number. It is, in short, the computer equivalent of the conversion table in the introduction to this article.

● The movement of high resolution graphics is covered in detail in a later article.

On the Acorn machines the list of numbers to define your character can be in decimal or hexadecimal, whichever you like. It is usually easier to look up the table on page 38 and convert the binary pattern directly into hex rather than add up the decimal equivalent of each line of dots. Remember, though, that each hex number has to start with an ampersand sign—&—to let the computer know what sort of number to expect.

The numbers for the little ghost shown below are:

Binary	Hex	Decimal
ØØ1111ØØ	3C	6Ø
Ø111111Ø	7E	126
11Ø11Ø11	DB	219
11111111	FF	255
11ØØØØ11	C3	195
Ø111111Ø	7E	126
Ø1Ø11Ø1Ø	5A	9Ø
11ØØØØ11	C3	195

So to define the character, you use either:

```
1Ø VDU 23,224,&3C,&7E,&DB,&FF,
   &C3,&7E,&5A,&C3
```

or:

```
1Ø VDU 23,224,6Ø,126,219,255,195,
   126,9Ø,195
```

The VDU 23 part means 'define a character'. The next number is the code number for that character. There are 32 code numbers available, numbered from 224 to 255. It doesn't matter what order you use them in. Then, after the code number, come the eight numbers you worked out for the ghost.

The character has now been defined and the computer knows what it is, but how do you display it on the screen? That part is easy. First put the computer into any of the modes except mode 7. For example:

```
5 MODE 1
```

Then all you do is PRINT the character:

```
2Ø PRINT TAB (5,1Ø) CHR$ 224
```

RUN the program and see.

CHR$ is pronounced 'character string' and the number is just the code number described above.

Actually, CHR$ 224 doesn't mean much in itself. So to remind you what it is, you can add an extra line:

```
15 ghost $ = CHR$ 224
```

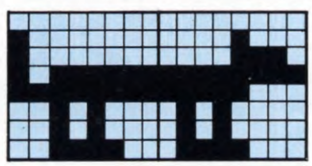
Then Line 2Ø becomes:

```
2Ø PRINT TAB (5,1Ø) ghost $
```

Now it is obvious what is going on.

DEFINING LARGER CHARACTERS

The next character (below) is a sausage dog and, being long and thin, he takes up two character grids and needs two VDU statements to define him—one for each grid:



```
1ØØ VDU 23,225,&ØØ,&8Ø,&8Ø,&BF,&FF,
    &28,&28,&3C
11Ø VDU 23,226,&ØØ,&Ø8,&ØE,&FF,&F8,
    &5Ø,&5Ø,&7C
```

Then both parts have to be PRINTed next to each other in the right order:

```
12Ø dog$ = CHR$ 225 + CHR$ 226
13Ø PRINT TAB(1Ø,1Ø) dog$
```

Note you can use PRINT TAB to PRINT the character exactly where you want.

For a longer sausage dog you will need to define an extra middle section:

```
115 VDU 23,227,&ØØ,&ØØ,&ØØ,&ØØ,&FF,
    &FF,&ØØ,&ØØ,&ØØ
```

Then add this character between his front and back end:

```
12Ø dog$ = CHR$225 + CHR$227 + CHR$226
```

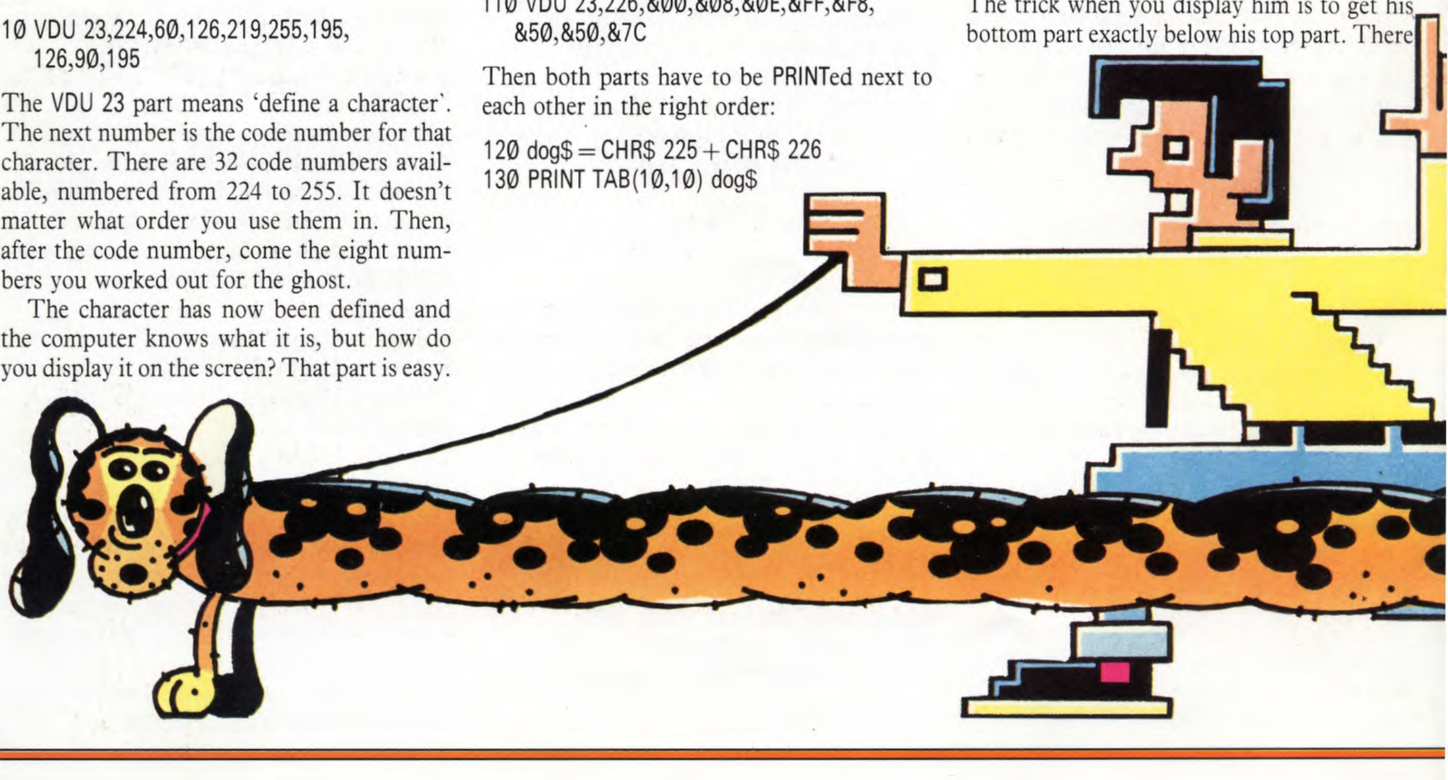
Now RUN the program to see the elongated version. In fact, you can add as many middle sections as you like—up to the whole width of the screen!

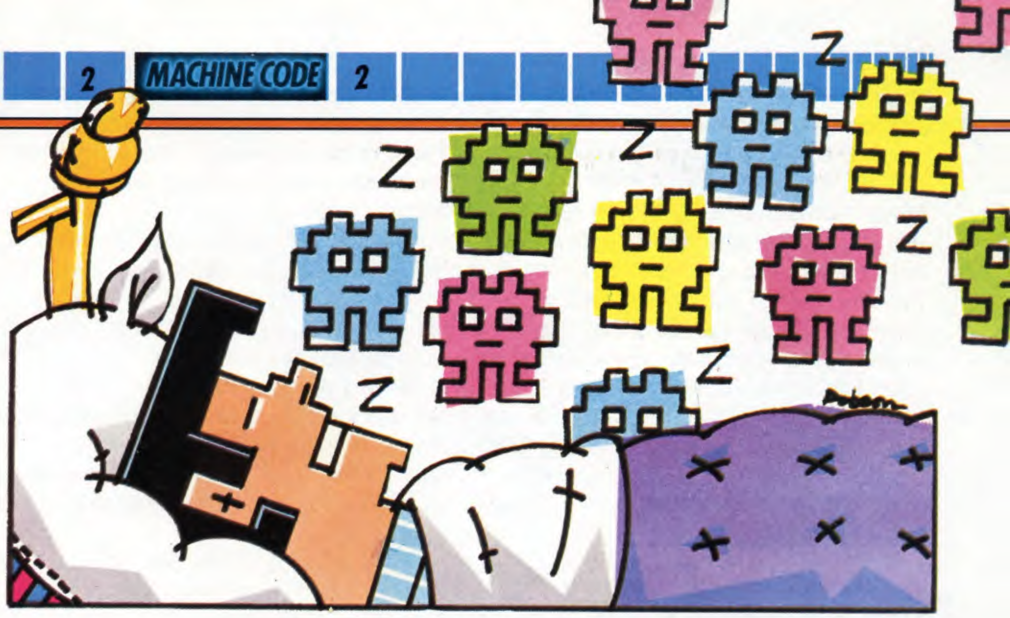
MAKING A TALL, THIN CHARACTER

To display a tall, thin character, first define him as usual:

```
2ØØ VDU 23,228,&3C,&3C,&3C,&18,
    &FF,&BD,&BD,&BD
21Ø VDU 23,229,&BD,&3C,&3C,&24,
    &24,&24,&24,&E7
```

The trick when you display him is to get his bottom part exactly below his top part. There





are two ways of doing this. The first is to PRINT each part separately at the correct location, for example:

```
220 PRINT TAB(15,9) CHR$228 : PRINT
    TAB(15,10) CHR$229
```

RUN the program to see that it does work. The man should be standing next to the dog.

But a better way is to make up a complete character as you did for the dog. Here's how:

```
220 man$ = CHR$228 + CHR$10 + CHR$8 +
    CHR$229
230 PRINT TAB(15,9) man$
```

CHR$10 and 8 are there to control the cursor. CHR$10 moves it down one space and CHR$8 moves it back a space so it's in just the right place to PRINT the man's legs.

The three characters we've looked at were defined and PRINTed in turn. But normally in a program, all the VDU 23 character definitions would be grouped together near the start of the program, just to make your program neater.

So here are all the lines you have entered so far, but renumbered so you can see more clearly what is going on.

```
5 MODE 1
10 VDU 23,224,&3C,&7E,&DB,&FF,&C3,
   &7E,&5A,&C3
20 VDU 23,225,&00,&80,&80,&BF,&FF,
   &28,&28,&3C
30 VDU 23,226,&00,&08,&0E,&FF,&F8,
   &50,&50,&7C
40 VDU 23,227,&00,&00,&00,&FF,&FF,
   &00,&00,&00
50 VDU 23,228,&3C,&3C,&3C,&18,&FF,
   &BD,&BD,&BD
60 VDU 23,229,&BD,&3C,&3C,&24,&24,
   &24,&24,&E7
70 ghost $ = CHR$224
80 dog$ = CHR$225 + CHR$227 + CHR$226
90 man$ = CHR$228 + CHR$10 + CHR$8 +
   CHR$229
100 PRINT TAB(5,10) ghost$
110 PRINT TAB(10,10) dog$
120 PRINT TAB(15,9) man$
```

Standard UDGs based on an 8 × 8 pixel matrix are used on the Commodore 64 less frequently than the somewhat more versatile sprite (see page 15) which is much better for games programming.

However, 8 × 8 UDGs do prove useful at times, and a fairly common application is for redefining part or all of the normal character set. You may wish to introduce changes of this sort in programs which have to incorporate foreign punctuation, symbols or letters, for instance.

Take, for example, the French character è which doesn't form part of the normal character set. This and other characters could be incorporated within a new character set for use whenever French lettering was required.

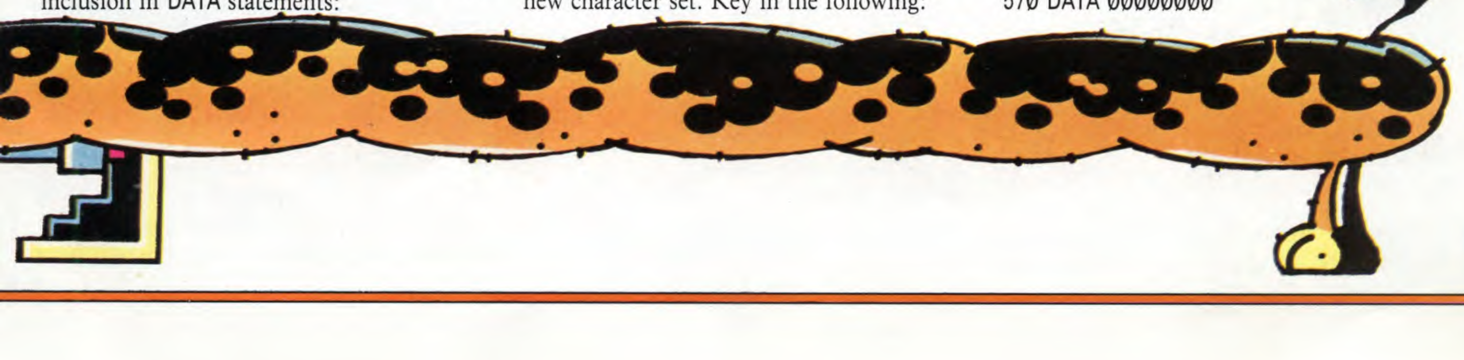
Creating the actual UDGs follows exactly the same methods outlined elsewhere in this article. For example, the character è can be represented in the following forms, ready for inclusion in DATA statements:

Binary	Hex	Decimal
00010000	10	16
00001000	08	8
00111100	3C	60
01100110	66	102
01111110	7E	126
01100000	60	96
00111100	3C	60
00000000	00	0

The decimal form is most readily usable but inconvenient to calculate. With the appropriate programming either the binary or the hex systems can be used.

Let's incorporate this character within a new character set. Key in the following:

```
10 A=12 : Z=A*1024/256
20 POKE 53272, (PEEK(53272)AND240) OR A
30 POKE 52,Z: POKE 56,Z: CLR: A=12
40 POKE 56334, PEEK (56334) AND 254
50 POKE 1, PEEK (1) AND 251
60 FOR J=0 TO 56832-53248
70 POKE A*1024+J, PEEK (53248+J)
80 NEXT J
90 POKE 1, PEEK (1) OR 4
100 POKE 56334, PEEK (56334) OR 1
110 SC=5: Z=1024*12: FOR J=Z+(SC*8)
    TO Z+(SC*8)+7: READ A$
120 N=0: FOR T=1 TO LEN(A$)
130 IF MID$(A$,T,1)="1" THEN N=N+2↑
    (LEN(A$)-T)
140 NEXT T:POKE J,N: NEXT J
500 DATA 00010000
510 DATA 00001000
520 DATA 00111100
530 DATA 01100110
540 DATA 01111110
550 DATA 01100000
560 DATA 00111100
570 DATA 00000000
```



If you now RUN this program, nothing appears to happen for about a minute. When the 'ready' prompt appears, press the E key and you should see è displayed. If you see just graphics symbols, simultaneously press the C= and SHIFT keys.

Now try changing the pattern of Øs and 1s in the DATA statements to create another letter, or a simple graphic. (Remember to hit RETURN to enter each line.) Then reRUN the program, and press E again to display your new UDG.

PROGRAM CLOSE-UP

For those with more experience, here's an explanation of how the program works:

The normal characters, many of which can be used in a newly designed character set, are in ROM and cannot themselves be changed. Nor can they be used in addition to UDG characters, which poses problems. But the technique is to copy what you want of the ROM character set into RAM, where you can make the necessary changes, replacing unwanted characters with new ones. These characters can be letters or graphics or a mixture of the two. The ROM character set is then switched out and the RAM character set takes over.

The program undertakes several very distinct operations, the first of which is to allocate RAM memory for storage of the new character set. The value of A in Lines 1Ø and 3Ø enables you to choose which area of memory is to be used. Any integer value in the 4 to 16 range can be used, and in the program the value 12 changes the character memory pointer to 12288 (which is 12*1Ø24). A value 4 would effectively place character storage at location 4Ø96 (which is 4*1Ø24) . . . and so on.

Line 3Ø changes two pointers (for end-of-BASIC, and start of string storage) so that a BASIC program does not overwrite—and so ruin—the character set. This is a much used technique for protecting programs from BASIC.

Next, in Line 4Ø, the program turns off what is called the interrupt keyboard scan. Line 5Ø switches in the character ROM.

The copying routine occurs in Lines 6Ø, 7Ø and 8Ø. The figures 53248 and 56832 in Line 6Ø refer to the start and finish addresses of the eight-part character sets which are copied into RAM. Although the program copies the lot, you can restrict the amount that is copied by changing the range of values taken by J. You can even pick and choose which characters you want to copy, as will be explained in a later article.

When copying is complete—this explains the minute-long delay when the program is RUN—the character ROM is switched out (Line 9Ø) and the interrupt is then restored (Line 1ØØ).

Lines 11Ø and 14Ø change the definition of a selected character (SC) in character memory by READing the relevant DATA statements (Lines 5ØØ–58Ø) via the binary-decimal conversion routine in Lines 12Ø and 13Ø. The value of SC is the screen code poke value which you can find listed in the appendices of your manuals. An SC of 5 will display your character when E is pressed, as previously mentioned. Try changing this value and reRUNing the program for other screen code values, so assigning another key to the UDG.

To save program space the DATA statement lines may be compressed to a single line: (delete 51Ø–58Ø)

```
500 DATA00010000,00001000,00111100,
    01100110,01111110,01100000,001111
    00,00000000
```

But note that this form doesn't allow you to gauge the appearance of your UDG quite so easily—nor can you edit it so quickly.

If you prefer to work in hex notation substitute the following lines, first deleting Lines 51Ø to 58Ø:

```
130 M=ASC(MID$(A$,T,1))-48:N=
    (M+(M>9)*7)*16↑(LEN(A$)-T)+N
500 DATA 10,08,3C,66,7E,60,3C,00
```

In this form, it is much simpler to key in and edit line 5ØØ.

The Spectrum will accept DATA in either binary numbers or decimal, but not in hex.

To see how it does this, first type in

```
PRINT "(graphics A)"
```

To get the (graphics A) bit, you first hit CAPS SHIFT and the GRAPHICS key together, then type a.

What you see will look like an ordinary capital A. But as described earlier (page 8), it is one you can redefine into any 8 × 8 shape you choose.

And to do *that*, all you need is a five-line program. To 'plant' the fir tree opposite, for instance, you enter:

```
10 FOR n=0 TO 7
20 READ data
30 DATA BIN 00010000, BIN 00011000,
   BIN 00111000, BIN 00111100, BIN
   01111100, BIN 01111110, BIN 11111110,
   BIN 00010000
40 POKE USR "a"+n, data
50 NEXT n
```

This program uses a FOR . . . NEXT loop, Lines 1Ø and 5Ø, to call up in order the eight lines

into which you want to enter DATA. Line 2Ø tells the computer to scan the DATA in Line 3Ø and Line 4Ø POKEs it in.

