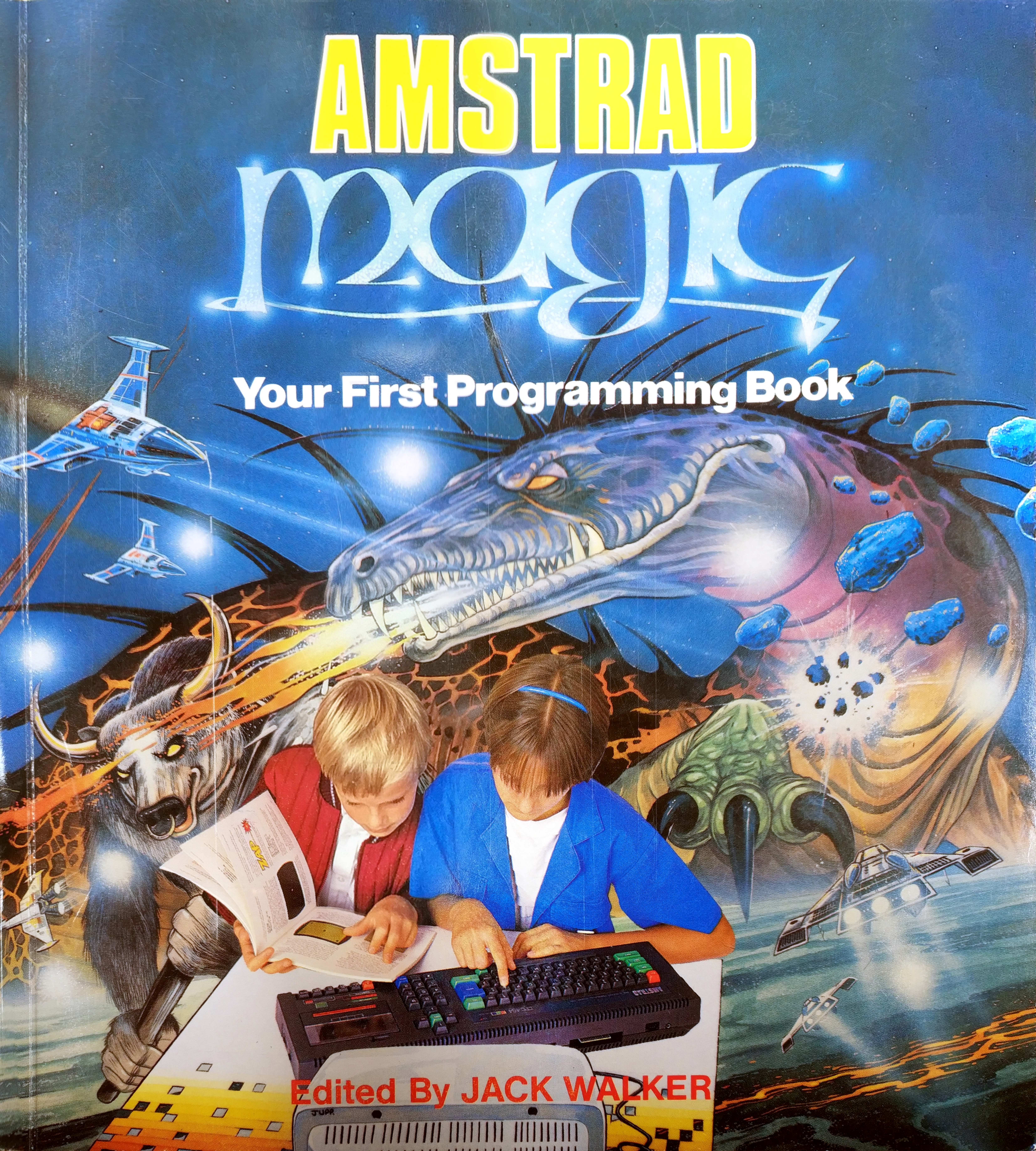


# AMSTRAD magic

Your First Programming Book

Edited By JACK WALKER



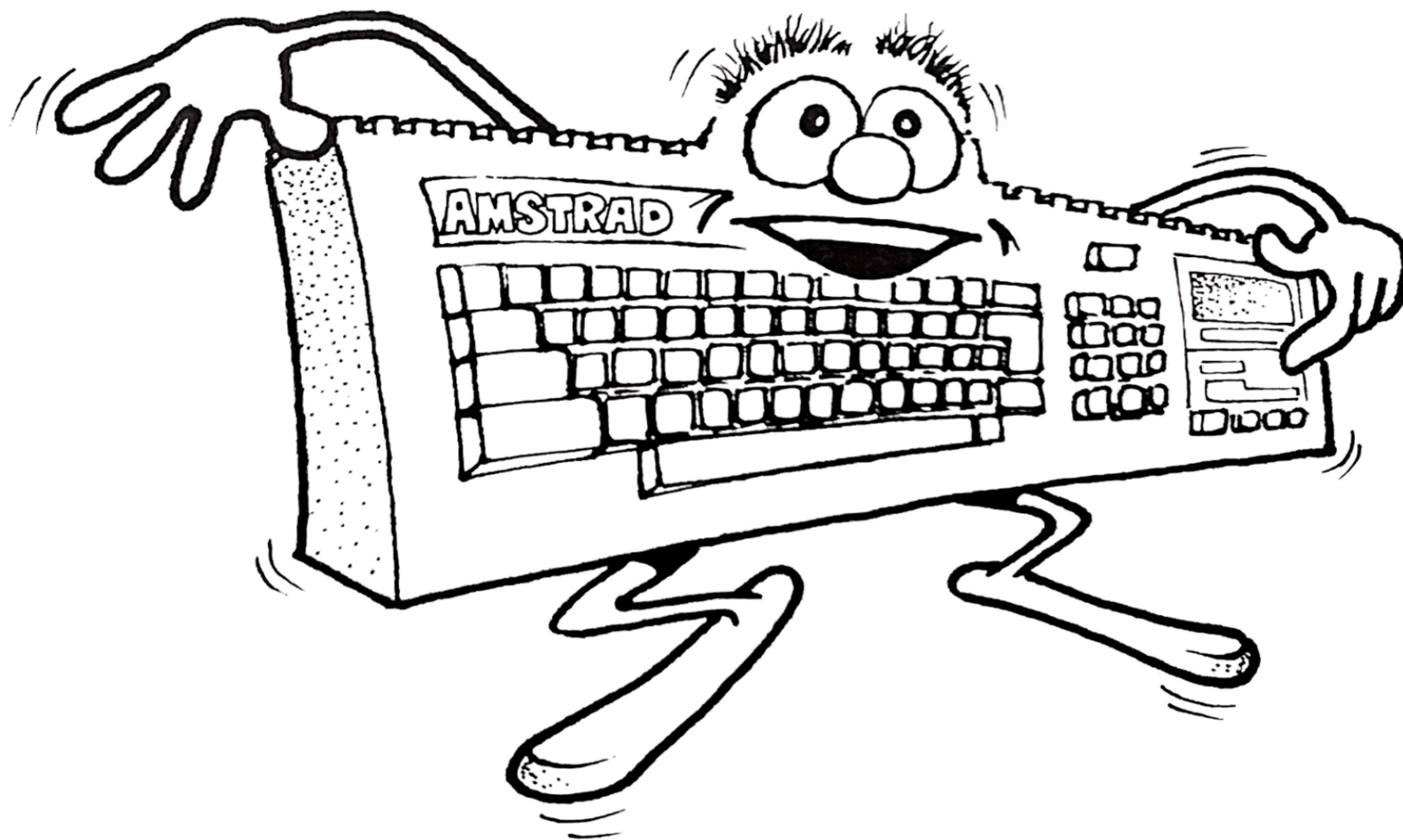


# Amstrad Magic

---

Your First Programming Book

Edited by Jack Walker



W. Foulsham & Co. Ltd.  
London · New York · Toronto · Cape Town · Sydney

W. Foulsham & Company Limited  
Yeovil Road, Slough, Berkshire, SL1 4JH

---

ISBN 0-572-01297-7

Copyright © 1985 W. Foulsham & Co. Ltd.

All rights reserved.

The Copyright Act (1956) prohibits (subject to certain very limited exceptions) the making of copies of any copyright work or of a substantial part of such a work, including the making of copies by photocopying or similar process. Written permission to make a copy or copies must therefore normally be obtained from the publisher in advance. It is advisable also to consult the publisher if in any doubt as to the legality of any copying which is to be undertaken.