RUN the program, then type

```
PRINT "(graphics A)"
```

. . . again. You will find that the capital A has vanished, and you have a fir tree in its place.

If you type NEW at this stage, the program itself will of course be erased. But the fir tree graphic will stay in memory until you disconnect the power supply to the computer. So you can move it around the screen, or use it for decorative effects, just as though it were a standard character. Try this, for example:

```
5 CLS
10 FOR y=3 TO 19
20 LET x=INT (RND*20)+5
30 PRINT AT y,x; INK 4;"(graphics A)"
40 LET xx=INT (RND*20)+5
50 PRINT AT y, xx; INK 2; "(graphics A)"
60 NEXT y
```

Could these be the hazards for a skiing game?

When you want to create a graphic larger than the standard 8 × 8 UDG character, all you do is call up and edit two or more UDGs in turn. This program, for example, creates the bow section of the destroyer below (which you can build into a game by the methods given in Games Programming 1):

```
10 FOR n=0 TO 7
20 READ a
30 DATA BIN 0, BIN 0, BIN 0, BIN 00000111,
   BIN 00000011, BIN 11111111
40 POKE USR "a"+n, a
50 NEXT n
```

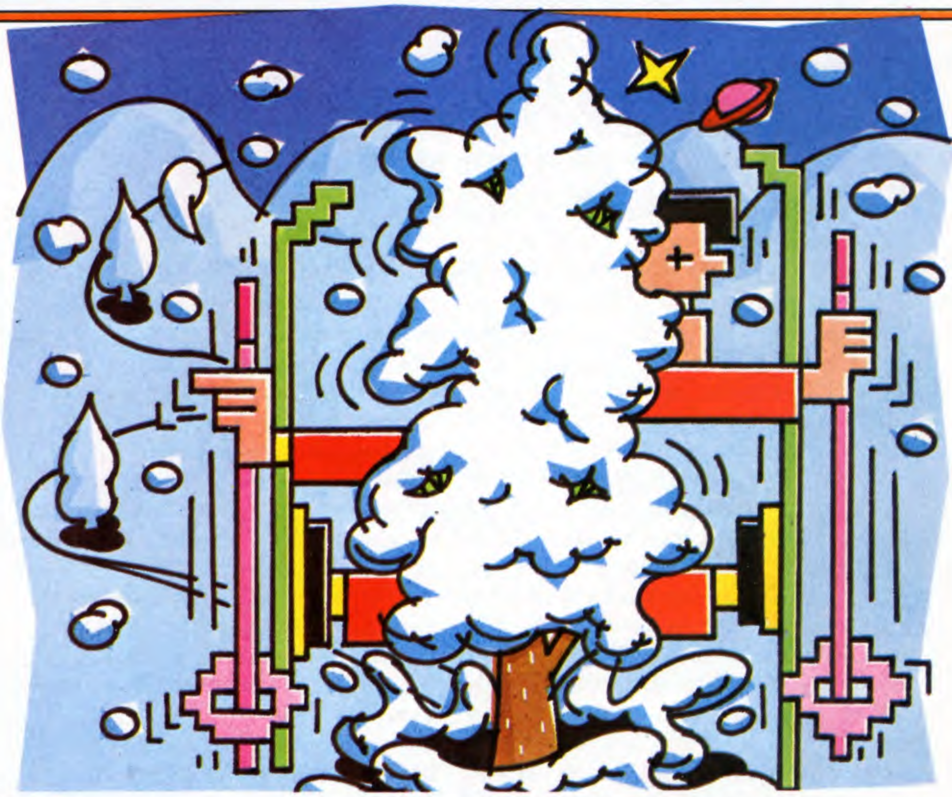
Two points are worth noting here. The first is that you can use just BIN Ø—not eight Øs—if the whole row is to be blank. The second is that in Line 2Ø the simple variable a has been used instead of the word DATA, which we inserted just to make the earlier program easier to understand. It could just as easily be b, or c, or x, or even UNCLE BERT, so long as you use the identical variable in Line 4Ø.

When you come to enter the DATA for the midships and stern sections, there is no need to retype the whole program. With the program RUN once—and the first bit of graphic safely in memory—all you have to do is to edit Line 4Ø, changing USR "a" to USR "b". And, of course, enter a new or edited Line 3Ø to carry the new DATA.

Be careful when entering DATA that you do have eight lines each time, even if some are just Øs. Too few lines and you will get an error report: E Out of DATA, 2Ø: 1. Too many, and you'll find that your ship is sinking!

Finally, to enter DATA in decimal instead of binary, you first convert the binary numbers to decimal, as described in this article. Then you omit the BIN (for binary) from the DATA line of your program. Line 3Ø of the destroyer program, for example, would become:

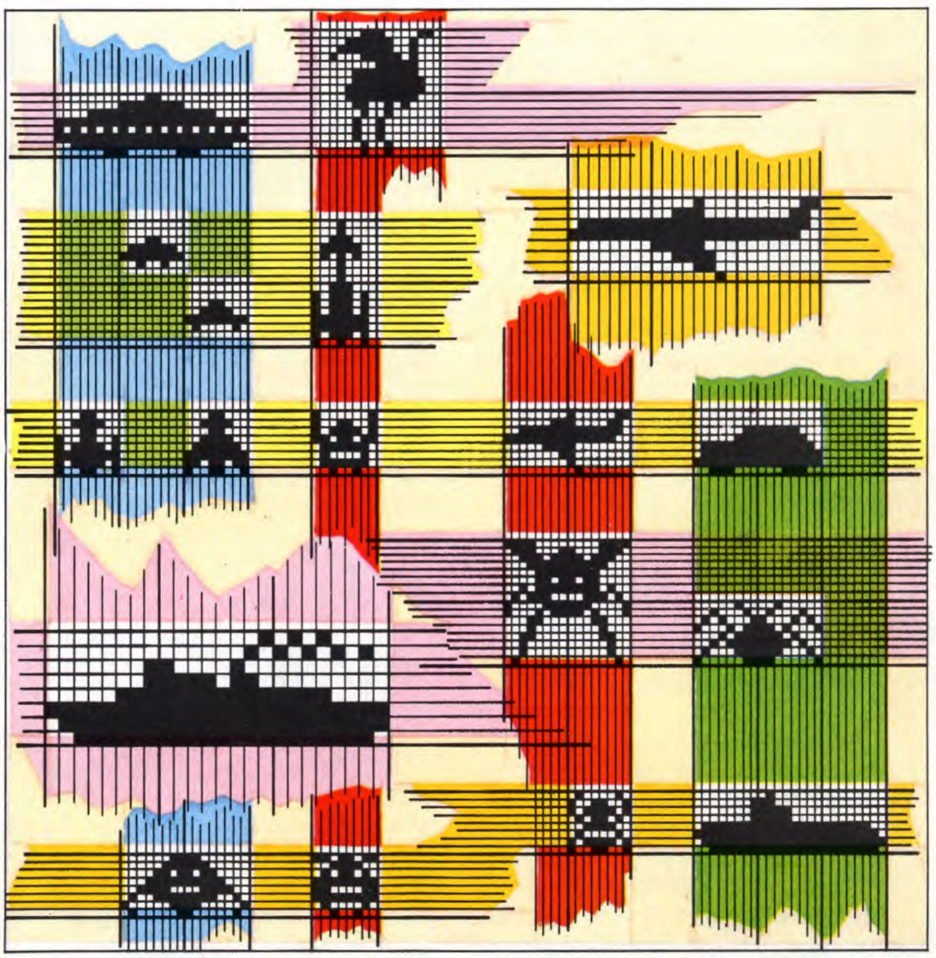
```
3Ø DATA Ø, Ø, Ø, 7, 3, 255, 127, 63
```



AND NOW . . .

SCRAWL...

Once you have grasped the general principles, you can key in any of the graphics on these pages. All will work on any computer covered here. And by the time you've done about three of them, you'll find you are an expert, ready to design new graphics of your own.



Q+A

When entering DATA, which system—binary, decimal or hex—is best to use?

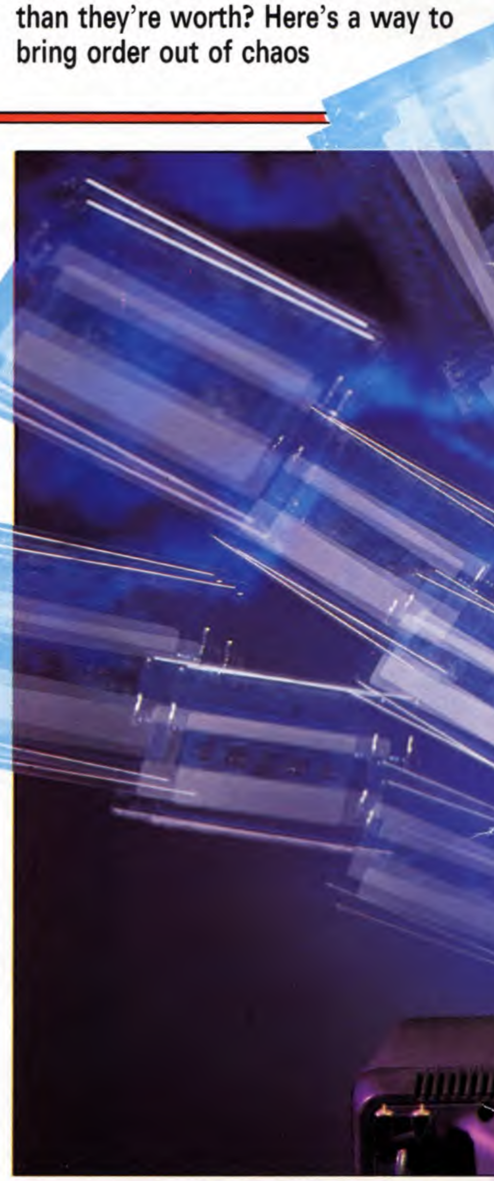
Given a choice between binary and decimal, binary is usually better. It allows you to alter individual lines of numbers, or even a single 1 or Ø, until you are quite satisfied with your tiny picture.

A decimal conversion is worth doing, however, if you want to remember the graphic for later—or repeated—use.

Given a choice between decimal and hex, the latter is much quicker. It has another advantage, too: it helps make you familiar with hex itself—the language of machine code programming.

STREAMLINE YOUR HOBBIES FILES

Is your address book in a mess? Are your club files disorganized or your cassette collection files more trouble than they're worth? Here's a way to bring order out of chaos



'But what does it actually *do?*' is a question that people who don't have home computers constantly ask those who do. And short of saying, 'They're for doing home computing on', there is usually no satisfactory answer.

But here is a program that does make your computer do some useful work. It is a computer filing system which is so flexible it has dozens of applications in everyday life. You can use it to store the names and addresses of friends or the members of a club, or to keep track of family and friends' birthdays, or to store the details of coin, butterfly or recipe collections, or even to keep track of your growing collection of computer games.

The only limit to what you can do with this program lies in the size of your computer's RAM memory. For most jobs, you will find that a 32K machine is a practical minimum. There are thus no programs for the ZX81, Vic 20, or Spectrum 16K. And anyway, you should remember this: because home computers' memories are small compared with those of business machines, the shorter you can keep each entry the better.

THE MAIN MENU

Once you have typed in the program and RUN it, it will automatically PRINT on the screen the main menu. This is a list of the things you can do to the file. You can 'enter a record', for example, or 'search the file'.

But first of all you have to 'open a file' and feed some records into it.

OPENING A FILE

As you will already have discovered, computers need precise details of what you want before they'll do anything at all.

To open a new file you first need to tell the computer the number of records you want, and the maximum length each record can be. 'OPEN A FILE' is option 1 on the main menu, so to select it you press the 1 key. The words 'Are you sure?' will then flash up on the screen. This is a precaution against your accidentally pressing the 1 key—because if the filing system is already storing DATA, going into the 'open a file' routine would destroy it.

If you *are* sure that you want to open a new file, press Y. But if the file is already storing information that you want to keep, press any key other than Y—N for example—and the computer will automatically return you to the main menu.

HOW LONG A FIELD?

Once you have pressed Y to continue, the computer will ask you how many fields you want. *Fields* are the items of information you want stored in each record. For example, if you are a keen train spotter the fields you would need might be: locomotive class; number; date when seen; and place where seen—four in all.

The maximum number of fields in any individual record is eight; otherwise you could not display them all on the screen at the same time.

With the number of fields entered, the computer's next question will be, 'Name of first field?'. (In the example above, your answer would be 'CLASS'.)

Then you will be asked the length of the first field—that is, the maximum number of characters that the first field is to hold.

The maximum length of field allowed in the program is 19 characters (27 on the Acorn machines). This means that if the information you want to file—an address, say—will not fit into this, you will have to divide the field into two or more pieces. With an address you could do this by initiating separate fields for number/street and city/postcode.

Once it has the information about the first field, the computer will ask the same questions about the second field, third field and so on. Obviously, the shorter you can keep both the names of fields and the number of characters in each, the more records your file can hold.

This done, the computer will quickly work out how many records it has room for. This number will be displayed on the screen.

On the Spectrum only, you will next be asked to specify how many records you actually want. Otherwise, if the number of records you need is much smaller than the permitted maximum, you will have a problem when you SAVE the file on tape. The Spectrum will spend a lot of time recording unused memory.

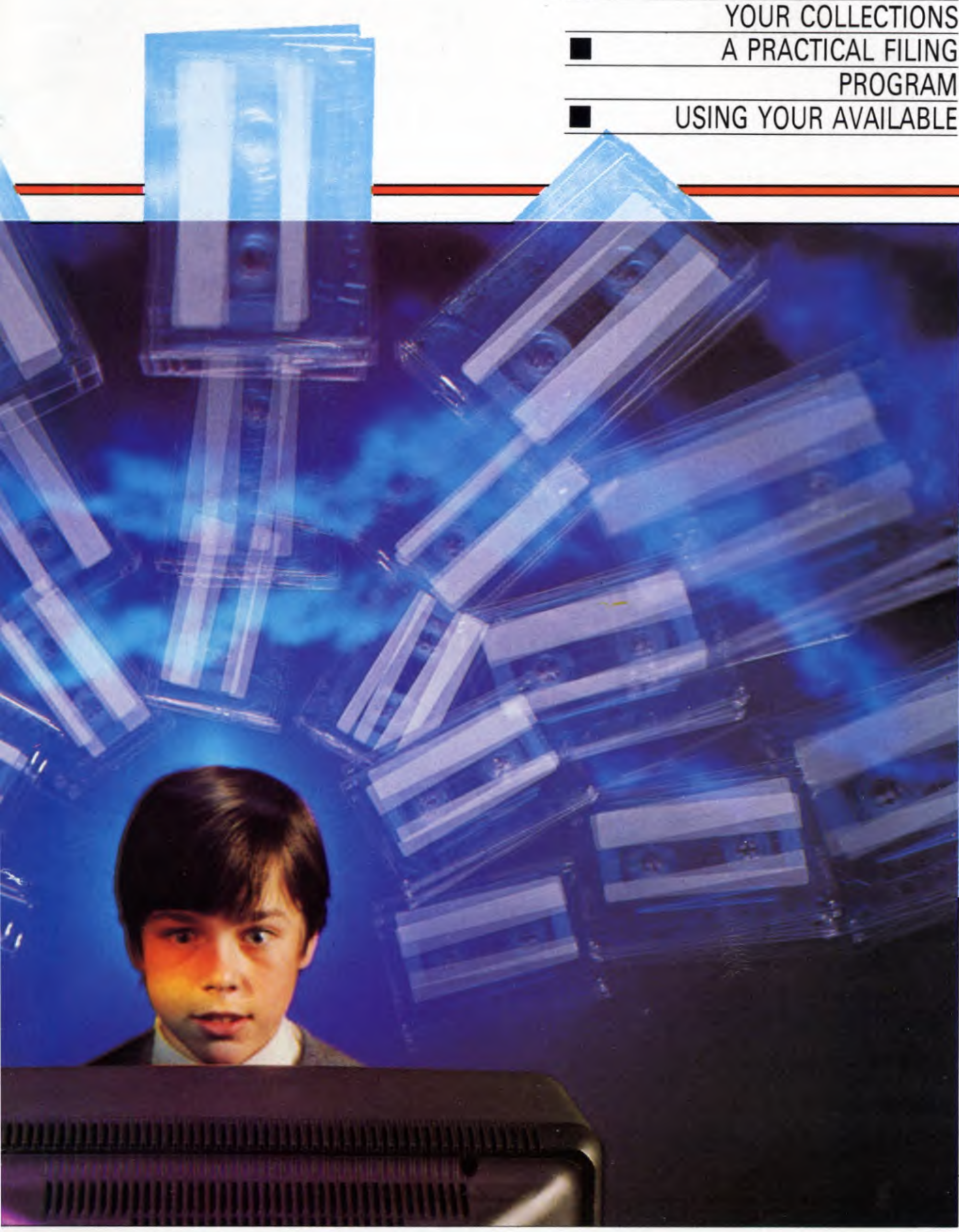
ENTERING A RECORD

Once you have completed the opening-a-file procedure, the program will automatically take you back to the main menu, where you select option 2—by pushing the 2 key—to start entering your records.

At the top of the screen the computer will keep a running tally of how many records you have entered, along with the total space in the store. It will say: 'You have used 1Ø out of 1ØØ records' or whatever the numbers are.

Under that, the computer will display the field names. At the bottom where the cursor is, write in the details you want recorded

- KEEPING TRACK OF YOUR COLLECTIONS
- A PRACTICAL FILING PROGRAM
- USING YOUR AVAILABLE MEMORY EFFICIENTLY
- SETTING UP A NEW FILE
- ADDING TO AND VIEWING YOUR FILE
- STORING AND RECALLING YOUR INFORMATION



under each field heading. Remember to keep them as short as possible and within the maximum character length you have set.

When you press the ENTER or RETURN key the information you have keyed in will be PRINTed out next to the field name. The bottom of the screen will be cleared, ready for you to key in the next piece of information.

This method starts with the first field at the top of the screen and works its way down the screen each time you key in information and press RETURN or ENTER. When you have filled in the last field on the record the computer will move on to the next—blank—record.

If you hit the RETURN or ENTER key again before you start filling in the first field, the computer will take you back to the main menu.

VIEWING THE RECORDS

To look over the records you have entered, you select option number 3 on the main menu, 'VIEW RECORDS', by pressing key 3. The screen will then display the first record—not necessarily the first one you put in, but the first one according to the program's own selection method.

Computers' methods of arranging alphabetical order vary slightly. But broadly, they select the records in alphabetical order by the first field, which in many cases will be 'NAME'. To do this, the computer looks at the first entry in the first field and orders the records alphabetically. If more than one record has the same first letter, it orders them by the second letter. And then by the third, and so on.

The first problem arises when you have numbers in the first field. The computer will select any number before any letter, but it goes through the same ordering method digit by digit when deciding between numbers, rather than looking at the number as a whole. In other words, if you fed in records with the first fields carrying the numbers from 1 to 1ØØ, the computer would select 1, 1Ø, 11, 12, 13, 14, 15, 16, 17, 18, 19 and 1ØØ before it got around to 2, 2Ø, 21 and so on.

The way round this is to number the records ØØ1, ØØ2 .... Ø1Ø, Ø11 ... up to 1ØØ. Or, of course, not to use numbers in the first field at all.

The second problem arises if you use a mixture of capital and lower case letters, because the computer chooses capitals ahead of lower case. So 'ABC Limited' would be ahead of 'Aaron and Co.' Depending on what your datafile is to contain, you may find it convenient to list everything in capitals to get round this problem.

When viewing the records you will find:

F(ORWARD) B(ACK) M(ENU)

written near the bottom of the screen. If you press the F key, the computer will display the next record, and if you press F repeatedly, it will flip through the whole file record by record.

Pressing the B key takes you back to the record before the one on the screen. So between the F and the B you can run backwards and forwards through the file.

Pressing M will return you to the main menu at any point.

Underneath the 'F(ORWARD) B(ACK) M(ENU)' line, you will find:

A(MEND) D(ELETE) P(RINTER)

These are explained in the next article in this series, pages 75 to 79.

SAVEING AND LOADING

As you can see, the main menu gives you SAVE and LOAD options—5 and 6. These are in the main menu because, except on the Spectrum, you have to store the DATA contained in the file separately from the datafile program itself.

When you want to SAVE your file, you press 5 and the computer will ask you to give the file a filename. Once you have keyed in the filename and pressed [RETURN] or [ENTER], the computer will tell you that it is 'SAVING INFORMATION NOW'.

On the Commodore, Dragon and Acorn computers, the DATA alone will now be stored. To SAVE the program, you will have to select option 7 on the main menu, 'QUIT PROGRAM', by pressing key 7 and then go into the normal SAVE routine for your machine.

When, at some time in the future, you want to consult your files, you will (on Commodore, Acorn and Dragon) have to LOAD in two stages. First you LOAD the program using your machine's normal LOAD routine. Then you LOAD the DATA by selecting option 6, 'LOAD FILE', and pressing the 6 key. The computer will then ask you for the name of the file you want to see. When you have keyed in the filename and pressed the [RETURN] key, the machine will tell you to 'PRESS PLAY AND ANY KEY'.

The computer will search down the tape as it runs until it finds the file you want, which it LOADs. It will then tell you that the file has been 'LOADED CORRECTLY'. If the file you want isn't on the tape, the computer will simply list all the files that are. In either case it will then take you back to the main menu. At that point, or at any time later when you have the main menu in front of you, you can select option 6 and LOAD another file.

On the Spectrum, the DATA and the main program LOAD together. Once you have LOADed the first file using your standard LOAD method, you can LOAD any subsequent files simply by selecting option 6 on the main menu.

AMENDING AND DELETING

The programs for the individual machines as given below have huge gaps in their line numbers—from Line 2ØØØ-odd to Line 6ØØØ.

But number them as they are given here. The missing lines are for amending a record, deleting a record and for cross-referencing—the function at which a computer is so much more efficient than a mechanical system.

Details of these options are in the next article in this series when some temporary lines present here will be overwritten.

```
2 printchr$(8):gosub6:goto1ØØ
6 dimfc(7,1):fori=1to7:readfc(i,Ø):fc(i,1)=-1
   :next:data-1,,,,,-1,-1
12 dimof(3):of(Ø)=64:of(1)=Ø:of(2)=
   128: of (3)=64
14 dimlx(8),hx(8)
2Ø vic=Ø
22 bd=5328Ø:bg=53281:bb=Ø
24 cc$=chr$(5):bc=Ø
28 sb=1Ø24:LL=4Ø:sh=25
4Ø pokebg,Ø:pokebd,Ø
42 printchr$(14)
44 gr$=chr$(3Ø):pu$=chr$(156):yl$=
   chr$(158)
46 cs$=chr$(147):ch$=chr$(19)
48 cd$=chr$(17):cu$=chr$(145)
5Ø rv$=chr$(18):ro$=chr$(146)
52 cl$=chr$(157):cr$=chr$(29):c4$=
   cd$+cd$+cd$+cd$
56 dl$=chr$(2Ø):d4$=dl$+dl$+dl$+dl$:
   is$=chr$(148)
58 rt$=chr$(13)
6Ø qt$=chr$(34):cm$=chr$(44)
7Øul$=right$("⊟⊟⊟⊟⊟⊟⊟⊟⊟
   ⊟⊟⊟⊟⊟⊟⊟⊟⊟⊟⊟⊟
   ⊟⊟⊟⊟⊟⊟⊟⊟⊟⊟⊟⊟
   ⊟⊟⊟⊟⊟",LL)
72 x1$=cd$+rv$+cc$:x2$=ro$+
   "☐"+gr$+rv$:x3$=ro$+"☐☐☐"
   +rv$+cc$
74 x4$=x4$+cu$
76 dimm$(7):fori=1to7:readm$(i):next
78 data"☐Open a file☐☐☐☐","☐Enter
   records☐☐","☐View records☐☐☐"
8Ø data"☐Search records☐☐☐","☐Save
   tape file☐☐","☐Load tape file☐☐"
82 data"☐Quit program☐"
84 dimau$(6):fori=1to6:readau$(i):next
```

```
86 data"Forward","□Back□□",
   "□Menu□□","□Amend□",
   "Delete□","□Print□"
96 w1$=rv$+pu$+"□ARE YOU SURE
   (y/n)□?□"+cc$+ro$:return
1ØØ printcs$+cc$+ul$
11Ø printpu$"□*□□□*□□*
   □□*□*□□*□□□*□□□
   *□***□*□□*□*□□*"
12Ø print"□**□**□*□*□
   *□**□*□□□**□**□
   *□□□**□*□*□□*"
13Ø print"□*□*□*□***□*
   □*□**□□□*□*□*□
   **□□*□**□*□□*"
14Ø print"□*□□□*□*□*
   □*□*□□*□□□*□□□*□
   *□□□*□□*□*□□*"
15Ø print"□*□□□*□*□*□
   *□*□□*□□□*□□□*□
   ***□*□□*□□**"
16Ø printcc$ul$
5ØØ printx1$"1"x2$m$(1)x3$;
51Ø iffd=Øthen 55Ø
52Ø print"4"x2$m$(4)
53Ø printx1$"2"x2$m$(2)x3$"5"x2$m$(5)
54Ø printx1$"3"x2$m$(3)x3$;
55Ø print"6"x2$m$(6)
56Ø printtab(11)x1$"7"x2$m$(7)c4$
6ØØ printtab(11)x4$"□SELECT□□
   OPTION□"ro$:ford=1to 25Ø:next
61Ø geta$:ifa$=""then65Ø
62Ø a=asc(a$)-48:ifa<1ora>7thengosub
   1ØØØØ:goto65Ø
63Ø goto7ØØ
65Ø printtab(11)x4$rv$"□SELECT□□
   OPTION□":ford=1to25Ø:next
66Ø geta$:ifa$=""then 6ØØ
67Ø a=asc(a$)-48:ifa<1ora>7thengosub
   1ØØØØ:goto6ØØ
7ØØ ifnotfc(a,-fd)thengosub1ØØØØ: goto6ØØ
8ØØ iffc(a,Ø)=Øorfd=Øthen 89Ø
81Ø ix$=m$(a):gosub145ØØ
82Ø ifaa$<>"y"then1ØØ
83Ø ifa=7then9ØØ
84Ø poke631,a:clr:gosub6:a=peek(631):
   goto7ØØ
89Ø ifu=Øand((a=3)or(a=4))thengosub
   1ØØØØ:goto6ØØ
9ØØ onagosub1ØØØ,2ØØØ,398Ø,4ØØØ,95Ø,
   95Ø,7ØØØ
91Ø goto1ØØ
95Ø printro$cs$gr$tab(11)m$(a)c4$
96Ø print"Name of file:?":x=16:y=5-vic:
   z=1Ø:gosub13ØØØ:f$=ix$
972 iff$=""then95Ø
98Ø ifa=6then6ØØØ
99Ø ifa=5then5ØØØ
1ØØØ printcs$cd$"Number of fields (1-8):?□";
1Ø1Ø ok$="12345678":gosub1Ø6ØØ:nf=ix:
   printaa$cd$:y=3-vic:tt=5-sh-2*vic
1Ø2Ø forn=1tonf
1Ø3Ø print "Enter heading"n"□:?□";
1Ø4Ø x=2Ø:z=1Ø:gosub13ØØØ
1Ø45 iflen(ix$)=Øthenprintro$rt$cu$cu$:
   goto1Ø3Ø
1Ø5Ø hd$(n)=ix$
1Ø6Ø printtab(13):printd4$d4$d4$:x=Ø:
   gosub115ØØ
1Ø7Ø print"Enter field□"ix$"□length:?"
1Ø8Ø x=23+len(hd$(n)):z=2:gosub13ØØØ:
   ifix$=""thengosub1ØØØØ:goto1Ø8Ø
1Ø9Ø gosub12ØØØ:hx(n)=2+int((12+ix)/ll)
1Ø92 ifix<1orhx(n)>3thengosub1ØØØØ:
   goto1Ø8Ø
1Ø94 tt=tt+hx(n)
1Ø96 lx(n)=ix:y=y+2:print:print
1Ø98 next
11ØØ ln=Ø:fori=1tonf:ln=ln+lx(i):next:
   fr=fre(Ø):iffr<Øthenfr=fr+65536
111Ø v=int(fr/(ln+5+3*nf))
112Ø print"You can use□"v"□records":
   ford=1to15ØØ:next
113Ø dimt$(v,nf-1),r(v)
114Ø uØ=Ø
12ØØ foru=uØtov
121Ø printcs$cc$rv$"You have used"u"
   ▮□out of"v"▮□records"cd$
122Ø up=u:r(up)=up
123Ø forix=1tonf
124Ø gosub372Ø
125Ø ifix=1andix$=""then14ØØ
126Ø fori=1to5ØØ:next
128Ø next
13ØØ ifu=Øthen134Ø
13Ø2 ix$=t$(u,Ø):ru=u:su=u
131Ø foru2=Øtou-1
132Ø ift$(r(u2),Ø)>ix$then135Ø
133Ø next
134Ø u2=su:goto138Ø
135Ø fordn=utou2+1step-1
136Ø r(dn)=r(dn-1)
137Ø next
138Ø r(u2)=ru:ifa>2thenup=u2:
   printch$cc$rv$"□THIS IS RECORD□"
   up+1:goto31ØØ
139Ø nextu
14ØØ fd=-1
199Ø return
2ØØØ uØ=u:b=1
21ØØ goto12ØØ
3ØØØ uØ=up-1
3Ø1Ø ifuØ<ØthenuØ=u-1:ifa=4then392Ø
3Ø2Ø forup=uØtou-1
3Ø3Ø ifa=4then411Ø
3Ø4Ø printcs$cc$rv$"□THIS IS RECORD□"
   up+1
3Ø5Ø forix=1tonf:gosub377Ø:next
31ØØ x=Ø:y=sh-2+2*vic:gosub115ØØ
311Ø fori=1to6:printx3$x2$au$(i);:
   ifi=3thenprintro$"□□□□□□□□";
312Ø next
32ØØ ok$="fbmadp□":gosub1Ø6ØØ
321Ø b=ix
33ØØ printro$;:onbgoto39ØØ,3ØØØ,199Ø,
   37ØØ,34ØØ,36ØØ,39ØØ
34ØØ goto1ØØ
36ØØ gosub372Ø:print"Is printer ready
   (y/n)";:gosub1Ø5ØØ
361Ø ifaa$="n"thengosub372Ø:goto31ØØ
362Ø open4,4,7:cmd4
363Ø print#4,"□this record□"up+1
   "□used□"u"□records"
364Ø forn=1tonf:print#4:print#4,"□"
   hd$(n)":"spc(12-len(hd$(n)))t$(r(up),
   n-1)
365Ø next:print#4,ul$:close4:goto31ØØ
37ØØ goto1ØØ
372Ø y=sh-2+2*vic:x=Ø :gosub115ØØ:
   printro$;:z=ll*(2-2*vic)-2:gosub
   135ØØ
3722 ifb=6thenreturn
373Ø printhd$(ix)ro$"□:";:x=12:
   z=lx(ix):gosub115ØØ
376Ø gosub13Ø1Ø:t$(r(up),ix-1)=ix$
377Ø y=2-vic:ifix>1thenforn=1toix-1:
   y=y+hx(n)+(n<=tt):next
378Ø x=Ø:gosub115ØØ:printro$hd$(ix)"□:"
   tab(13);:ifa>2thengosub135ØØ
379Ø printt$(r(up),ix-1):return
38ØØ ifix>1then31ØØ
381Ø ifup>Øthenift$(r(up),Ø)<t$(r(up-1),Ø)
   then383Ø
382Ø ifup=u-1ort$(r(up),Ø)<=
   t$(r(up+1),Ø)then31ØØ
383Ø ix$=t$(r(up),Ø):ru=r(up):
   ifup=u-1then385Ø
384Ø fordn=uptou-1:r(dn)=r(dn+1):next
385Ø su=u-1
3855 goto131Ø
```

Microtip

Making long programs easier

Typing in long programs which someone else has written can be a laborious—even daunting—task. But you can make it easier by typing only a short section at a time, then checking it. Many programs are structured into more or less self-contained sections, such as subroutines, procedures and loops, but if you cannot see such a structure then just use short sections of 20 or 30 lines. Fortunately, the more you learn about programming and the better you understand each line, the fewer mistakes you are likely to make.

```
39ØØ nextup
391Ø ifa<>4then398Ø
392Ø printcs$x1$"□END OF FILE WHILE
   SEARCHING"
393Ø printx1$"□DO YOU WISH TO TRY
   FROM START (y/n)?":gosub1Ø5ØØ
3935 if aa$="y" then goto 4ØØØ
394Ø ifaa$<>"y"thenreturn
395Ø ifb=2then3Ø2Ø
398Ø uØ=Ø:b=1
399Ø goto3Ø1Ø
4ØØØ return
411Ø ix$=t$(r(up),fx-1)
412Ø fe=len(ix$)-ff+1
413Ø iffe<1then416Ø
414Ø forj=1tofe:ifmid$(ix$,j,ff)=fx$ then3Ø4Ø
415Ø next
416Ø ifb=2then3ØØØ
417Ø goto39ØØ
5ØØØ printcs$rv$"□□□□□□□□
   POSITION TAPE FOR OUTPUT□□□□
   □□□□"
5ØØ5 print"□□□□□□Press return key
   when ready."
5Ø1Ø geta$:ifa$<>rt$then5Ø1Ø
51ØØ open 1,1,1,f$
511Ø print"□Saving information now□"
512Ø print#1,u;cm$;nf;cm$;tt
513Ø forn=1tonf:print#1,qt$hd$(n)qt$cm$
   lx(n)cm$hx(n):next
514Ø forup=Øtou:forn=1tonf:print#1,qt$
   t$(up,n-1)qt$:next:print#1,r(up):next
515Ø close1
599Ø return
6ØØØ print cs$rv$"□□□□□□□□
   POSITION TAPE FOR OUTPUT□□□
   □□□□□"
6ØØ5 print"□□□□□□Press return key
   when ready."
6Ø1Ø getaa$:ifaa$<>rt$then6Ø1Ø
61ØØ open 1,1,Ø,f$
611Ø print"found and loading"
612Ø input#1,u,nf,tt
613Ø forn=1tonf:input#1,hd$(n),lx(n),
   hx(n):next
614Ø ln=Ø:forn=1tonf:ln=ln+lx(n):next:
   fr=fre(Ø):iffr<Øthenfr=fr+65536
615Ø v=int(fr/(ln+5+3*nf))
616Ø dimt$(v,nf-1),r(v)
62ØØ forup=Øtou:forn=1tonf:input#1,
   t$(up,n-1):next
621Ø input#1,r(up):next
622Ø close1
698Ø fd=-1
699Ø return
7ØØØ printcs$:end
1ØØØØ poke54277,33:poke54278,255:
   poke54273+23,15
1ØØØ5 poke54273,6:poke54276,33:
   ford=1to5Ø:next
1ØØØ6 poke54273+23,Ø
1ØØ1Ø return
1Ø5ØØ ok$="yn"
1Ø6ØØ getaa$:ifaa$=""then1Ø6ØØ
1Ø61Ø ix=Ø:fori=1tolen(ok$):ifaa$=
   mid$(ok$,i,1)thenix=i
1Ø62Ø next:ifix=Øthengosub1ØØØØ:goto1Ø6ØØ
1Ø63Ø return
115ØØ printch$;
1151Ø ify>Øthenforyy=1toy:printcd$;:next
1152Ø ifx>Øthenforxx=1tox:printcr$;:next
1153Ø return
12ØØØ ix=-1:fori=1tolen(ix$)
12Ø1Ø a$=mid$(ix$,i,1)
12Ø2Ø ifa$<>"□"then12Ø5Ø
12Ø3Ø ifi=1ori=len(ix$)then12Ø6Ø
12Ø4Ø ifmid$(ix$,i-1,1)="□"then12Ø6Ø
12Ø5Ø ifa$<"Ø"ora$>"9"thengosub1ØØØØ:
   return
12Ø6Ø next
12Ø7Ø ix=val(ix$):return
13ØØØ gosub115ØØ:gosub135ØØ
13Ø1Ø pØ=sb+ll*y+x:p1=pØ:i=128:
   ix$=""
13Ø2Ø pokep1,(peek(p1)and127)or
   (iand128)
13Ø3Ø geta$:i=(i+12)and255:
   ifa$=""then13Ø2Ø
13Ø4Ø ifa$=dl$then1315Ø
13Ø5Ø ifa$=rt$thenreturn
13Ø6Ø ifasc(a$)and127<32then1319Ø
131ØØ ifp1>=pØ+zthen1319Ø
1311Ø pokep1,peek(p1)and127:p1=p1+1:
   printa$;:ix$=ix$+a$:goto13Ø2Ø
1315Ø ifp1=pØthen1319Ø
1316Ø pokep1,peek(p1)and127:p1=p1-1:
   printcl$"□"cl$;:ix$=left$(ix$,p1-pØ)
1317Ø goto13Ø2Ø
1319Ø gosub1ØØØØ:goto13Ø2Ø
135ØØ fori=1toz:print"□";:next
1351Ø fori=1toz:printcl$;:next
1352Ø return
145ØØ b=6:gosub372Ø:printcd$"□□□
   □□□□□"pu$w1$yl$rv$cu$
   "█████████████████████
   ██████████████□"ix$;
1453Ø gosub1Ø5ØØ
1454Ø return
```