Photoset in Great Britain by  
M. B. Graphics (Typesetting) Services  
and printed at The Alden Press, Oxford



# Contents

---

Note to parents	4	Chapter 8	Colour MODES	47	
Introduction	5	Chapter 9	Round and round we go	51	
Chapter 0	Getting started	6	Chapter 10	Decisions . . . decisions	59
Chapter 1	Words and numbers	9	Chapter 11	Using the LOCATE command	65
Chapter 2	Starting to program!	13	Chapter 12	Your first arcade game	68
Chapter 3	Making Amstrad remember!	19	Chapter 13	Lines and pictures	74
Chapter 4	Questions and answers	24	Appendix		80
Chapter 5	How to INPUT information	28	Glossary of commands		87
Chapter 6	SAVE-ing and LOADing your programs	34	Index		92
Chapter 7	REM, RENUMBER and EDITing	39	Answers		93



## Note to parents

---

This book is aimed at the young, first-time user or the not-so-young perplexed beginner! An introduction to the simple programming commands is covered in the first section of the book, while the second section concentrates on some of the graphics commands available on the Amstrad.

Many of the more advanced BASIC commands found in the language of the Amstrad have not been introduced, the concept of the book precludes all but the easiest of programming ideas. Through reading the book, young computer users will develop a sound base on which their new-found skills can be built. Many more complex books exist, but very few offer such an easy, step-by-step introduction for young people.

A few comments concerning the machine may be helpful, as many first-time users know little about the machine they have invested in.

The Amstrad is perhaps the most sophisticated computer in its price range. The language used is the powerful LOCOMOTIVE BASIC, an extension of the well known BASIC (Beginners' All Purpose Symbolic Instruction Code) computer language, which, despite its name, is now used by a very large number of professional programmers.

The Amstrad's graphics capabilities, that is, the ability to draw and colour areas of the screen, are very impressive for a machine in its price range.



# Introduction

---

You are growing up into a world that is changing very quickly. Not so long ago, having a home computer would have been very rare. Today millions of people own home computers.

This book is to help you learn to use the Amstrad computer. With the Amstrad you can draw pictures, play games, make music, do sums or (with additional equipment) even control robots! But before you try to write complicated programs, you must learn how to use some of the easy commands.

Work through this book carefully. Try to experiment with your Amstrad. You will learn

a lot by making small changes to programs. Remember that you are in control – the computer is just a willing slave! All you have to do is give it the correct instructions . . .

## **Important**

To avoid confusion between certain BASIC words and normal words, the BASIC language words are usually printed in CAPITAL LETTERS. If you come across a word printed inside odd-looking brackets like this

< ENTER >

it means press the **key** with the word *enter* printed on it.



# Chapter 0

## Getting started

---

There are three parts which make up the Amstrad computer system: the computer itself, a video monitor and a cassette recorder.

You will be able to add on other parts, such as joysticks or a printer or a floppy disk when the special connectors become available. Each part of the system has a special job to do, so let's have a look at what each part does.

### **The computer**

The computer is the heart of the system. Inside it is a very complicated electrical circuit using microprocessors – which you know as 'chips'. These 'micro-chips' use electrical signals to make the computer very fast at working out problems that might take you or me several days.

Never take the computer or its power supply apart; you may damage yourself and the computer. The power supply uses mains electricity which can be dangerous, so if anything goes wrong with your Amstrad, you must get an expert to fix it.

On its own, the computer cannot display pictures or words for you. A second part to the

system is a video monitor.

The Amstrad will work with either a colour monitor or an ordinary green tube monitor, but obviously, you won't see the colours on a green tube monitor.

### **Large and small letters**

Now try pressing some of the keys. To get the Amstrad to print capital letters on your TV screen, hold down either of the keys marked

**SHIFT**

while you press the key marked

**CAPS LOCK**

at the left of the second row of keys. This is called the CAPS LOCK key because it 'locks' the keyboard into all-capitals mode.

To test this, try typing something along these lines:

**JOHN BLOGGS is my name**

If you make a mistake when typing, press



the DELETE key marked

## DEL

which is in the top right-hand corner of the keyboard. This removes ('rubs out') the character immediately on the left of the cursor.

When the **CAPS LOCK** is off the computer acts like a normal typewriter – everything appears in small letters, unless you hold down the **SHIFT** key.

## Special keys

Now press the key marked



## ENTER

This is on the right of the keyboard, third row down. The computer will probably reply

## Syntax error

## Ready



Don't worry – this is just the Amstrad's way of telling you that it doesn't recognise what you have typed as a proper BASIC instruction.

Underneath the word 'Syntax error' you should see a READY message. This is called the BASIC *prompt* or just *prompt* for short. Beneath it is a square which is called the cursor.

The *prompt* tells you that the Amstrad has finished what it was doing and is ready for another instruction. The *cursor* tells you where the next character (letter, symbol or number) will be printed on the screen.

## Symbols

Take a look at the keyboard. You may notice that some keys have two symbols printed on them. To get the top symbol you have to hold down the

## SHIFT

key as you press the key for the symbol you want. See if you can make the Amstrad print some of the following symbols on the screen.

(! " £ \$ % & ' ) < > + \* ? <

### Auto-repeat

What happens if you keep a letter or number key pressed for more than a couple of seconds?

What happens if you hold down the key marked DEL?

This *auto-repeat* effect can be very useful

when you want to type a whole row of letters or symbols such as a row of asterisks (\*) or stars, or when you want to rub-out whole words or sentences.

Enough of this messing about! You probably want to get your computer doing something. So let's move on to try some simple ideas.

### Remember

1. Never eat or drink near the Amstrad. Crumbs and spilt milk can ruin it.
2. Never open the Amstrad. Curiosity kills computers!
3. When the green CAPS LOCK key is pressed on, you get *capital letters*.
4. When the green CAPS LOCK is *off*, you get *small letters* when you press a letter key. This is the normal typewriter mode. Use SHIFT to get occasional capital letters.
5. The DELETE key rubs out the character on the left of the flashing cursor.
6. If you keep any key depressed for more than a second or two, it will start to *auto-repeat*.
7. The ENTER key tells the computer that you have finished typing and want it to do something. If you have given it a proper BASIC command it will carry out your instruction. If not, it will tell you that you have made a mistake by printing **Syntax error**.
8. When your instruction has been carried out, the BASIC prompt READY and cursor will re-appear.
9. The *prompt* tells you that the computer is waiting for another INSTRUCTION or COMMAND.
10. The *cursor* line tells you where the next character will be printed on the screen.
11. To obtain the symbols printed at the top of a key, hold down SHIFT while you press the required key.



# Chapter 1

## Words and numbers

---

Computers are able to work out sums very quickly. But unlike pocket calculators, they can also print messages on the screen of video monitor, and have very large memories.

To make your Amstrad do something, you have to tell it what to do, using a special language called BASIC.

BASIC is a very easy language to learn. Whereas you already know thousands of English words, your computer only understands around a hundred very simple words like PRINT, LIST, END, and a few specially made up commands like CLS – which is short for CLear the Screen.

Let's try using some BASIC commands.

Commands may be keyed in capital or small letters. Your Amstrad will convert all commands entered into capital letters.

Type:

**CLS**

After typing a command, you must always

press the key marked

**ENTER**

to tell the computer that you want it to obey your command. So after typing CLS and pressing the ENTER key, the screen should clear. The *prompt* (READY) and the cursor should then appear at the top of the screen.

If you make a mistake when typing, use the DELETE key – one press rubs out the last letter typed, provided you haven't yet pressed <ENTER>.