The □ symbol denotes an important space. Enter on the space key, not as a graphic.

```
5 LET R=Ø: LET U=Ø: LET V=1
1Ø BORDER V: PAPER V: INK 7: POKE
   236Ø9, 2Ø: POKE 23658,8
1ØØ CLS : PRINT INVERSE V;AT V,6;"□M
   □A□I□N□□□M□E□N□U□"
11Ø PRINT AT 5,6;"1 :— Open a
   file"''TAB 6;"2 :— Enter a record"'' TAB 6;
   "3 :— View records"''TAB 6;"4 :— Search
   option"''TAB 6;"5 :— Save file"''TAB 6;
   "6 :— Load file"''TAB 6;"7 :—Quit pro-
   gram"; #V;TAB 6;"— SELECT OPTION —"
5ØØ LET I$=INKEY$: IF I$="" THEN GOTO
   5ØØ
51Ø IF I$<"1" OR I$>"7" THEN GOTO 5ØØ
52Ø IF R=U AND I$<>"1" AND I$<>
   "6" AND I$<>"7" THEN GOTO 5ØØ
53Ø BEEP .1,1Ø: CLS : GOSUB (CODE
   I$-48)*1ØØØ: GOTO 1ØØ
1ØØØ PRINT AT 7,9;"ARE YOU SURE ?":
   PAUSE U: IF INKEY$="" THEN
   GOTO 1ØØØ
1Ø1Ø IF INKEY$<>"Y" THEN RETURN
1Ø2Ø PRINT INVERSE V;AT 1Ø,6;"□CREATE
   A NEW FILE□"
1Ø3Ø INPUT AT Ø,Ø;"Number of fields
   (1–8)?□";A: IF A<1 OR A>8 THEN
   GOTO 1Ø3Ø
1Ø4Ø DIM A(A): DIM B(A+V): DIM
   N$(A,1Ø):LET T=U: FOR N=V TO A
1Ø5Ø INPUT AT Ø,Ø;"Name of
   field□";(N);"□?□"; LINE N$(N)
1Ø6Ø INPUT AT V,Ø;"Length of
   field□";(N);"□?□";A(N): IF
   A(N)>5Ø THEN GOTO 1Ø6Ø
1Ø7Ø LET B(N)=T: LET T=T+A(N): NEXT
   N:LET B(N)=T
1Ø8Ø PRINT AT 16,2;"Room for
   about□";INT(((PEEK 2373Ø+256*PEEK
   23731)–295ØØ)/T);"□records"
1Ø9Ø INPUT "How many records ?□";R:
   DIMA$(R,T): RETURN
2ØØØ LET C=V
2Ø1Ø IF A$(C,V)="□" THEN GOTO 21ØØ
2Ø2Ø IF C=R THEN GOTO 25ØØ
2Ø3Ø LET C=C+V: GOTO 2Ø1Ø
21ØØ PRINT AT Ø,Ø;C-V;"□out
   of□";R;"□records in use"
211Ø FOR N=V TO A: PRINT INVERSE
   V;AT V+N*2,U;N$(N); INVERSE Ø;AT
   V+N*2,12; FLASH V;"?": INPUT "(up
   to□";(A(N));"□characters)", LINE
   A$(C,B(N)+V TO B(N+V)): PRINT AT
   V+2*N,12;A$(C,B(N)+V TO B(N+V)):
   NEXT N
212Ø FOR F=V TO 15Ø: NEXT F: IF C=V
   THEN RETURN
213Ø LET N=C
214Ø IF A$(C)>=A$(C-V) THEN RETURN
215Ø LET X$=A$(C): LET A$(C)=A$(C-V):
   LET A$(C-V)=X$: LET C=C-V: IF
   C=V THEN RETURN
216Ø GOTO 214Ø
25ØØ CLS : PRINT FLASH 1;AT 1Ø,6;"□F□
   I□L□E□□□F□U□L□L□":
   FOR F=V TO 4ØØ: NEXT F: RETURN
3ØØØ LET D=V: IF A$(V,V)="□" THEN
   RETURN
3Ø1Ø IF D=U THEN LET D=V
3Ø15 IF D-V=R THEN LET D=D-V
3Ø2Ø IF A$(D,V)="□" THEN LET D=D-V
```

```
3Ø3Ø GOSUB 95ØØ
3Ø4Ø IF OP=V THEN LET D=D+V: GOTO
   3Ø1Ø
3Ø5Ø IF OP=2 THEN LET D=D−V: GOTO
   3Ø1Ø
3Ø6Ø IF OP=3 THEN RETURN
3Ø7Ø IF OP=4 THEN GOSUB 8ØØØ
3Ø8Ø IF OP=5 THEN LET MD=V: GOSUB
   9ØØØ:IF D=U THEN RETURN
3Ø9Ø GOTO 3Ø3Ø
4ØØØ RETURN: REM TEMPORARY LINE
5ØØØ INPUT "Enter file name□□"; LINE
   Q$:IF LEN Q$<V OR LEN Q$>1Ø
   THEN GOTO 5ØØØ
5Ø1Ø SAVE Q$ LINE 1Ø: RETURN
6ØØØ PRINT AT 8,U;"Enter name of file to be
   loaded, or just ENTER to load first file"
6Ø1Ø INPUT LINE X$: IF LEN X$>1Ø
   THEN GOTO 6Ø1Ø
6Ø2Ø PRINT AT 13,U;"PRESS PLAY
   ON CASSETTE RECORDER": LOAD X$
7ØØØ PRINT AT 1Ø,8;"Are you sure ?":IF
   INKEY$="" THEN GOTO 7ØØØ
7Ø1Ø IF INKEY$<>"Y" THEN RETURN
7Ø2Ø RANDOMIZE USR U
8ØØØ RETURN: REM TEMPORARY LINE
9ØØØ RETURN: REM TEMPORARY LINE
95ØØ PRINT AT U,U;"Record number□";D;
   "□□": FOR N=V TO A: PRINT
   INVERSE V;AT V+2*N,U;N$(N); INVERSE
   U;TAB 12;A$(D,B(N)+V TO
   B(N+V)): NEXT N
951Ø PRINT INVERSE V;AT 2Ø,U;"□F(orward)
   □□B(ack)□□□□M(enu)□□□□□
   A(mend)□□□□
   D(elete)□□P(rinter)□"
952Ø IF INKEY$="" THEN GOTO 952Ø
953Ø LET V$=INKEY$: IF V$="P" THEN
   COPY :LPRINT : LPRINT : LPRINT :
   GOTO 952Ø
954Ø LET OP=U: IF V$="F" THEN LET
   OP=V:LET MO=V
955Ø IF V$="B" THEN LET OP=2: LET
   MO=−V
956Ø IF V$="M" THEN LET OP=3
957Ø IF V$="A" THEN LET OP=4
958Ø IF V$="D" THEN LET OP=5
959Ø IF OP=U THEN GOTO 952Ø
96ØØ BEEP .1,1Ø: RETURN
```

```
2Ø PCLEAR1:CLEAR 11ØØØ:RS$="F□
   BMADP":B$=CHR$(128)
3Ø CLS:PRINT@39,B$;"m";B$;"a";
   B$;"i";B$;"n";B$;B$;B$;"m";B$;"e";
   B$;"n";B$;"u";B$
35 POKE 144,3
4Ø PRINT@164,"1 :—OPEN A FILE"
5Ø PRINT@196,"2 :— ENTER A RECORD"
6Ø PRINT@228,"3 :— VIEW RECORDS"
7Ø PRINT@26Ø,"4 :— SEARCH OPTION"
8Ø PRINT@292,"5 :— SAVE FILE TO TAPE"
9Ø PRINT@324,"6 :— LOAD FILE FROM TAPE"
1ØØ PRINT@356,"7 :— QUIT PROGRAM"
11Ø PRINT@481,"SELECT OPTION :";
12Ø IN$=INKEY$:IFIN$<"1"ORIN$>
   "7" THEN12Ø
13Ø IFIN$<>"1" ANDIN$<>"6"AND
   R=Ø ANDIN$<>"7"THEN12Ø
14Ø SOUND3Ø,1:CLS:IN=VAL(IN$)
15Ø ON IN GOSUB1ØØØ,2ØØØ,6ØØØ,5ØØØ,
   7ØØØ,8ØØØ,9ØØØ
16Ø GOTO3Ø
1ØØØ PRINT@41,"SET UP NEW FILE":
   PRINT@231,"ARE YOU SURE (Y/N)?"
1Ø1Ø IN$=INKEY$:IFIN$<>"Y"AND
   IN $<>"N" THEN1Ø1Ø
1Ø2Ø IFIN$<>"Y" THENRETURN
1Ø3Ø IFR>Ø THENRUN92ØØ
1Ø4Ø CLS:PRINT@41,"SET UP NEW FILE"
1Ø5Ø PRINT@385,"NUMBER OF FIELDS
   (1-8)";:INPUTA:A=ABS(INT(A))
1Ø6Ø IFA>8 ORA<1 THEN1Ø5Ø
1Ø7Ø DIM A(A),N$(A)
1Ø8Ø PRINT@384,"":PRINT@96,"":
   FORN=1TOA
1Ø9Ø PRINT:PRINT" NAME OF FIELD";
   N;"?";:LINEINPUTN$(N):N$(N)=
   LEFT$(N$(N),1Ø)
11ØØ PRINT"LENGTH OF FIELD";
   N;:INPUTA(N):A(N)=ABS(INT(A(N)))
111Ø IFA(N)>19 OR A(N)<1 THEN11ØØ
112Ø TS=TS+A(N)
113Ø NEXT:R=INT(11ØØØ/(5+5*A))−1:
   PRINT"□MAX NUMBER OF
```

```
   RECORDS=";R
1140 DIMA$(R,A):FORI=1TO2ØØØ:
   NEXT:RETURN
2ØØØ G=Ø
2Ø1Ø IFNR=R THEN218Ø
2Ø2Ø NR=NR+1
2Ø3Ø CLS:PRINT@Ø,NR-1;"□OUT
   OF";R;"RECORDS IN USE"
2Ø4Ø FORN=1TOA:PRINT@32*N+32,
   N$(N);"□:":PRINT@448,"":PRINT@
   416,""
2Ø5Ø PRINT@416,"(UP TO";A(N);
   "CHARACTERS)□□";:LINEINPUTA$
   (NR,N)
2Ø6Ø IFA$(NR,N)="" AND N=1
   THENN=A:G=1:GOTO2Ø8Ø
2Ø7Ø A$(NR,N)=LEFT$(A$(NR,N),
   A(N)):PRINT@32*N+45,A$(NR,N)
2Ø8Ø NEXT
2Ø9Ø IFG=1 THEN 216Ø
21ØØ C=NR:FORF=1TO15Ø:NEXT:
   IFNR=1 THEN215Ø
211Ø IFA$(C,1)>=A$(C-1,1) THEN215Ø
212Ø FORN=1TOA:X$=A$(C,N):
   A$(C,N)=A$(C-1,N):
   A$(C-1,N)=X$: NEXT: C=C-1
213Ø IFC=1 THEN215Ø
214Ø GOTO211Ø
215Ø GOTO2Ø1Ø
216Ø NR=NR-1
217Ø RETURN
218Ø CLS3:PRINT@235,"□FILE FULL□"
   ;:FORG=1TO5:SCREENØ,1:FORF=1TO
   5ØØ: NEXT
219Ø SCREENØ,Ø:FORF=1TO5ØØ:
   NEXTF,G:RETURN
3ØØØ RETURN: REM TEMPORARY LINE
4ØØØ RETURN: REM TEMPORARY LINE
5ØØØ RETURN: REM TEMPORARY LINE
6ØØØ D=1
6Ø1Ø IFNR<1 THEN617Ø
6Ø2Ø GOSUB85ØØ
6Ø3Ø PRINT@451,"fORWARDS□□□□
   bACKWARDS□□□mENU□□aMEND
   □□□□□dELETE
   □□□□□pRINT";
6Ø4Ø IN$=INKEY$:IFIN$="" THEN6Ø4Ø
6Ø5Ø IN=INSTR(1,RS$,IN$)
6Ø6Ø ON IN GOTO 6Ø8Ø,6Ø8Ø,6Ø9Ø,61ØØ,
   611Ø,612Ø,613Ø
6Ø7Ø GOTO6Ø3Ø
6Ø8Ø D=D+1:GOTO614Ø
6Ø9Ø D=D-1:GOTO614Ø
61ØØ RETURN
611Ø GOSUB3ØØØ:GOTO6Ø2Ø
612Ø GOSUB4ØØØ:GOTO6Ø1Ø
613Ø GOSUB1ØØØØ:GOTO6Ø3Ø
614Ø IFD>NR THEND=1
615Ø IFD<1 THEND=NR
616Ø GOTO6Ø1Ø
617Ø CLS3:PRINT@233,"□FILE EMPTY□";
618Ø FORG=1TO5:SCREENØ,1:FORF=1
   TO3ØØ:NEXT:SCREENØ,Ø:FORF=1
   TO3ØØ:NEXTF,G:RETURN
7ØØØ AUDIOON:MOTORON:CLS:PRINT@
   65,"POSITION TAPE THEN PRESS
   ENTER";
7Ø1Ø IN$=INKEY$:IFIN$<>CHR$(13)
   THEN7Ø1Ø
7Ø2Ø MOTOROFF:PRINT@129,"PLACE
   RECORDER INTO RECORD MODE THEN
   PRESS ENTER";
7Ø3Ø IN$=INKEY$:IFIN$<>CHR$(13)
   THEN7Ø3Ø
7Ø4Ø PRINT:INPUT"□FILE NAME□";FI$
7Ø5Ø CLS6:PRINT@232,"SAVING□";FI$;
7Ø6Ø MOTORON:FORI=1TO1ØØØ:NEXT
7Ø7Ø OPEN"O",#-1,FI$
7Ø8Ø PRINT#-1,FI$,R,A,NR
7Ø9Ø FORN=1TOA:PRINT#-1,N$(N),
   A(N):NEXT
71ØØ C=1
711Ø IFA$(C,1)="" THEN714Ø
712Ø FORN=1TOA:PRINT#-1,A$(C,N):
   NEXT
713Ø C=C+1:GOTO711Ø
714Ø PRINT#-1,CHR$(13):CLOSE
   #-1:RETURN
8ØØØ CLS:PRINT@7Ø,"ARE YOU SURE (Y/N)?"
8Ø1Ø IN$=INKEY$:IFIN$<>"Y"ANDIN$
   <>"N" THEN8Ø1Ø
8Ø2Ø IFIN$="N" THENRETURN
8Ø3Ø AUDIOON:MOTORON:CLS:PRINT@65,
   "POSITION TAPE THEN PRESS ENTER"
8Ø4Ø IN$=INKEY$:IFIN$<>CHR$(13)
   THEN8Ø4Ø
8Ø5Ø MOTOROFF:PRINT@129,"PLACE
   RECORDER INTO PLAY MODE□□□
   THEN PRESS ENTER"
8Ø6Ø IN$=INKEY$:IFIN$<>CHR$(13)
   THEN8Ø6Ø
8Ø7Ø IFR>Ø THENRUN921Ø
8Ø8Ø INPUT"□NAME OF FILE";FI$
8Ø9Ø CLS7:PRINT@231,"SEARCHING□";
81ØØ OPEN"I",#-1,FI$
811Ø INPUT#-1,FI$
812Ø PRINT@231,"□FOUND□□";
   FI$;"□";
813Ø INPUT#-1,R,A,NR
814Ø DIMA(A),N$(A),A$(R,A)
815Ø FORN=1TOA:INPUT#-1,N$(N),
   A(N):NEXT
816Ø C=1
817Ø IFEOF(-1) THEN82ØØ
818Ø FORN=1TOA:INPUT#-1,A$(C,N)
819Ø NEXT:C=C+1:GOTO817Ø
82ØØ CLOSE#-1:RETURN
85ØØ CLS:PRINT@Ø,"RECORD NUMBER";
   D:FORN=1TOA:PRINT@32*N+32,N$(N);
   "□:";TAB(13);A$(D,N):NEXT:RETURN
9ØØØ CLS4:PRINT@7Ø,"ARE YOU SURE
   (Y/N)?";
9Ø1Ø IN$=INKEY$:IFIN$<>"Y"ANDIN$
   <>"N" THEN9Ø1Ø
9Ø2Ø IFIN$="N" THENRETURN
9Ø3Ø CLS:END
92ØØ GOSUB1Ø4Ø:GOTO922Ø
921Ø GOSUB8Ø8Ø
922Ø B$=CHR$(128):RS$="F□BMADP":
   GOTO3Ø
1ØØØØ PRINT@451,"□□□CHECK
   PRINTER□□□□□□□cONT□";
1ØØ1Ø PRINT@48Ø,"□□□□□□□□□
   □□□□□□□□□□□□□□
   □□□□□□□";
1ØØ2Ø IN$=INKEY$:IFIN$="" THEN1ØØ2Ø
1ØØ3Ø IFIN$<>"C" THENRETURN
1ØØ4Ø FORY=ØTOA+4:FORX=ØTO31:P=
   PEEK(1Ø24+X+Y*32):IFP>95AND
   P<127 THENP=P-64
1ØØ5Ø IFP>ØANDP<27 THENP=P+96
1ØØ6Ø IFP=Ø THENP=32
1ØØ7Ø PRINT#-2,CHR$(P);:NEXT:PRINT
   #-2,CHR$(13);:NEXT
1ØØ8Ø FORN=1TO3:PRINT#-2,CHR$(13):
   NEXT
1ØØ9Ø RETURN
```