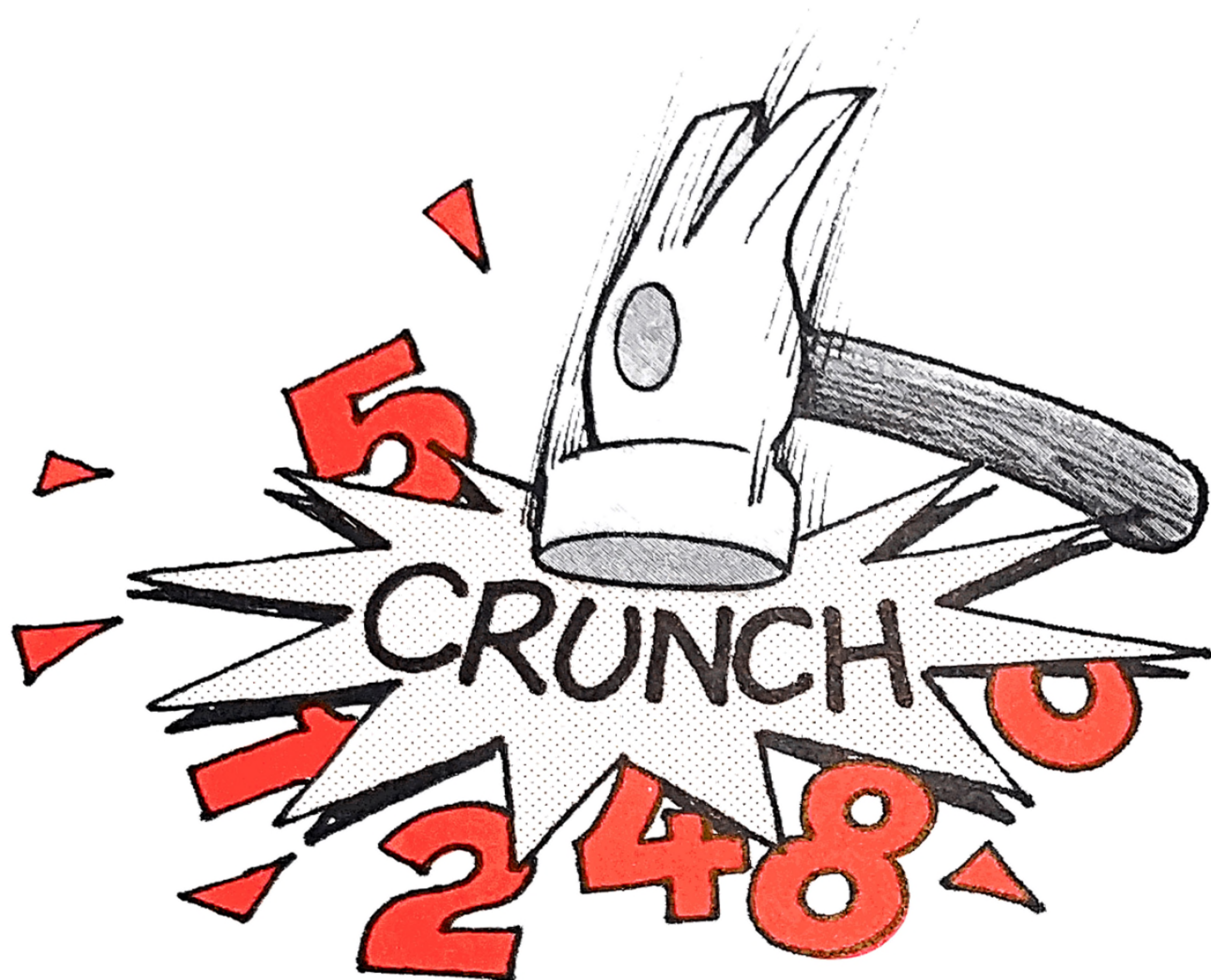
**Number crunching!**

Now you can try using the command

**PRINT**

to make the computer PRINT some messages on the screen. We shall start with some sums, to see how quickly the Amstrad can work them out. Computer people call this 'number crunching'. Try typing something like this

**PRINT 5678+7890 <ENTER>**



Remember you can use *capital* or *small letters* for the word PRINT. Press the ENTER key to make the computer work out the answer. The screen should now look like this:

```
PRINT 5678+7890
13568
Ready
■
```

You now know how to use the computer like a calculator! The main difference is that the word PRINT has to be used to tell the computer to PRINT the answer.

Now try the following sums, which use addition and subtraction. The subtraction sign (or minus sign) is next to the zero (0). Unlike the addition sign, you don't have to press SHIFT to get the minus sign. Don't forget to

press the ENTER key after each sum to get the answer.

```
PRINT 3567-987      < ENTER >
PRINT 4567+89765    < ENTER >
PRINT 34535-568+45678
PRINT 5678+3456-567 < ENTER >
                       < ENTER >
```

If you want to multiply or divide, you have to use some strange signs.

### Multiplication

To multiply, we use the \* (star) sign, instead of an x. This is because the computer might confuse the normal 'times' sign with the letter x. So you enter the sum  $4 \times 3$  into the computer as

```
PRINT 4 * 3
```

then press < ENTER > to get the answer.

The \* key is near to the ENTER key. Notice that there are two symbols printed on this key – a star (\*) at the top and a colon (:) below. Because the star is printed at the top, you will need to hold down the SHIFT key to get the star sign.

Just for fun, try this one, which stretches your Amstrad to the limit of its accuracy:

```
PRINT 11111.111 * 11111.1111
```

Be careful to put *five* ones in front of the decimal point and *three* after it the first time.

Note that the second number has *five* ones in front of the decimal point and *four* after it.