Disc users should delete lines 3, 8 and 8ØØ4.

```
1MODE7:M%=Ø:N%=1
2ONERRORGOTO13ØØØ
3*OPT1,1
4HIMEM=PAGE+&3ØØØ
5DIMA(8),N$(8),TRL(8)
6VDU23;82Ø2;1;Ø;Ø;Ø;Ø;
7B%=HIMEM+1
8*OPT3,6
3ØCLS:PRINT'"□□□□□□MAIN MENU"
4ØPRINT''"□□□□□□□1 :—Open a file"
5ØPRINT'"□□□□□□□2 :—Enter a record"
8ØPRINT'"□□□□□□□3 :—View records"
9ØPRINT'"□□□□□□□4 :—Search option"
1ØØPRINT'"□□□□□□□5 :—Save file"
11ØPRINT'"□□□□□□□6 :—Load file"
115PRINT'"□□□□□□□7 :—Quit program"
12ØPRINT''"□□□□□□□Select option
   number"
13ØG=GET-48:IF G<1 OR G>7 THEN 13Ø
14ØIF M%=Ø AND (G>1 AND G<6) THEN
   13Ø
145IF G<>7 THEN 148
146CLS:PRINTTAB(13,12)"Are you sure ?":
   G=GET AND &5F
147IF G=89 THEN END ELSE 3Ø
148ON G□GOTO 15Ø,16Ø,17Ø,18Ø,19Ø,2ØØ
15ØPROCNEWFILE:GOTO 3Ø
16ØPROCENTER:GOTO 3Ø
17ØPROCVIEW:GOTO 3Ø
18ØPROCSEARCH:GOTO 3Ø
19ØPROCSAVE:GOTO 3Ø
2ØØPROCLOAD:GOTO3Ø
```

```
1ØØØDEF PROCNEWFILE
1ØØ5CLS:PRINT'"Are you sure ?"
1Ø1ØG=GET AND &5F:IF G<>89 THEN
  111Ø
1Ø15N%=1:R%=Ø
1Ø2ØCLS:PRINT"Starting new file"
1Ø4ØPRINT''"How many fields do you want (1
  TO 8)"
1Ø5ØA=GET-48:IF A<1 OR A>8 THEN
  1Ø5Ø
1Ø58FOR N=1 TO A
1Ø59PRINT'"Name of field□";N;"□":
  PROCINPUT(1Ø):IF $B%=""
  THENPROCINPUT(1Ø)
1Ø6ØN$(N)=$B%
1Ø65PRINT'"What is the max length of
  field□";N;"□";
1Ø75INPUTA(N):IF A(N)>27 OR A(N)<1
  THEN 1Ø75
1Ø78TRL(N)=R%:R%=A(N)+1+R%
1Ø9ØNEXT:IFR%<11 THEN R%=11
11ØØM%=INT((&7CØØ-HIMEM)/R%):
  PRINT'' "You can use up to□";M%;
  "□records":D%=INKEY(3ØØ)
111ØENDPROC
2ØØØDEF PROCENTER
2ØØ2IF N%=M%+1 THEN CLS:PRINT'"File
  full":G=INKEY(3ØØ):GOTO 213Ø
2ØØ3Q=Ø:CLS:PRINT"You have used□";
  N%-1;"□out of□";M%;"□records"
2ØØ7PRINT'"Press RETURN for MAIN MENU or
  continue□ □to enter files"
2Ø1ØFOR N=1TOA:PRINTTAB(Ø,3+N*2)
  N$(N); "□:"
2Ø2ØPRINTTAB(Ø,23)STRING$(27,"□")TAB
  (Ø,23);:PROCINPUT(A(N))
2Ø21PRINTTAB(13,3+N*2);$B%
2Ø22IF ASC($B%)=-1 AND N=1 THEN
  N=A:Q=1:GOTO 2Ø3Ø
```

Program alterations for the Electron

A couple of alterations have to be made to this program to get it to run on the Electron. Electron owners will have noticed in Line 1 that there is no Mode 7 on their computer. Use Mode 6 instead. So Line 1 should read:

```
1 MODE6:M%=Ø:N%=1
```

Then add these lines:

```
9ØØ4 IF G=67 THEN VDU2
9Ø25 IF G=67 THEN VDU3
```

Line 95ØØ should be replaced with:

```
95ØØ DEF PROCPRINTER
```

And add:

```
95Ø3 PRINT'"Check printer—
  C(ontinue)":G=GET AND &5F:IF
  G<>67 THEN 954Ø
951Ø PROCVDU
```

In Line 11ØØ change &7CØØ to &6ØØØ

```
2Ø25$(B%+N%*R%+TRL(N))=$B%
2Ø3ØNEXT
2Ø35IF Q=1 THEN 213Ø
2Ø37FOR T=1 TO 2ØØØ:NEXT
2Ø4ØN%=N%+1
2Ø7ØIF N%=2 THEN 2ØØØ
2Ø8ØG=N%-1
2Ø9ØX=B%+G*R%:Y=B%+(G-1)*R%:IF
  $X>=$Y□THEN 2ØØØ
21ØØFOR T=1 TO A:$B%=$(X+TRL(T))
  :$(X+TRL(T))=$(Y+TRL(T)):$(Y+
  TRL(T))=$B%:NEXT
211ØG=G-1:IF G=1 THEN 2ØØØ
212ØGOTO2Ø9Ø
213ØENDPROC
3ØØØDEF PROCAMEND
319ØENDPROC
4ØØØDEF PROCDELETE
4Ø9ØENDPROC
5ØØØDEF PROCSEARCH
51ØØENDPROC
6ØØØDEF PROCVIEW
6ØØ2IF N%=1 THEN ENDPROC
6ØØ5D%=Ø:C=1:Q=Ø
6Ø1ØREPEAT
6Ø2ØD%=D%+C
6Ø23IF D%>N%-1 THEN D%=1
6Ø25IF D%<1 THEN D%=N%-1
6Ø3ØPROCVDU
6Ø35PROCKEY
6Ø37IF N%<2 THEN Q=1
6Ø4ØUNTIL Q=1
6Ø5ØENDPROC
7ØØØDEF PROCSAVE
7ØØ2CLS
7ØØ3PRINT"File's name ?□□
  □□□□□□□□□□□";
  TAB(14,Ø);:PROCINPUT(1Ø)
7ØØ4PRINT''
7Ø1ØIF LEN($B%)<1 THEN 7ØØ3
7Ø3ØX=OPENOUT $B%
7Ø35PRINT'"Saving information now"
7Ø4ØPRINT#X,M%,N%,A,R%
7Ø5ØFOR N=1 TO A:PRINT#X,N$(N),
  A(N),TRL(N):NEXT
7Ø55Y=B%+R%:Z=B%+R%*N%
7Ø6ØFOR T=Y□TO Z
7Ø7ØBPUT#X,?T
7Ø8ØNEXT
7Ø9ØCLOSE#X
71ØØENDPROC
8ØØØDEF PROCLOAD
8ØØ2CLS
8ØØ3PRINT"Load which file ?□□□□
  □□□□□□□□□";TAB(18,Ø);:
  PROCINPUT(1Ø)
8ØØ4PRINT''"PRESS PLAY ON RECORDER"
8ØØ7X=OPENIN $B%
8Ø1ØINPUT#X,M%,N%,A,R%
8Ø15B%=HIMEM+1
8Ø2ØFOR N=1 TO A:INPUT#X,N$(N),
  A(N),TRL(N):NEXT
8Ø25Y=B%+R%:Z=B%+R%*N%
8Ø3ØFOR T=Y□TO Z
8Ø4Ø?T=BGET#X
8Ø5ØNEXT
8Ø6ØCLOSE#X
8Ø65VDU13:PRINT''"LOADED CORRECTLY":
  G$=INKEY$(3ØØ)
81ØØENDPROC
9ØØØDEFPROCVDU
9ØØ3CLS
9ØØ5PRINT"Record number□";D%'
9Ø1ØFOR S=1 TO A:PRINTN$(S);"□:"
  TAB(13);$(B%+D%*R%+TRL(S))
9Ø2ØNEXT
9Ø3ØPRINT''"F(orward)□□□□□□B(ack)
  □□□□□□□□M(enu)"'"A(mend)
  □□□□□□□D(elete)□□□□□□
  P(rinter)"
9Ø35ENDPROC
9Ø37DEF PROCKEY
9Ø4ØG=GET AND &5F
9Ø42IF G=7Ø OR G=Ø THEN C=1:GOTO
  91ØØ
9Ø44IF G=77 THEN Q=1:GOTO 91ØØ
9Ø45IF G=8Ø THEN PROCPRINTER:
  GOTO9Ø4Ø
9Ø47IF G=65 THEN PROCAMEND:GOTO
  91ØØ
9Ø5ØIF G=66 THEN C=-1:GOTO 91ØØ
9Ø55IF G=68 THEN PROCDELETE:
  GOTO91ØØ
9Ø6ØGOTO 9Ø4Ø
91ØØENDPROC
95ØØDEF PROCPRINTER:PRINT'"Check
  printer—C(ontinue)":G=GET AND &5F:IF
  G<>67 THEN 954Ø ELSE VDU2:FOR
  Y=Ø TO A*2+3:FOR X=Ø TO 39:VDU1,
  ?(&7CØØ+Y*4Ø+X):NEXT:VDU13:NEXT:
  VDU3
954ØVDU11:PRINTSTRING$ (4Ø,"□"):
  VDU11,11,11
955ØENDPROC
12ØØØDEF PROCINPUT(X)
12Ø1Ø$B%="":FOR T=1 TO X+1
12Ø2ØK=GET
12Ø3ØIF K=127 AND T>1 THEN T=T-1:
  VDU 127:$B%=LEFT$($B%,T-1):
  GOTO12Ø2Ø
12Ø5ØIF K=13 THEN T=X+1:GOTO121ØØ
12Ø6ØIF K<>13 AND T=X+1 THEN
  12Ø2Ø
12Ø7ØIF K<32 OR K>126 THEN12Ø2Ø
12Ø8Ø$B%=$B%+CHR$(K)
12Ø9ØVDU K
121ØØNEXT:ENDPROC
13ØØØIF ERR=17 THEN 3Ø
13Ø1ØIF ERR>215 AND ERR<224 THEN
  PRINT''"FILE HANDLING ERROR":FOR
  T=1 TO 75ØØ:NEXT:GOTO3Ø
13Ø5ØREPORT:PRINT:END
```

RIGHT... UP... LEFT... FIRE!

Arcade-type games rely on the player being able to control events on screen. Here we show you how to control movement, fire missiles and integrate them into a game.

Games like Space Invaders would be awfully dull if the laser base movement or firing couldn't be controlled in some way. Keyboard control of this type is a facet of even the simplest of arcade-type games, and so it is important to grasp the principles if you intend writing your own.

The first step is to get the computer to react when you press a key.

DETECTING A KEYPRESS

In principle, all home computers use the same method of detecting a keypress. In detail, they vary quite widely.

Whenever the Sinclairs, Dragon or Tandy find the function INKEY$ in a program they scan the keyboard to see if a key is being pressed. Here is a short program using INKEY$:

```
2Ø CLS
3Ø IF INKEY$ = "" THEN GOTO 3Ø
4Ø PRINT AT 11,14;"OUCH"
```

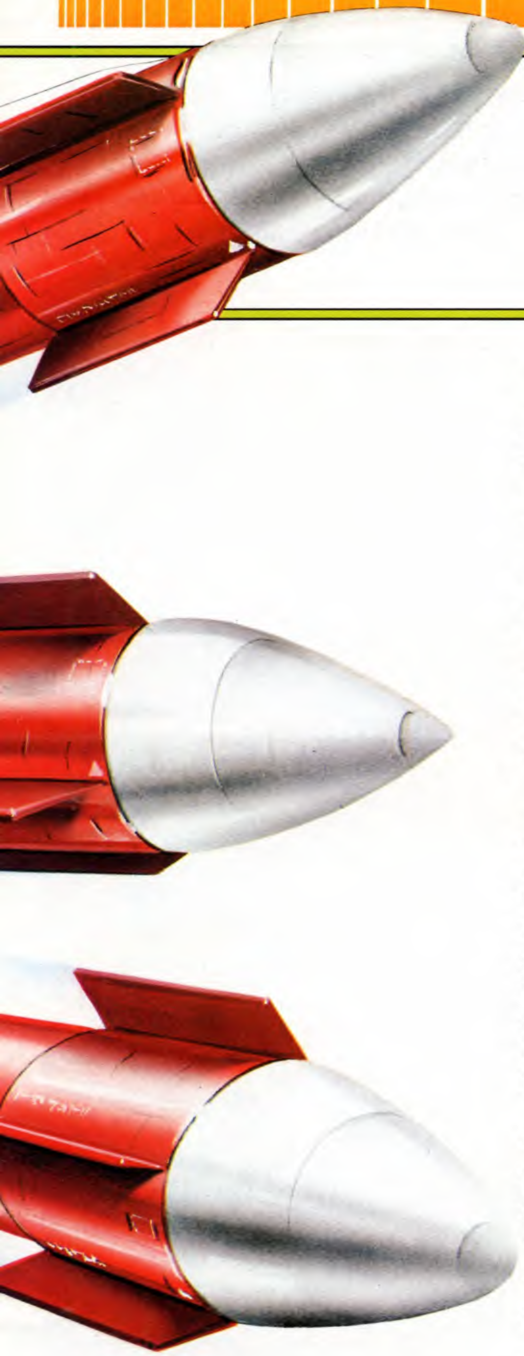
```
3Ø LET A$ = INKEY$: IF A$ = ""
   THEN GOTO 3Ø
4Ø PRINT@ 269, "OUCH"
```

Run the program and then press any key except CAPS SHIFT or SYMBOL SHIFT (on the Sinclairs) or BREAK or SHIFT (on the Dragon or Tandy). The machine will display 'OUCH' in the middle of the screen. The program works like this:

Line 2Ø clears the screen. Line 3Ø makes the computer wait until a key is pressed before continuing with the program. Note that there is no space between the inverted commas. Because of this, Line 3Ø means: 'If INKEY$ = nothing, or if no key is being pressed, check again'. It is important to have the IF ... THEN GOTO 3Ø because otherwise the computer would check only once whether a key was being pressed, and then only for a fraction of a second.

As soon as a key is pressed INKEY$ is made equal to that key. For example, if 3 is pressed INKEY$ = "3". And this is enough to make Line 4Ø display 'OUCH!!' on the screen.

- DETECTING KEYPRESSES
- FIRING MISSILES
- CONTROL A MOVING GRAPHIC
- BUILDING BLOCKS FOR ARCADE-TYPE GAMES
- DESTROYING AN 'ALIEN'
- MISSILE BASES
- USING AN AUTO-REPEAT
- GET$ AND INKEY$
- PLOTTING RANDOM TARGETS



In most games you have to press a certain key to move a tank, spacecraft or whatever. If you change Line 40 you will see how this is done. On the ZX81, use capital D, delete the :STOP and add 45 STOP:

```
40 IF INKEY$="d" THEN PRINT AT
   11,9;"MMM, THAT'S NICE"
45 STOP
50 PRINT AT 11,14;"OUCH!!"
```

```
40 IF A$="D" THEN PRINT@
   264,"MMM, THAT'S NICE" ELSE PRINT@
   269, "OUCH!!"
```

Line 40 checks to see if the D key has been pressed; in other words is INKEY$ equal to D or d? If it isn't, the Spectrum, which has no ELSE statement, will ignore Line 40 and go on to Line 50. (To find out why the STOP is necessary, try omitting it!)

Two more things are important in this program. The first is that the "D" or "d" must be in quotation marks, or the computer might mistake it for a variable. The second, on the Spectrum only, is that you normally need a lower case "d", not a capital. Otherwise you would have to press CAPS SHIFT and D to make it work.

On the Acorn computers you can use either GET$ or INKEY$ to see if a key is being pressed. The main difference between them is that GET$ will halt the program and then wait, for ever if necessary, until you press a key, whereas with INKEY$ you can detect a key at any time while a program is actually RUNning—an essential feature in many games programs.

Here is the short program using GET$ to surprise the unwary:

```
20 CLS
30 key$=GET$
40 PRINT TAB(17,13) "OUCH!!"
```

Type this in and RUN it, then press any of the character keys.

It works like this. Line 30 waits for a key to be pressed and puts the character into key$—it remembers it, but doesn't PRINT it on the screen. The program then goes to Line 40 which PRINTs the word 'OUCH!!' in the centre of the screen.

This program will react to any key, but usually in a game you want different keys to do different things. To detect a specific keypress, all you do is change Line 40 to:

```
40 IF key$="D" THEN PRINT TAB(11,12)
   "MMM, THAT'S NICE" ELSE PRINT
   TAB(17,13) "OUCH!!"
```

Line 40 checks to see if the character it has stored in key$ is equal to "D". If it is, it PRINTs 'MMM, THAT'S NICE', but if any other key is pressed it PRINTs 'OUCH!!'

The GET statement can be used by the Commodore machines to detect each keyboard press. This short program shows how GET is typically used to do this:

```
20 PRINT "♥"
30 GET A$ : IF A$="" THEN GOTO 30
50 PRINT TAB (17) "OUCH!":END
```

RUN and reRUN the program to see that all keypresses complete the program by displaying 'OUCH' on the screen—except RUN/STOP, SHIFT and the 'Commodore' key.

Line 20 of the program clears the screen. Line 30 with the GET statement is set to accept a keypress. The quotation marks with nothing between them mean: 'if no key is pressed'. In this case, the line doubles back on itself and waits for a key to be pressed before continuing.

Line 50 then displays 'OUCH!' on the screen, and the program ends.

In a typical game you may be asked to press a certain key to indicate choice from a menu selection, or to move a frog, laser base or whatever. By adding another line to the program you can specify which key has to be pressed for the program to continue in a particular way:

```
40 IF A$="D" THEN PRINT TAB(12) "MMM,
   THAT'S NICE":END
```

Line 40 is reached after any keypress. It checks if the key responsible was the D key, completing the PRINT instruction if it was. Otherwise the program skips to the next line. (Why is the END necessary? Omitting it will show you!)

FIRING A MISSILE

Now you can see how from detecting a specific keypress using IF INKEY$="key" it is a short step to firing a missile or moving a missile base. This program fires a missile from a base at the bottom of the screen when the F key is pressed:

```
20 CLS
30 PRINT@ 397,"###"
```

```
40 LET K$=INKEY$:IF K$="" THEN
   GOTO 40
50 IF K$="F" THEN LET M=334 ELSE
   GOTO 40
60 PRINT@ M,"↑"
70 PRINT@ M+32,"□"
80 LET M=M−32
90 IF M>0 THEN GOTO 60
100 GOTO 20
```

Line 30 PRINTs a missile base made up of three hash marks (#) on the lower part of the screen, starting at position 397.

In Line 40 the LET K$=INKEY$ is very important because you want to check the keypress several times during the program. In fact this program wouldn't work without it! The computer only remembers the value of INKEY$ for a split second—if you're not quick enough checking INKEY$ the computer forgets that a key has been pressed. You can make the computer remember the keypress, though, by calling the keypress K$, and checking K$ later in the program.

Line 50 checks to see if F was pressed, and if F was not pressed, continues to scan the keyboard by going back to Line 40. M is the position of the missile *after* it has been fired.

Line 60 displays the missile and Line 70 blanks out the previous position of the missile.

Line 80 subtracts 32 from the missile's position so that the missile moves up one line every time it is PRINTed. (The reason for subtracting 32 is that there are 32 columns or character spaces on each line of the computer's screen.)

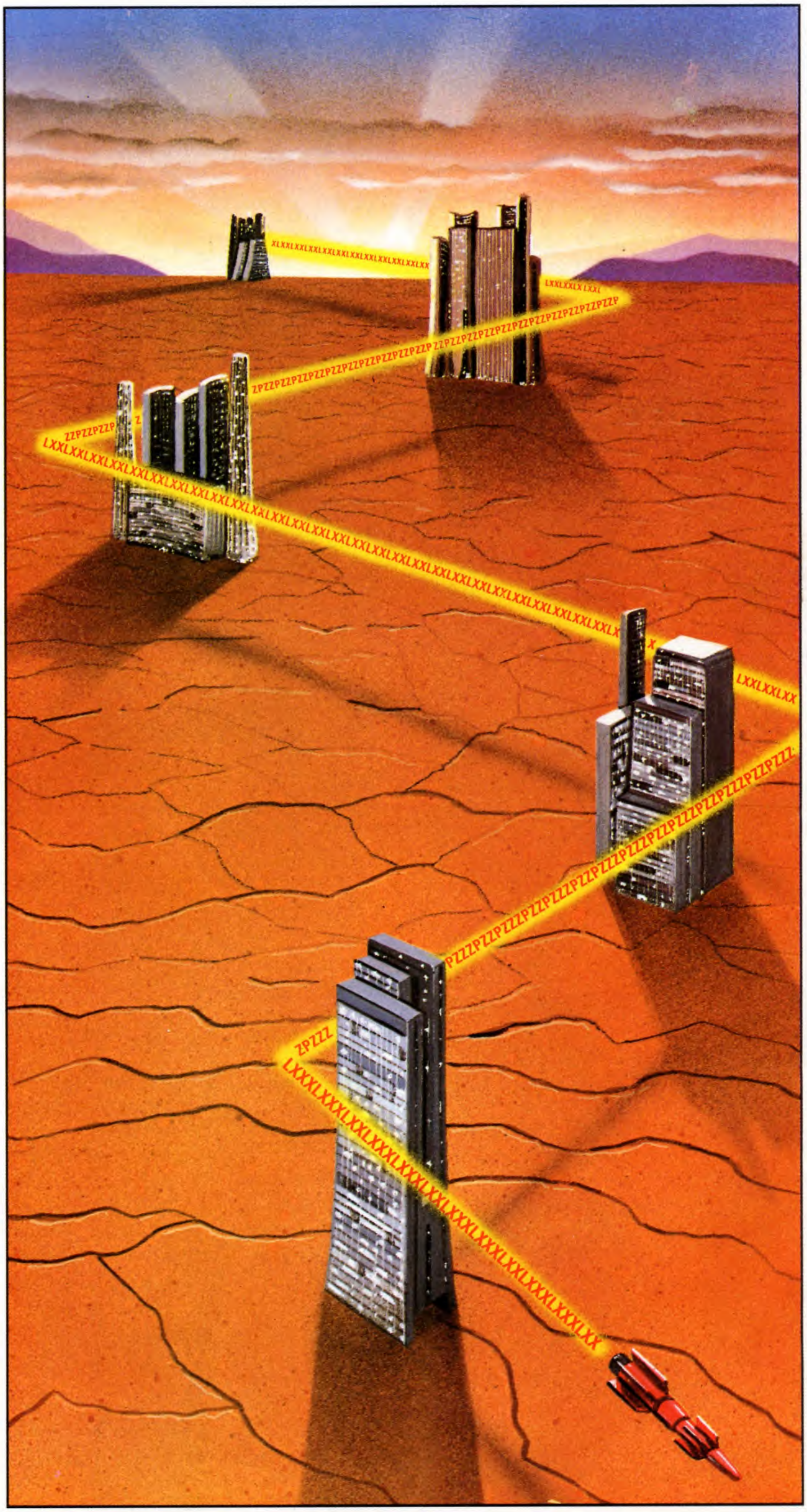
Line 90 stops the computer trying to PRINT the missile at a position not on the screen, which would give an error message. The screen starts at M=0, and the computer cannot PRINT at a position which is less than zero. When M=0, the program reRUNS.

This program fires a missile when f is pressed. On the ZX81, use capitals throughout, and an asterisk instead of the arrow in Line 60.

```
20 CLS
30 PRINT AT 21,14;"▄█▄"
40 IF INKEY$="" THEN GOTO 40
50 IF INKEY$<>"f" THEN GOTO 40
55 LET y=20
60 PRINT AT y,15;"↑"
70 LET y=y−1
75 PAUSE 1
80 PRINT AT y+1,15;"□"
90 IF y>0 THEN GOTO 60
```

Line 30 uses the low resolution ROM graphics to display the missile base.

Line 40, as before, makes the Spectrum



wait until a key is pressed.

Line 50 checks if the f-key has been pressed. If it was, the start position of the missile is set. This is on the twentieth screen line, one above the missile base.

Line 40 makes the computer scan the keyboard again if any key other than f has been pressed.

Line 60 displays the missile on the screen. The Spectrum character is an up arrow and is obtained by pressing SYMBOL SHIFT and H.

Line 80 blanks out the previous position of the missile.

Line 70 subtracts 1 from y, which is the line coordinate of the missile position, thus moving the missile up one screen line.

Line 90 stops the missile going off the screen (when it reaches the top line, the y coordinate is 0).

This program fires a missile when F is pressed:

```
20 CLS
30 PRINT TAB(19,20)"#∧#"
40 LET K$=GET$
50 IF K$="F" THEN M=19 ELSE GOTO 40
60 PRINT TAB(20,M)"∧"
70 PRINT TAB(20,M+1)"□"
80 LET M=M-1
90 IF M>0 THEN GOTO 60
100 GOTO 20
```

Line 30 displays the shape of the missile base on the screen.

Line 40 waits for a keypress. Line 50 checks if F was pressed, and if it was, sets the start position of the missile. If F hasn't been pressed the Acorn keeps waiting.

Line 60 displays the missile, and Line 70 blanks out the previous missile position.

Line 80 subtracts 1 from the missile position so that the missile appears to move up the screen.

Line 90 loops the program until the missile reaches the top of the screen, and then Line 100 reRUNs the program.

This program fires a missile when F is pressed. On the Vic 20, change 18 to 8 and omit two ■s in Line 30. In Line 50, use 471, not 939. In Line 70, use 7680 instead of 1024, 38400 for 55296, and N,0, not N,1. Use 7680, not 1024, in Line 80, and 22 instead of 40 in Lines 80 and 90.

```
20 PRINT "□"
30 PRINT TAB(18) "■■■■■■■■
   ■■■■■■■■■■■
   ■■▄↑▄"
40 GET K$ : IF K$="" THEN GOTO 40
50 IF K$="F" THEN N=939 : GOTO 70
60 GOTO 40
70 POKE 1024+N,30 :
   POKE 55296+N,1
80 POKE 1024+N+40,32
90 N=N-40
100 IF N>0 THEN GOTO 70
110 IF N<0 THEN GOTO 20
```

Low resolution ROM graphics are used to form a missile base at the bottom centre of the screen (Line 30).

Lines 40, 50 and 60 effectively cause the program to wait until the F key is pressed, triggering the missile when it has been, and giving a value to N which is the position of the missile *after* it has been fired.

Line 70 PRINTs the missile, Line 90 moves it up a line at a time, and Line 80 blanks out the previous position to create the illusion of movement.

Lines 100 and 110 check that the missile is within the screen area, restarting when it no longer is so.

MOVING AROUND THE SCREEN

As it stands, the missile base program is rather boring, but adding side to side movement to the base improves matters a little. Let's look at how the base itself can be moved:

```
20 CLS
30 LET P=397
40 PRINT@ P,CHR$(143)+CHR$(140)+
   CHR$(128)+CHR$(140)+
   CHR$(143)
50 LET K$=INKEY$:IF K$="" THEN
   GOTO 50
60 IF K$="L" THEN LET P=P-1:
   GOTO 90
70 IF K$="R" THEN LET P=P+1:GOTO 90
80 GOTO 50
90 IF P<384 OR P>411 THEN GOTO 50
100 GOTO 40
```

Line 30 sets the start position of the missile base, and Line 40 displays the missile at that position.

Line 50 scans the keyboard as before, making the computer wait until a key is pressed before continuing.

Line 60 checks if L has been pressed, and if it has, moves the base one space to the left by subtracting 1 from the number that sets the position of the base.

Line 70 checks if R has been pressed, and changes the base position by adding 1.

Line 90 checks if the program is telling the computer to PRINT the base off the screen—if it is, the program goes back to Line 50.

Can I choose any keys to operate a game with?

Yes—all you have to do is to change the letters in your INKEY$ or GET$ lines. But beware: what looks logical sometimes works very badly in practice. For example L, R, U and D for left, right, up and down are impossibly awkward. If your machine has a space bar, this is often handy as a fifth key, especially for 'firing'.

This program moves a missile base around the screen. On the ZX81, type this entirely in capital letters:

```
30 CLS
40 LET x=15
50 LET y=13
60 PRINT AT y,x;"▄█▄"
70 IF INKEY$="" THEN GOTO 70
80 LET lx=x: LET ly=y
90 PRINT AT ly,lx;"□□□"
100 IF INKEY$="q" THEN STOP
110 IF INKEY$="p" THEN LET y=y-1
120 IF INKEY$="l" THEN LET y=y+1
130 IF INKEY$="z" THEN LET x=x-1
140 IF INKEY$="x" THEN LET x=x+1
150 IF x<1 OR x>29 THEN LET x=lx
160 IF y<1 OR y>20 THEN LET y=ly
170 GOTO 60
```

Lines 40 and 50 set the start position of the missile base, 13 lines from the top and 15 spaces from the left. Line 60 displays the base on the screen.

Line 70 makes the Spectrum wait until a key is pressed.

Lines 80 and 90 are perhaps the most difficult to understand. But what they do, in effect, is to make a row of three spaces follow the missile base around the screen. Because Line 60 always comes before Line 90 in the loop, the base in its new position is always PRINTed before the old position is cleared.

Line 100 terminates the program if the letter q is pressed (q for 'quit').

Line 110 checks if p has been pressed, and if it has, subtracts 1 from the y value. The effect this has is to move the base one 'space' up the screen.

Lines 120 to 140 operate similarly, Line 120 moving the base down if l is pressed, Line

130 moving the base left if z is pressed, and Line 140 moving the base right if the x-key is pressed.

Line 150 prevents the base being PRINTed off the edge of the screen.

Line 160 checks that the program is not trying to PRINT the base off the top or bottom of the screen. If this is the case making y = ly prevents this happening.

Line 170 completes the loop, causing the keyboard to be scanned and the process to start again.

This program will move the missile base around the screen:

```
20 VDU 23;8202;0;0;0;
30 CLS
40 X=19
50 Y=13
60 PRINT TAB(X,Y)"#^#"
70 KEY$=GET$
80 LX=X:LY=Y
90 PRINT TAB(LX,LY)"□□□"
100 IF KEY$="Q" THEN END
110 IF KEY$="P" THEN Y=Y-1
120 IF KEY$="L" THEN Y=Y+1
130 IF KEY$="Z" THEN X=X-1
140 IF KEY$="X" THEN X=X+1
150 IF X<1 OR X>36 THEN X=LX
160 IF Y<1 OR Y>23 THEN Y=LY
170 GOTO 60
```

When the program is RUN you will see the base positioned in the middle of the screen. Use Z and X to move it left and right, and P and L to move it up and down. Type Q to quit the program.

The program works like this: Line 20 turns off the flashing cursor. Lines 40 and 50 set the start position of the base, and Line 60 displays it. Line 70 waits for a keypress.

Lines 80 and 90 work in the same way as in the program for the Spectrum and ZX81 (see previous page).

Lines 100 to 140 check which key has been pressed and act accordingly, either ending the program or moving the base. Lines 150 and 160 stop the base moving off the screen.

Line 170 returns the program to Line 60 which displays the base in its new position.

On the Vic, use 8, not 18, in Line 20. Omit two ■s in Line 40, and use 18s, not 36s in Line 90.

```
20 P=18
30 PRINT "♥"
40 PRINT TAB(P) "■■■■■■■
   ■■■■■■■■■■■■
   ■■▄↑▄"
50 GET K$ : IF K$="" THEN GOTO 50
60 IF K$="L" THEN P=P-1 : GOTO 90
70 IF K$="R" THEN P=P+1 : GOTO 90
80 GOTO 50
90 IF P>36 THEN P=36
100 IF P<1 THEN P=1
110 GOTO 30
```

The P in Line 20 sets the start position of the missile base, and Line 40 displays the base at that position. The keypress routine appears in Lines 50, 60 and 70, checking to see if either L or R keys have been pressed, subtracting 1 from P to move the base left, adding 1 to move it right. The GOTO in Line 80 returns to Line 50 if any other key is pressed.

Lines 90 and 100 check that the value of P falls within the screen area, repeating the GET loop if it doesn't. Finally, Line 110 returns the program to the missile base PRINTing line.

CREATING A GAME

You now have some building blocks from which games can be constructed. The game below shows one way of using them. On the Tandy, use 247, not 223, in Lines 100 and 110.

```
20 CLS
30 FOR N=1 TO 100:NEXT N
40 LET PO=430
50 LET B$=CHR$(143)+
   CHR$(140)+CHR$(128)+
   CHR$(140)+CHR$(143)
60 LET A=RND(30)+64
70 PRINT@ A,"*"
80 LET LP=PO
90 PRINT@ PO,B$
100 IF PEEK(340)=223 THEN LET
   PO=PO-1
110 IF PEEK(338)=223 THEN LET
   PO=PO+1
120 IF PO<415 OR PO>444 THEN LET
   PO=LP
130 LET K$=INKEY$
140 IF K$="F" THEN LET M=PO-30 ELSE
   GOTO 80
150 PRINT@ M,"↑";
160 PRINT@ M+32,"□";
170 LET M=M-32
180 IF M=A THEN GOTO 20
190 IF M>0 THEN GOTO 150 ELSE PRINT@
   M+32,"□";
200 GOTO 80
```

On the ZX81, type this entirely in capitals and use an asterisk not an arrow, delete LET y=21 and add 45 LET Y=21:

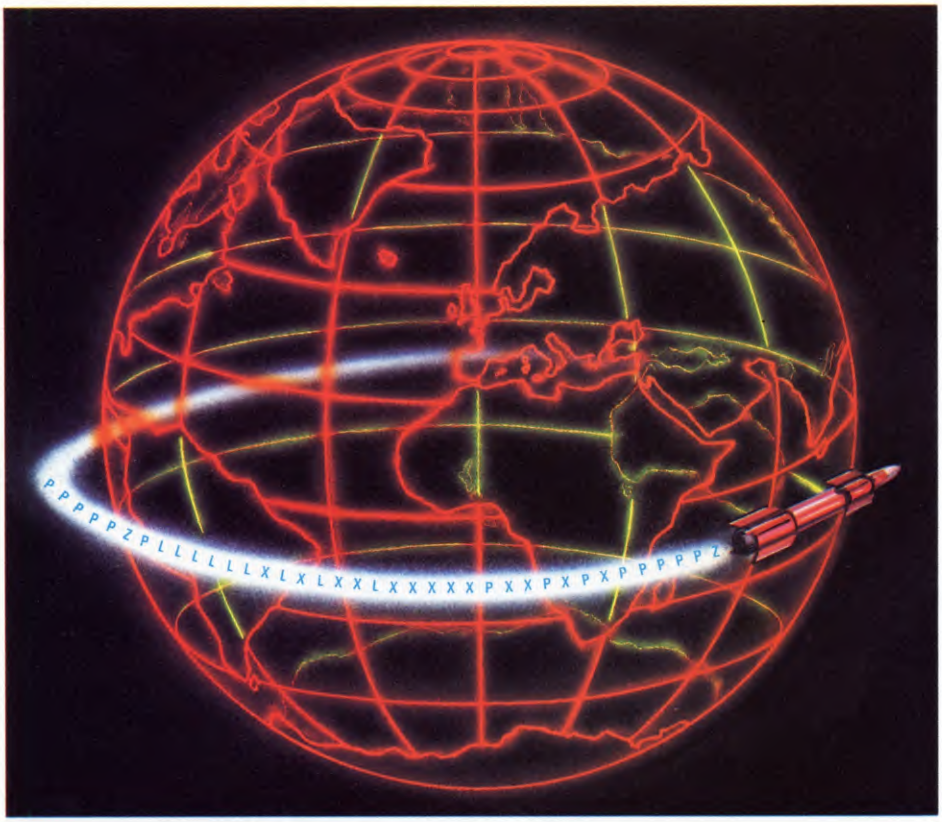
```
20 CLS
30 PAUSE 25
40 LET x=15
45 LET y=21
50 LET B$="□▄■▄□"
60 LET a=INT (RND*28)+2
70 PRINT AT 2,a;"*"
80 LET xx=x
90 PRINT AT y,x;B$
100 IF INKEY$="z" THEN LET x=x-1
110 IF INKEY$="x" THEN LET x=x+1
120 IF x<0 OR x>27 THEN LET x=xx
140 IF INKEY$<>"f" THEN GOTO 80
145 LET m=y-1
150 PRINT AT m,x+2;"↑"
160 PRINT AT m+1,x+2;"□"
170 LET m=m-1
180 IF m=2 AND x+2=a THEN GOTO 20
190 IF m<>1 THEN GOTO 150
195 PRINT AT m+1,x+2;"□"
200 GOTO 80
```

```
15 VDU 23;8202;0;0;0;
20 CLS
30 FOR N=1 TO 200:NEXT N
40 LET X=19: LET Y=20
50 LET B$="□#^#□"
60 LET AX=RND(36)+1
70 PRINT TAB(AX,3) "*"
80 LET LX=X
90 PRINT TAB(X,Y);B$
95 LET K$=GET$
100 IF K$="Z" THEN LET X=X-1
110 IF K$="X" THEN LET X=X+1
120 IF X<0 OR X>35 THEN LET X=LX
140 IF K$="F" THEN LET M=19 ELSE GOTO 80
150 PRINT TAB(LX+2,M) "^"
160 PRINT TAB(LX+2,M+1) "□"
170 LET M=M-1
180 IF M=3 AND LX+2=AX
   THEN GOTO 20
```

Q+A

Why does my program crash when a graphic reaches the edge of the screen?