### **Division**

To divide, we use the / (slash) sign, which is printed below the ? (question mark – or ‘query’) on the bottom row of keys. As the slash sign is printed at the bottom of the key, you don’t need to press <SHIFT> this time.

So 20 divided by 5 would be typed in as

**PRINT 20/5 <ENTER>**

Try some more sums using \* and /. Make them as difficult for the computer as you can by using really big numbers. Don’t forget to press <ENTER> to get the answer. The computer will then let you know that it is ready for the next job by displaying the READY prompt and the cursor square.

### **Too big!**

If the answer becomes too big for even the Amstrad to cope with, you’ll find that it gives you some strange looking answers, like this

**9.87654321E+8**

The E+8 at the end of the number means ‘move the decimal point EIGHT places to the RIGHT’. If it read

**9.87654321E-6**

the E-6 would mean ‘move the decimal point SIX places to the LEFT’. This is called

*exponential notation*. Don’t worry if you can’t understand it – it isn’t really important!

### **Making mistakes!**

Take your time typing and don’t worry if you make mistakes – the computer won’t be harmed. Unlike most parents and schoolteachers, the computer has infinite patience. It won’t get angry, or call you a ‘steaming twit’! All it does is print out the message

**Syntax error**

on the screen. For the moment, just type the command again and all should be well.

If you spot a mistake before you press the ENTER key, you can use the DELETE key to rub out the error.

### **Words and sentences**

You can also print words and sentences on the screen. The difference is that words have to be placed between quotation marks, just like when you use speech marks (or ‘quote marks’) in your English lessons at school. The “ (quote) mark is on the 2 key. You will have to use SHIFT to make it appear. Try the following.

**CLS <ENTER>**  
**PRINT “All pigs have wings”**

What do you have to press to make the computer obey your instruction?

The computer will print *any* message that



you put between quote marks. Even if you make a spelling mistake, the computer will print it exactly as you asked it to.

You can also PRINT numbers inside quote marks, but they cannot be used to do sums like the numbers you typed at the beginning of this chapter.

Because the computer prints anything in quote marks, you can make it appear foolish! Try this

**PRINT "2+2=5"**

If you want some more practice finding your way round the keyboard, try getting your

Amstrad to print some really stupid things, like

**PRINT "Elephants are fish with curly hair and gossamer wings"**

### Remember

1. You may use *capital or small letters* when typing a COMMAND. Your Amstrad will convert the commands to capital letters.
2. Use the CLS command to CLear the Screen.
3. After giving the Amstrad a command, you must press the ENTER key to make it carry out your instruction.
4. The PRINT command can be used to print numbers, symbols, words or sentences.
5. To PRINT words or sentences, they must be enclosed in double quotation marks ("").
6. To do simple arithmetic, use the +, -, \* and / signs.
7. The computer cannot spell. Words or sentences to be printed must be typed in carefully.



## Chapter 2

# Starting to program!

You may not realise it, but computers have simply dreadful memories. The moment you turn them off, they forget everything you have told them.

Unlike you and I, computers don't ever really learn anything. Each time you want your computer to do something, you have to tell it precisely what to do, and what order to do it in.