Probably, a number in one of the lines which control the graphic's position on the screen is too big. So re-check these lines if you get any of the following error reports: screen scrolling (Acorn), illegal quantity (Commodore), F/C error (Dragon), integer out of range (Spectrum).



```
190 IF M>Ø THEN GOTO 15Ø ELSE PRINT
    TAB(LX+2,M+1) "□"
2ØØ GOTO 8Ø
```

On the Vic, replace Line 15 with 15 POKE 36879, 29. Change the 16 in Line 4Ø to 8, and the 34s in Lines 6Ø and 1Ø5 to 16. Omit two ■s from Line 5Ø.

```
15 POKE 5328Ø,5:POKE 53281,1
2Ø PRINT "■"
3Ø CLR
4Ø LET P=16:LET A=1
5Ø LET D$="■■■■■■■
   ■■■■■■■■■■■
   ■■■■"
6Ø LET A=INT(RND(1)*34)+3
7Ø PRINT "■"TAB(A)"♣"
8Ø PRINT "■"TAB(P)D$"□▄✝▄□"
9Ø GET K$:IF K$="" THEN 9Ø
95 IF K$="Z" THEN P=P-1
1ØØ IF K$="X" THEN P=P+1
1Ø5 IF P>34 THEN P=34
11Ø IF P<1 THEN P=1
115 IF K$="F" THEN P1=P:D=22:
    GOTO 13Ø
12Ø GOTO 8Ø
13Ø PRINT "■" TAB(P1);
14Ø PRINT LEFT$(D$,D)"■■✝■■
    □":D=D-1
15Ø PRINT "■" TAB(P1);
16Ø PRINT LEFT$(D$,D)"■■□■■
    □":D=D-1
17Ø IF D>Ø THEN 13Ø
18Ø IF P1=A-2 THEN 2Ø
2ØØ GOTO 8Ø
```

When you RUN this you will see a star near the top of the screen. The Z and X keys move the missile base left and right, and the F key fires a missile to destroy the star.

Think of the program as having three sections: up to Line 7Ø, Lines 8Ø to 12Ø, and Lines 13Ø/14Ø to 2ØØ.

Lines 13Ø/14Ø to 2ØØ are similar to the earlier missile firing program for your machine. The variables have been changed, along with the GOTOs, but the only addition is Line 18Ø. This simply looks to see if the missile and the star are in the same place. If they are, the program restarts.

The middle section, Lines 8Ø to 12Ø, is a shortened version of the 'moving around the screen' program for your machine. The Dragon and Tandy lines are borrowed from both 'moving missile base' and from 'better movement' (see below). The PEEKS in the program check if Z or X have been pressed and alter PO as appropriate.

The first section of the program, up to Line 7Ø, performs a variety of functions. In the Acorn program Line 15 turns off the flashing cursor. In all programs, Line 3Ø introduces a short pause before the program continues. This is important when Line 18Ø completes the loop at the end of the program. Lines 4Ø and 5Ø set the start position of the missile base and define its shape (on the Commodore, the shape is set by Line 8Ø). Lines 6Ø and 7Ø choose a place for the star and display it.

BETTER MOVEMENT

Having to press the 'left' or 'right' key each time you want a graphic to move, as you do on the Dragon, Tandy and Commodore, is rather laborious. So it is usual to build in an auto-repeat facility.

On the Commodore this is done by using a single POKE, so add this line to your program:

```
1Ø POKE 65Ø, 128
```

In fact, you can use any value of 128 or higher. To cancel auto-repeat, POKE the same location, 65Ø, with the value 127.

Continuous movement is difficult using INKEY$, and it is not possible to write smooth games this way. But, there is a way round the problem which is illustrated in the following program. On the Tandy, use 247 in place of 223 in Lines 7Ø and 8Ø; 251, not 239 in Line 9Ø; 253, not 247, in Line 1ØØ.

```
2Ø CLS
3Ø LET BL$=CHR$(128)
4Ø LET PO=238
5Ø PRINT@ PO,BL$
6Ø LET LP=PO
7Ø IF PEEK(34Ø)=223 THEN LET
   PO=PO-1:GOTO 12Ø
8Ø IF PEEK(338)=223 THEN LET
   PO=PO+1:GOTO 14Ø
9Ø IF PEEK(338)=239 THEN LET
   PO=PO-32:GOTO 15Ø
1ØØ IF PEEK(342)=247 THEN LET
    PO=PO+32:GOTO 15Ø
11Ø GOTO 7Ø
12Ø IF (LP AND 31)=Ø THEN LET PO=LP
13Ø GOTO 15Ø
14Ø IF (PO AND 31)=Ø THEN LET PO=LP
15Ø IF PO>51Ø OR PO<Ø THEN LET
    PO=LP:GOTO 7Ø
16Ø PRINT@ LP,"□";
17Ø PRINT@ PO,BL$;
18Ø GOTO 6Ø
```

When you RUN the program you will see a block positioned in the centre of the screen. The program will move the block from side to son side and up and down.

THE PROGRAMMER'S ROAD SIGNS

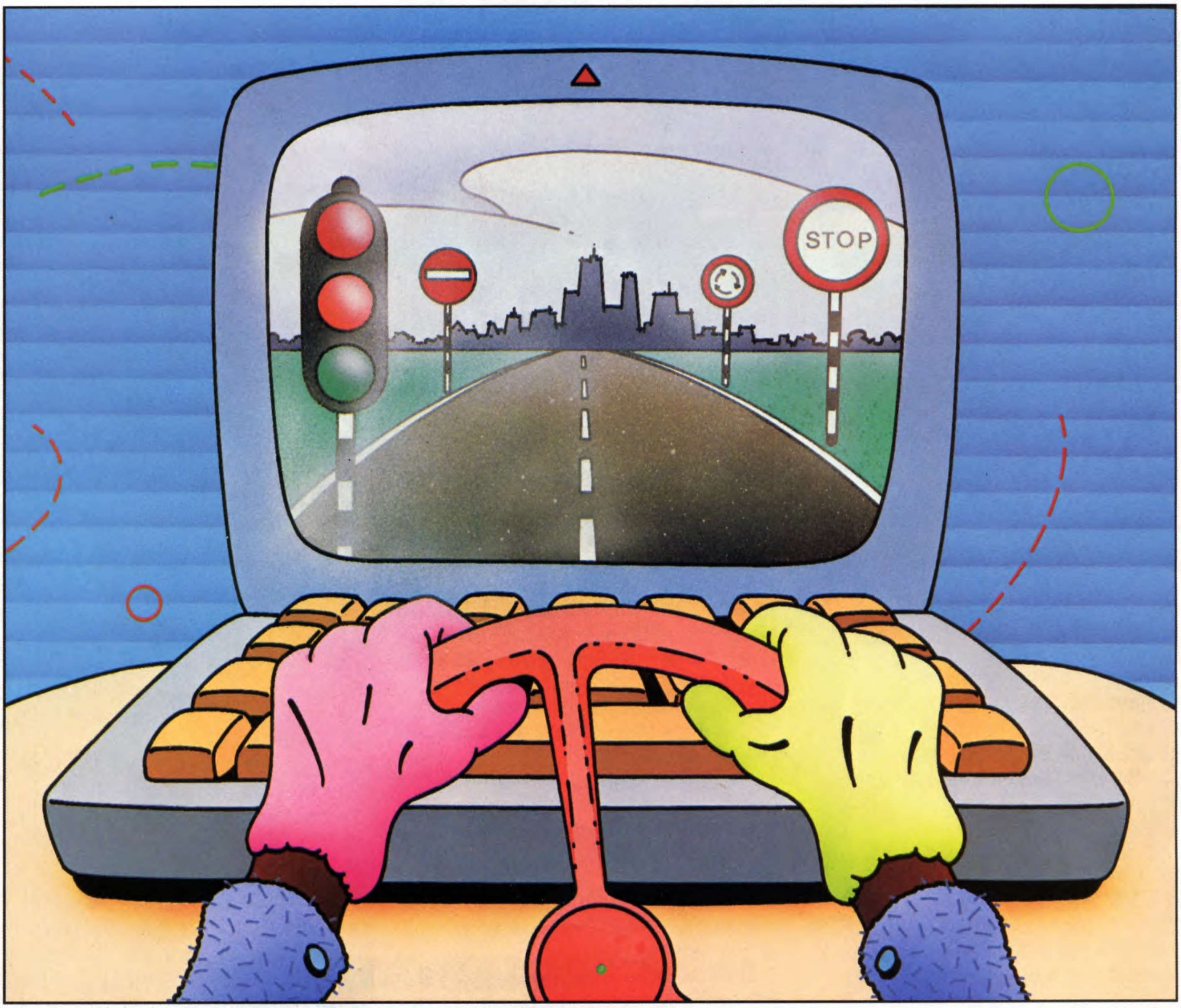
A close look at the two keywords—GOTO and GOSUB—which can carry out much of the work of steering the course of a program by creating forwards and backwards jumps

One of the most fundamental statements in BASIC programming is GOTO. Its function is to alter the pattern of a program so that instead of simply executing the program lines in numerical order, the computer jumps to the line specified in the GOTO statement.

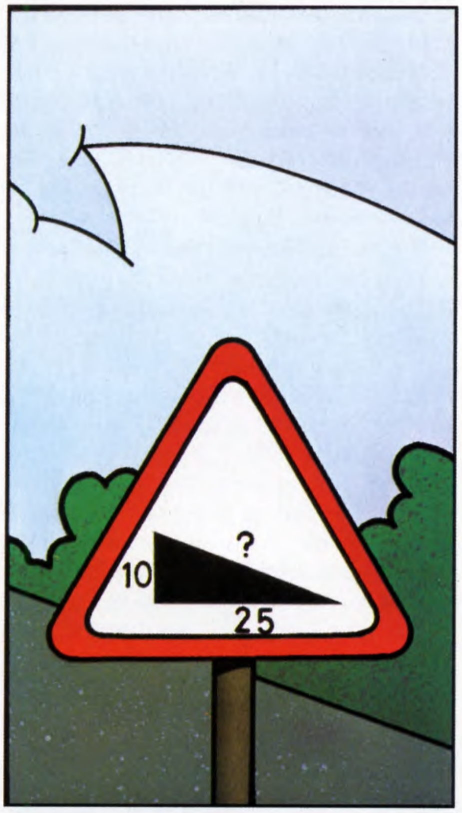
Although it sometimes appears on the screen as two words, GOTO is normally keyed in as one. On the Spectrum, you press the GOTO key. On other machines you type in GOTO as one word with no space between the GO and the TO.

The word GOTO is always followed by the number of the line you want to jump to. Sometimes, although not on the Commodore machines, this number can be represented by a letter—A, say—which assumes a numerical value when the program is RUN.

GOTO statements allow you to jump backwards, creating a loop. They are rather like those formed by FOR . . . NEXT loops (see page 16) but there is no limit to the number of times you go round it. This simple program, for example, calculates the length of the hypotenuse of a right-angled triangle. (Note how A and B values are squared here—it's quicker than using A ↑ 2 and B ↑ 2.)



- **GOTO** AND **GOSUB** IN PRACTICE—PROGRAMS FOR CALCULATING, NAME GUESSING AND DICE THROWING
- WHEN AND HOW TO CREATE FORWARDS AND BACKWARDS PROGRAM JUMPS
- USE IN COMPLEX BRANCHES
- SPEEDING UP SUBROUTINES
- USING **PROCEDURES**
- AVOIDING THE BAD PROGRAMMING PITFALLS OF **GOTO**




```
10 PRINT "Length of sides A,B, in centimetres"
15 INPUT A,B
20 LET C=SQR (A*A+B*B)
30 PRINT "The length of side C is□";
   C;"□centimetres."
40 GOTO 10
```

```
10 PRINT ' '"Length of sides A,B, in cms"
15 INPUT a,b
20 LET c=SQR (a*a+b*b)
30 PRINT "The length of side C is□"'c;
   "□centimetres"
40 GOTO 10
```

```
10 PRINT "LENGTH OF SIDES A,B IN CMS"
15 INPUT A,B
20 LET C=SQR(A*A+B*B)
30 PRINT "THE LENGTH OF SIDE C
   IS";C;"CM"
40 GOTO 10
```

In Line 2Ø, SQR (A*A+B*B) means 'the square root of A squared plus B squared, or $\sqrt{A^2+B^2}$—Pythagoras' formula for calculating the length of the hypotenuse of a right-angled triangle. Line 3Ø PRINTs out its value.

This program will RUN over and over again because, each time it gets to Line 4Ø, the GOTO statement sends it back to Line 1Ø again and the program RUNs again. The only way out of this cycle is to hit ESCAPE, BREAK or RUN/STOP, (or on the Spectrum, enter STOP to an INPUT) or to switch the computer off and start again.

FORWARD JUMPS

A GOTO statement can also be used to skip forward over a block of program, as in this coin-tossing program:

```
5 delay=INKEY(200):CLS
10 PRINT''"I'M TOSSING THE COIN...";
20 FOR J=1 TO 3
30 PRINT ".";
40 delay=INKEY(100)
50 NEXT
70 IF RND(1)<0.5 THEN GOTO 100
80 PRINT' "AND IT'S TAILS!"
90 GOTO 5
100 PRINT' "AND IT'S HEADS!"
110 GOTO 5
```

```
5 PAUSE 50: CLS
10 PRINT "I'M TOSSING
   THE COIN...";
20 FOR j=1 TO 3
30 PAUSE 25
40 PRINT ".";
50 NEXT j
60 PRINT
70 IF RND<.5 THEN GOTO 100
80 PRINT "AND IT'S TAILS!"
90 GOTO 5
100 PRINT "AND IT'S HEADS!"
110 GOTO 5
```

```
5 FOR F=1 TO 500:NEXT F:CLS
10 PRINT "I'M TOSSING THE COIN...."
20 FOR J=1 TO 3
30 FOR F=1 TO 250:NEXT F
40 PRINT ".";
50 NEXT J
60 PRINT
70 IF RND(0)<.5 THEN GOTO 100
80 PRINT "AND IT'S TAILS!"
90 GOTO 5
```

```
100 PRINT "AND IT'S HEADS!"
110 GOTO 5
```

```
5 FOR D=1 TO 1000 : NEXT : PRINT "♥"
10 PRINT "I'M TOSSING THE COIN...";
20 FOR J=1 TO 3
30 FOR D=1 TO 250 : NEXT D
40 PRINT ".";
50 NEXT J
60 PRINT
70 IF RND(0)<.5 THEN GOTO 100
80 PRINT "AND IT'S TAILS!"
90 GOTO 5
100 PRINT "AND IT'S HEADS!"
110 GOTO 5
```

The RND in Line 70 gives you a random number, selected by the computer, between 0 and 1. Here it forms part of the condition on the GOTO statement. If the random number selected by the computer is less than a half, the computer jumps forward to Line 100. If not, the computer will naturally execute the next program line, Line 80—any other instructions on 70 would have been disregarded.

This condition means that Line 70 forms a branch in the program. Either the computer reads Lines 70, 80, 90 to display 'AND IT'S TAILS!' or it reads Lines 70, 100, 110 to display 'AND IT'S HEADS!'. It does this quite randomly, by the flick of an electronic coin.

Lines 90 and 110 also contain GOTO statements. Whichever branch the computer has taken, these send it back to the beginning of the program, Line 5, to start all over again.

Again, you will note that this program has no end. The only way out of it is to hit ESCAPE, BREAK or RUN/STOP, or switch off.

MORE-COMPLEX BRANCHES

On Acorn computers and the Spectrum, the GOTO statement need not be accompanied by a natural number. A variable will do—GOTO A, for example, or GOTO (100+INT(RND*6)). This means that the GOTO statement can give a complex branch in your program, as in:

```
100 PRINT "Hello, what's your name?"
110 INPUT A$
120 GOTO (120+RND(4)*10)
130 PRINT "That's a nice name,□";A$:
    GOTO 170
140 PRINT "That's a funny name,□";A$:
    GOTO 170
150 PRINT "Pleased to meet you,□";A$:
    GOTO 170
160 PRINT "Hello□";A$;"□I'm your
    computer."
170 END
```

```
100 PRINT "Hello, what's your name?"
110 INPUT a$
120 GOTO (130+INT (RND*4)*10)
130 PRINT "That's a nice name,□";a$:
    GOTO 170
140 PRINT "That's a funny name,□";a$:
    GOTO 170
150 PRINT "Pleased to meet you,□";a$:
    GOTO 170
160 PRINT "Hello□";a$;",□I'm your
    computer."
170 STOP
```

The GOTO statement in Line 120 gives a random jump forward to any of the next four lines, which are then executed. This is often useful in games where, for example, you may want a character to follow an unpredictable course.

The GOTO 170 makes the computer skip forward, missing out the intervening lines. Note that Line 160 doesn't need a GOTO 170 on the end of it, as the computer goes to Line 170 anyway, once Line 160 has been executed.

This program RUNs only once, because all the GOTOs instruct the computer to skip forward, so no closed loops are formed. When it gets to Line 170 it stops.

ON ... GOTO

On the Commodore, Dragon and Acorns there is a near-equivalent—the ON ... GOTO statement. This takes the form:

ON A GOTO 100, 200, 300, 400

When A = 1, the computer will go to the first destination, Line 100. When A = 2, it will go to the second, Line 200, and so on. Again this allows a complex branch in your program and turns the above program into:

```
100 PRINT "HELLO, WHAT'S YOUR NAME?"
110 INPUT A$
120 ON RND(4) GOTO 130,140,150,160
130 PRINT "THAT'S A NICE NAME,□";
    A$:GOTO 170
140 PRINT "THAT'S A FUNNY NAME,□";
    A$:GOTO 170
150 PRINT "PLEASED TO MEET YOU,□";
    A$:GOTO 170
160 PRINT "HELLO,□";A$;",□I'M YOUR
    COMPUTER"
170 END
```

```
100 PRINT "HELLO, WHAT'S YOUR NAME?"
110 INPUT A$
120 ONINT(RND(1)*5)GOTO 130,140,150,160
130 PRINT "THAT'S A NICE NAME,□"
    A$:GOTO 170
140 PRINT "THAT'S A FUNNY NAME,□"
    A$:GOTO 170
150 PRINT "PLEASED TO MEET YOU,□"
    A$:GOTO 170
160 PRINT "HELLO□" A$ "□I'M YOUR
    COMPUTER"
170 END
```

GOOD AND BAD PROGRAMMING

The over-use of GOTO is considered bad programming style. One reason is that even in simple programs a GOTO statement that sends you back to a preceding line can create an endless loop which can only be escaped from by use of the ESCAPE, BREAK or RUN/STOP key—or by switching off!

But the main reason is that by allowing you to jump backwards and forwards to any point in the program on a whim, GOTO tends to break up the program's logical structure. This may not seem very important when you are dealing with five- or ten-line programs, but it can be vital when coping with 100- or 1,000-line programs.

Good programming style demands that programs are built up in logical modules, each of which does one job. This helps when you have to track down random faults that occur when the program is RUNning. It helps you to see what is going on when you read the program and makes modifying the program at a later date much easier.