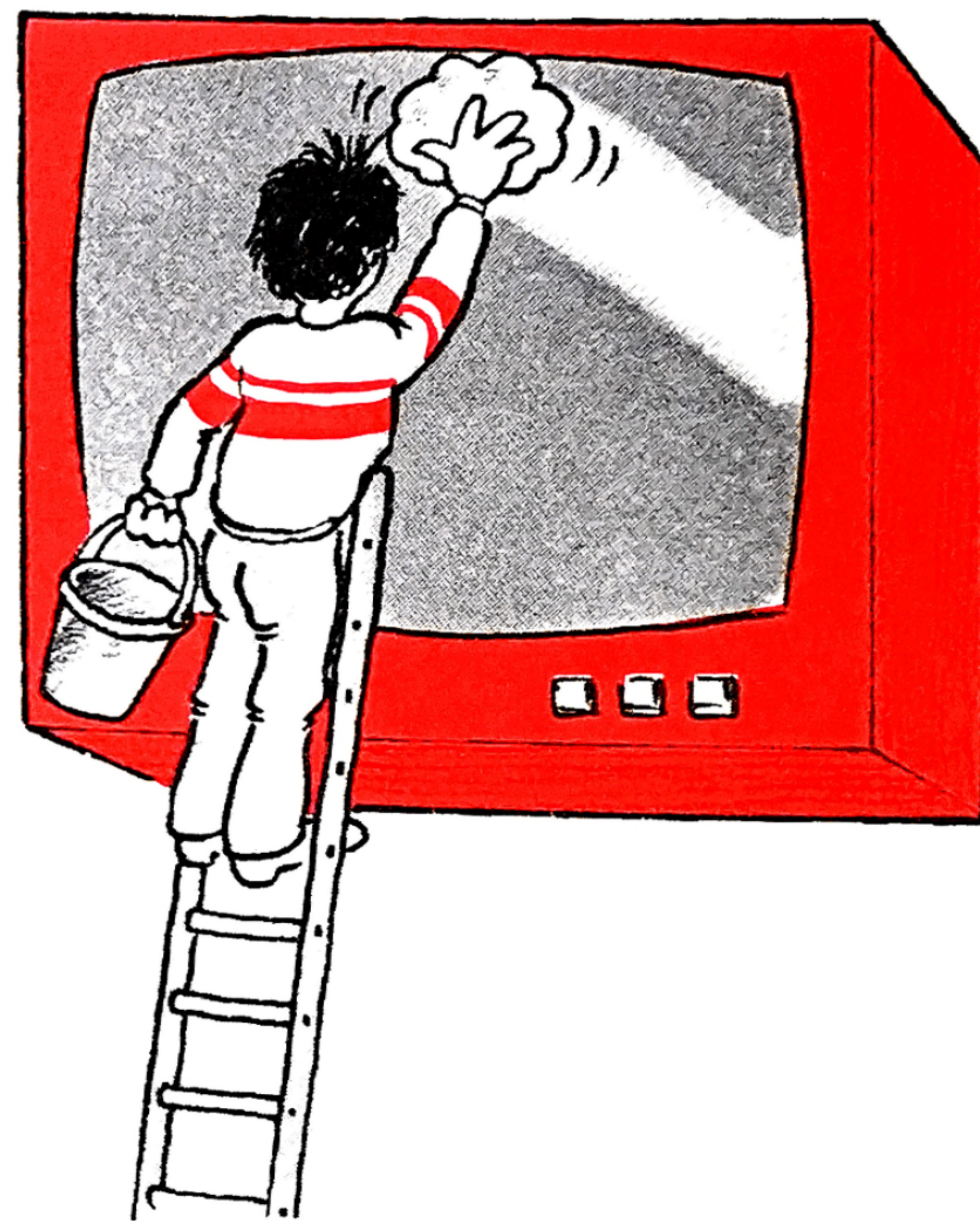
So far, you have been making the Amstrad do one thing at a time, by giving it *direct commands*, like

```
PRINT 3 * 4 <ENTER>
```

to which it gives an immediate answer.

To make your Amstrad do more interesting and complicated things, you have to give it a *list* of things to do. This list is called a *program* – using the American spelling, with only one 'm'.

A *program* is simply a list of commands, like 'clear everything off the screen', 'print my name', 'now print something else'.



Unfortunately, while you probably know and use several thousand words of English in everyday life, your Amstrad only knows about 154 BASIC keywords. (Even this is about double the number of words that many home computers understand).

So you have to learn to write your programs using words that the Amstrad can understand. And you make sure that the computer follows the instructions in the right order by giving a number to each program line, like this:

```
10 CLS
20 PRINT "BRIAN"
30 PRINT "is a clever lad"
40 END
```

You could number the instructions (or *program lines*): 1, 2, 3, 4, 5 etc, but it's usually better to go up in steps of ten, like this: 10, 20, 30, 40, 50.

There is a good reason for this. If you want to place additional instructions between program lines, it is easy if the numbers are ten units apart.

For example, if you wanted to add your surname to the above program, you could add it anywhere between lines 20 and 30. Let's choose the half-way mark:

```
25 PRINT "SMITH"
```

the program now becomes:

```
10 CLS
20 PRINT "BRIAN"
25 PRINT "SMITH"
30 PRINT "is a clever lad"
40 END
```

If you had numbered your program lines 1, 2,

3, instead of 10, 20, 30 there would have been no room to squeeze extra lines in between them!

### **RUNning a NEW program**

Before you type in your first program, make sure that there are no old instructions in the computer's memory by typing

```
NEW < ENTER >
```

The command NEW tells the computer to clear any old program from its memory and get ready for a fresh program. If you forget to do this you can sometimes run into horrible problems when old program lines start interfering with a new program!