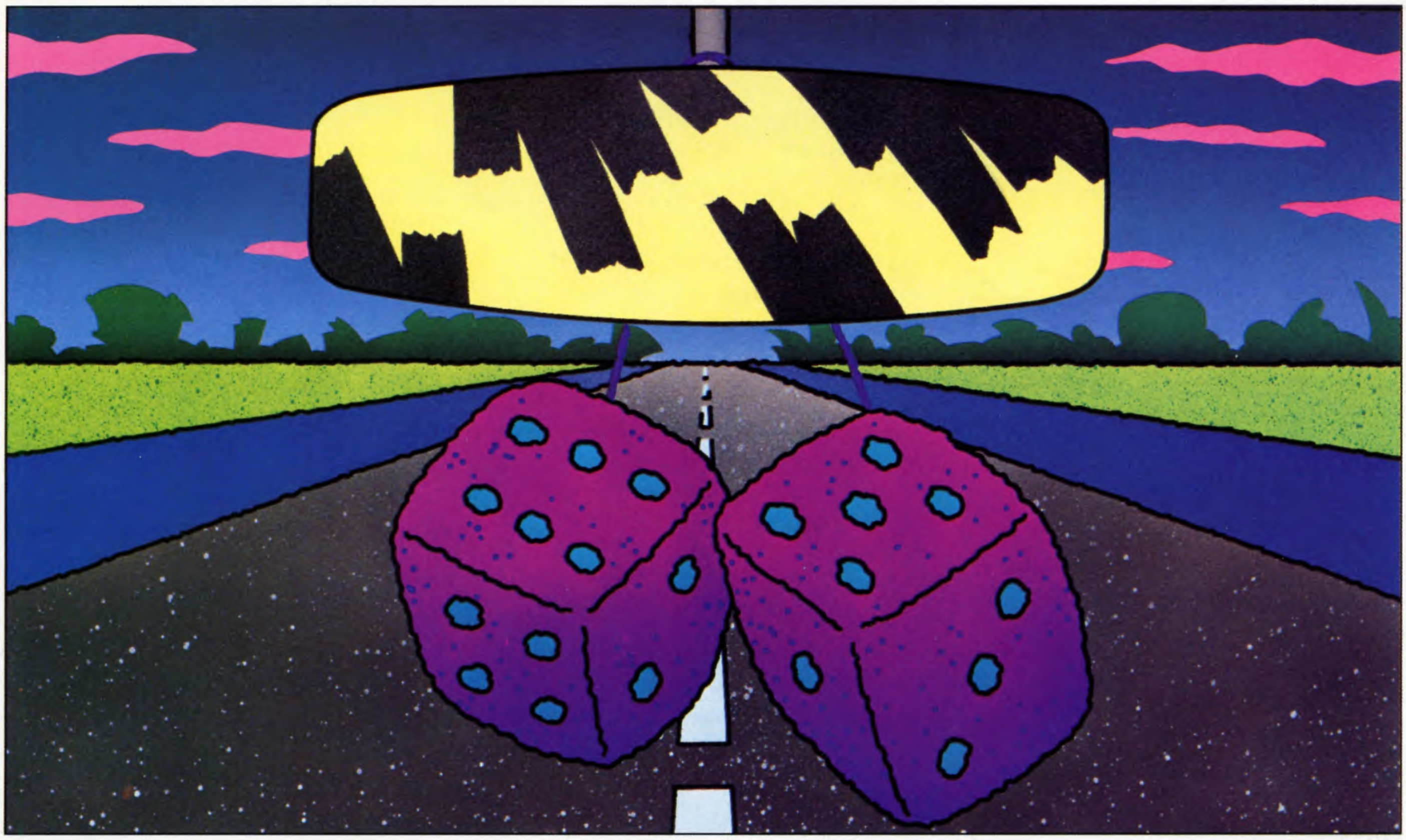
USING GOSUB

The programming tool which largely replaces GOTO in the sophisticated programmer's toolbox is GOSUB. Again it is keyed in as one word, and is followed by a line number.

GOSUB sends the computer to a *subroutine* which starts on the line number specified. A subroutine is simply a set of operations within the program that can be split off into a separate logical 'building block'. It is often used when an operation has to be repeated several times during a program. Instead of writing out the same routine each time it occurs in the program, the computer can simply be directed to the subroutine.

The crucial difference between a GOSUB and a GOTO is that at the end of a subroutine the word RETURN must appear. On the Spectrum, you hit the key labelled RETURN to do this. On the other machines, you must type in R E T U R N before hitting the RETURN or ENTER key. RETURN sends the computer back to the program line following the GOSUB, or, on the Spectrum to any statement following GOSUB in the same line.

The following program simulates the



American game of craps. In the game a pair of dice are thrown twice. Each time they are thrown the total is noted. If the totals of the two throws are the same the game ends; if not, the dice are thrown again.

```
20 LET A=1
30 REM..FIRST THROW..
40 GOSUB 150
50 LET T1=T
60 REM..SECOND THROW..
70 GOSUB 150
80 LET T2=T
90 IF T1=T2 THEN GOTO 120
100 LET A=A+1
110 GOTO 40
120 PRINT "EQUAL SCORES OF□";T1;
    "□IN□";A,"□THROWS"
130 END
140 REM..SUBROUTINE.
150 LET D1=RND(6)
160 LET D2=RND(6)
170 LET T=D1+D2
180 RETURN
```

```
20 LET A=1
30 REM ***FIRST THROW**
40 GOSUB 150
50 LET T1=T
60 REM ***SECOND THROW***
70 GOSUB 150
80 LET T2=T
90 IF T1=T2 THEN GOTO 120
100 LET A=A+1
110 GOTO 40
120 PRINT "EQUAL SCORES OF" T1 "IN" A
    "THROWS"
130 END
140 REM ***SUBROUTINE***
150 LET D1=INT(RND(X)*6+1)
160 LET D2=INT(RND(X)*6+1)
170 LET T=D1+D2
180 RETURN
```

On the ZX81, type this entirely in capitals:

```
20 LET a=1
30 REM first throw
40 GOSUB 150
50 LET t1=T
60 REM second throw
70 GOSUB 150
80 LET t2=T
90 IF t1=t2 THEN GOTO 120
100 LET a=a+1
110 GOTO 40
120 PRINT "Equal scores of□";t1;
    "□in□";a;"□throws"
130 GOTO 200
140 REM subroutine
150 LET d1=INT (RND*6)+1
160 LET d2=INT (RDN*6)+1
170 LET T=d1+d2
180 RETURN
```

The throw itself has to be performed twice, so it is consigned to a subroutine consisting of Lines 150 to 180. The GOSUB in Lines 40 and 70 sends the computer to Line 150 and the RETURN on Line 180 sends it back to Line 50 if it came from Line 40, or Line 80 if it came from Line 70. Line 140 just gives the name of the subroutine but it is not good programming style to include REM statements in subroutines. As subroutines are often performed many times, repeating the REM statement—which doesn't actually do anything—is a waste of time. So it is put on the line before the subroutine starts.

Note the END statement in Line 130 of all the programs except the Sinclairs'. If it was not there, after Line 120 the computer would run into the subroutine and display an error message when it got to the RETURN in Line 180, with no line to return to.

OUT OF RANGE LINE NUMBERS

As the Spectrum does not respond to an END statement, GOTO followed by a number beyond the range of the program is used instead. In this case we have:

```
130 GOTO 200
```

When it gets there and finds that there is no Line 200 it will assume that this is the end of the program and display OK, ready to begin again. This is better than using:

```
130 STOP
```

. . . for example, because when the Spectrum hits a STOP statement it throws up an error message. (In this case it would say: 9 STOP statement 130:1. This might lead you to believe that there was something wrong, especially as Line 130 is not the last line of the program—the subroutine follows it.)

But be careful when using GOTO followed by an out-of-range number. After GOTO 200, for example, the Spectrum will be left searching through all the lines following. And if it finds something on one of them it will execute, or try to execute, whatever that line tells it to do. So to be on the safe side it is best to send the computer to the last possible line of the program, that is Line 9999. So for Spectrum users the convenient 'end' statement is GOTO 9999 and on Line 9999 you should put:

```
9999 REM END
```

so that there can be no confusion about what you are doing.

BUILDING LAYERS

It is possible for one subroutine to call another one—or even itself—so you can build them up in layers. As the dice throwing here is performed twice it could be made into a subroutine within a subroutine by changing the last few lines of the program so that they look like this:

```
150 GOSUB 190
155 LET D1 = D
160 GOSUB 190
165 LET D2 = D
170 LET T = D1 + D2
180 RETURN
190 LET D = RND(6)
195 RETURN
```

```
150 GOSUB 190
155 LET D1 = D
160 GOSUB 190
165 LET D2 = D
170 LET T = D1 + D2
180 RETURN
190 LET D = INT(RND(X)*6 + 1)
200 RETURN
```

On the ZX81, type entirely in capitals:

```
150 GOSUB 190
155 LET d1 = d
160 GOSUB 190
165 LET d2 = d
170 LET T = d1 + d2
180 RETURN
190 LET d = INT (RND*6) + 1
195 RETURN
```

Here Lines 150 and 160 send the computer off to the subroutine in Line 190 to do the dice rolling, while Lines 155 and 165 make a record of the two separate scores.

On the other hand, for brevity's sake, it was actually unnecessary to perform the throwing of the two dice separately. Except on the Sinclairs, the whole of the subroutine could have been compressed to:

```
150 LET T = INT(6*RND(1) + 1) +
    INT(6*RND(1) + 1)
```

Alternatively, use this version:

```
150 T = INT(6*RND + 1) + INT(6*RND + 1)
```

On some computers you may even find that:

```
150 T = RND (6) + RND (6)
```

will work. But a skilled Acorn, Dragon or Tandy programmer could write the whole crap-shooting program in two lines:

```
10 A = 0
20 D1 = RND(6) + RND(6):D2 = RND(6)
   + 6:A = A + 1:IF D1 = D2 THEN PRINT
   "EQUAL SCORES OF□";D1;"□IN□";
   A;"THROWS": END ELSE GOTO 20
```

USING ON . . . GOSUB

As with GOTO, the GOSUB statement can on some computers be accompanied by a variable instead of a natural number. And on others an ON . . . GOSUB statement can be used to the same effect. These are used in more complex programs where a branch has to be made to a number of different subroutines.

PROCEDURES ON THE BBC

BBC BASIC allows the use of procedures. These are a bit like subroutines in that they are separate sections of program that are called from the main program. Unlike subroutines, though, they can have meaningful names such as PROCdrawtriangle, and they are a lot more versatile.

Here's an example:

```
10 CLS
20 FOR month = 1 TO 12
30 PRINT TAB(0,1)"Type in figures for
   month□";month
40 INPUT N
50 PROCdrawgraph
60 PRINT TAB(0,2)"□□□"
70 NEXT month:END
90 DEFPROCdrawgraph
100 FOR X = 1 TO N
110 PRINT TAB(X,month + 6)"*";
120 NEXT
130 ENDPROC
```

This program takes 12 figures, one for each month, and displays them as a graph. These figures could be amount of money saved, number of Mars Bars eaten, or any other amount that varies over the year. (As it stands each figure has to be less than 39 to fit on the screen.)

The procedure—PROCdrawgraph—is called 12 times, once for each month. Note that PROCdrawgraph calls the procedure, DEFPROCdrawgraph defines it and ENDPROC returns to the main program.

You can also pass numbers or *parameters* to a procedure, so a single procedure can be called with a different set of conditions each time. The next program uses one general procedure to print three colourful lines at various positions.

```
10 INPUT "WHAT IS YOUR NAME",N$
15 IF LEN(N$) > 20 THEN GOTO 10
20 MODE 2
30 VDU 23;8202;0;0;0;
40 PROCdisplay(5,1,130,"MERRY
   CHRISTMAS")
50 PROCdisplay(12,11,140,N$)
60 PROCdisplay(20,1,130,"AND A HAPPY
   NEW YEAR")
70 END
100 DEFPROCdisplay(row,ink,paper,
    message$)
105 LOCALX,L
110 LET L = LEN(message$)
120 col = INT((20 - L)/2)
130 COLOUR ink
140 COLOUR paper
150 PRINT TAB(col,row);message$
160 FOR X = 0 TO L - 1
170 PRINT TAB(col + X,row - 1) "*";
180 PRINT TAB(col + X,row + 1) "*";
190 NEXT
200 ENDPROC
```

You can display any text in this way simply by altering the parameters in the procedure call.

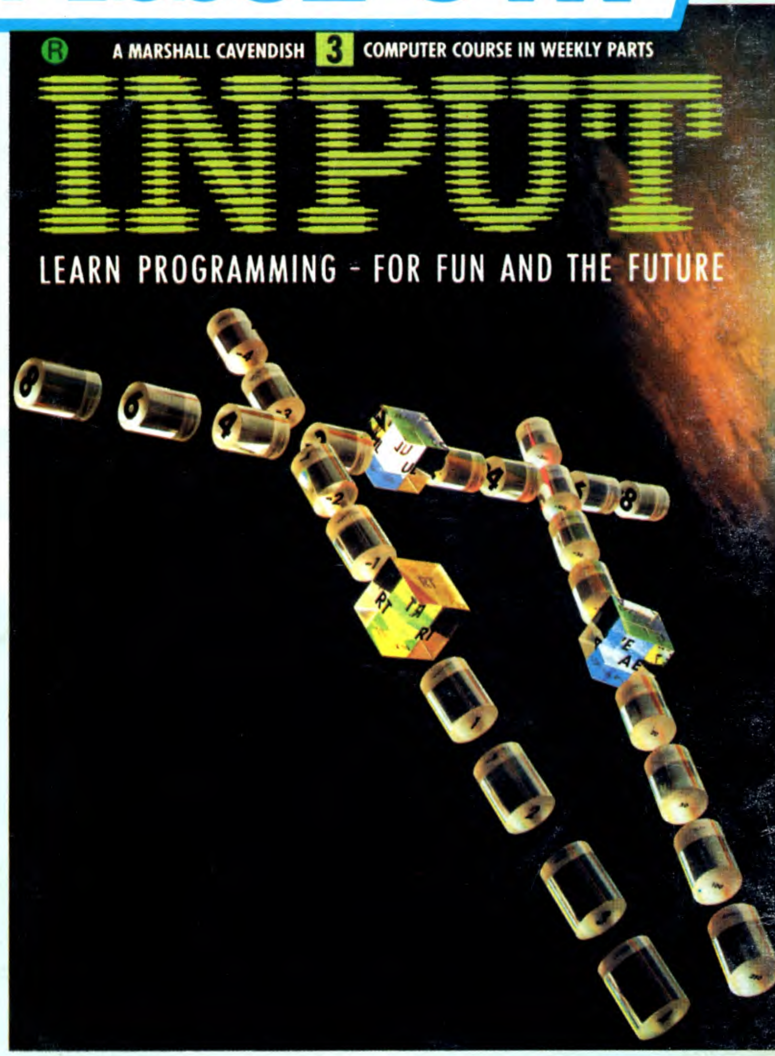
CUMULATIVE INDEX

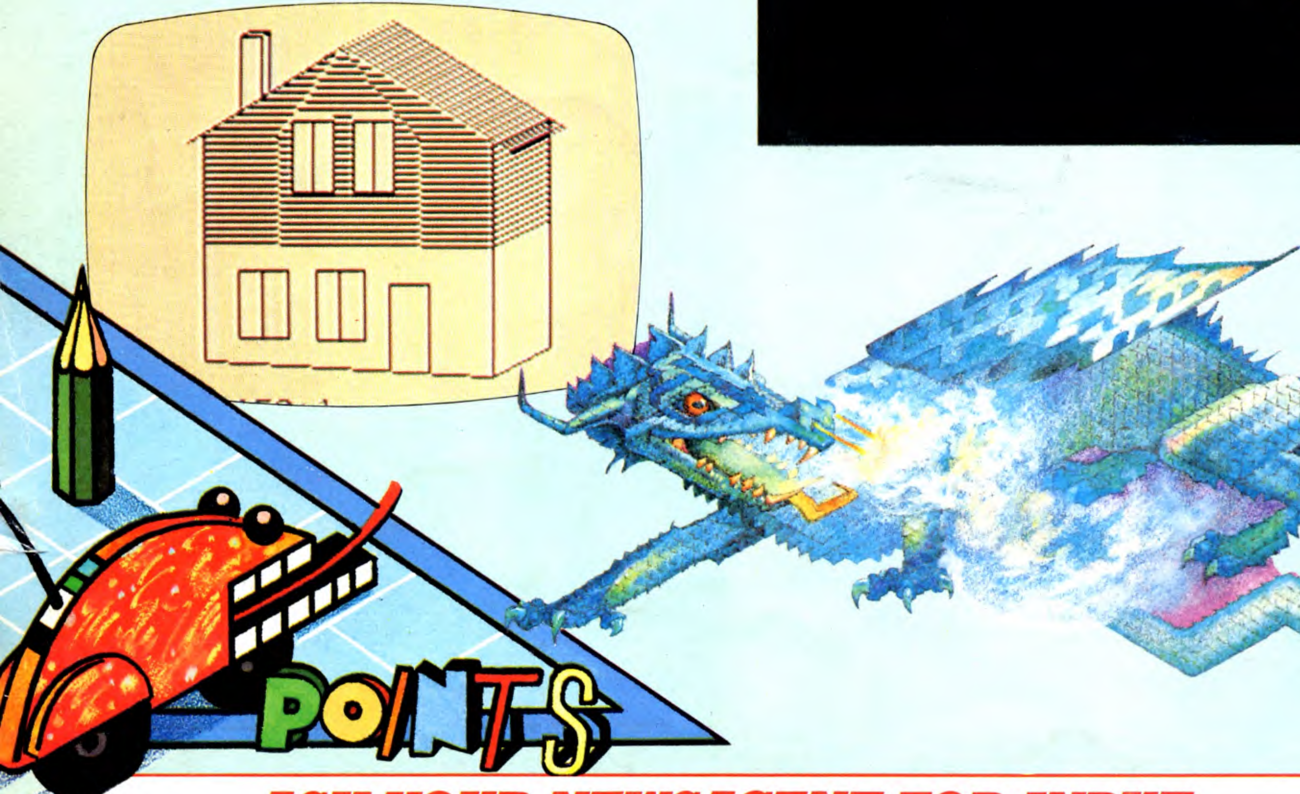
An interim index will be published each week. There will be a complete index in the last issue of *INPUT*.

The publishers accept no responsibility for unsolicited material sent for publication in INPUT. All tapes and written material should be accompanied by a stamped, self-addressed envelope.

COMING IN ISSUE 3 ...

- ❏ *MAZE GAMES have a long pedigree amongst computer owners. We show you how mazes are created — and how to make your own 'eater' to travel round the labyrinth*
- ❏ *Get started on some screen art by mastering your computer's PLOT and DRAW commands . . .*
- ❏ *Learn how to use your DATA FILING system fully, with routines for searching, amending and saving your records*
- ❏ *Start to learn about assembly language and machine code, the fast, accurate LOW-LEVEL LANGUAGES that bypass BASIC*
- ❏ *Add another graphic character to your repertoire, with a simple routine to create a moving, FIRE-BREATHING DRAGON*
- ❏ *Sort out your BASIC programs by understanding more about VARIABLES, the baffling Xs and Ys that hold the information*





ASK YOUR NEWSAGENT FOR INPUT