Now type the following two program lines. Try not to change anything. If you make a mistake, use the

```
DELeTe
```

key to rub out the mistake and retype.

At the end of each program line, check that you have typed it in correctly, then press

```
ENTER
```

If you find you have made a mistake after pressing the ENTER key, type the whole line in again.

```
10 PRINT "HELLO ";
20 GOTO 10
```



OK, you've typed the program in correctly – but absolutely nothing has happened!

To make the computer carry out the program, you need to type a special BASIC command. Type the word:

**RUN**

then press the ENTER key. The computer should immediately carry out the program instructions and screen full of 'HELLOs' should appear!

### How to stop it!

When everything is correctly typed in, and after typing RUN and pressing <ENTER>, you should see lots of words go whizzing up.

To stop the program press the red ESCAPE key marked

< **ESC** >

This tells the computer that you want it to stop carrying out the program temporarily.

Now you should be able to *read* the words on the screen!

After it has printed enough words to fill the screen, the Amstrad automatically moves the words up the screen so it can print more down below; this is called SCROLLing. It can be a nuisance at times.

Now RUN the program again, but this time,

instead of typing the word RUN, press any key.

Remember, you can completely *stop* the program at any time by pressing twice the key marked

< **ESC** >

### Bug hunting!

Programs seldom work perfectly the first time you try to RUN them. Usually the problem is a tiny typing error, which in a long program can take ages to find. Finding and correcting typing errors and program mistakes is called 'de-bugging'.

The Amstrad helps you find some (but by no means all) errors by printing certain error messages on the screen, such as

### Syntax error in 10

If the computer has told you that you have made a mistake, then you must check the following points:

1. Have you typed the letter (O) when you should have typed zero (0)? It is absolutely essential to use the **Zero** key (the 0 with a slash through it) when you want number zero (or nought). Only use the **O** key (without the slash) when you want the letter 'O'.

2. Have you made a spelling mistake? Check that you have spelled PRINT and GOTO correctly – using letter 'O' this time, not zero!

3. Have you used the double quote mark <"> at both ends of the word to be printed?

4. Did you put the semi-colon <;> in the right place (at the end of line 10)?

5. Did you leave a space after the word HELLO, but before the second set of quote marks?

### How it works

You know what the PRINT command does, so line 10 is easy to understand. It tells the computer to PRINT the word inside the quote marks – HELLO.

After carrying out the instruction in line 10, the computer moves on to the next biggest program line number – line 20. This tells it to GOTO another place in the program.

In this program, the GOTO line sends the computer back to line 10 where (surprise! surprise!) it finds the instruction to PRINT the word 'HELLO' again. Then it goes on to line 20 which sends it back to print the word again. It will carry on doing this (as fast as the micro-chips will let it) for ever and a day – unless you press ESCAPE or turn the power off!

### Making changes

You can make several changes to this program. It is worth making changes because you can learn a lot by altering programs.

Before changing a program, you need to

see the LISTing on the screen. So type the following commands:

```
CLS <ENTER>  
LIST <ENTER>
```

The LIST of program lines should now appear on the screen.

1. Change line 20, by typing:

```
20 GOTO 20 <ENTER>
```

Now type LIST again to see that the line has changed. You should still be able to see the old program on the screen (a CLS first of all would have got rid of it).

Before you RUN the revised program, what do you think will happen? Try it. Type RUN, then ENTER to make the program work.

Now press ESCAPE twice and change line 20 back to

```
20 GOTO 10 <ENTER>
```

Here's another change you can try:

2. Change line 10 to –

```
10 PRINT "HELLO "
```

Can you spot the difference? Find out what happens when there is no semi-colon at the end of line 10.

If you place a semi-colon <;> after a word that is to be PRINTed, the computer will print the next word immediately afterwards – on the same line if there is room. But if you leave the semi-colon out, each item will be printed on a separate line.

### Patterns

You can make some interesting patterns on the screen using the PRINT and GOTO commands.

In the next program, Line 10 uses CLS to clear the screen; then line 20 instructs the computer to print a line of stars and dots, instead of the word “HELLO”.

Line 30 is a special line that tells the computer to check if it has filled the screen. If so, the program should end. We shall look at the commands IF and THEN later in the book.

To get rid of any old program, type:

```
NEW < ENTER >
```

before starting to type these new program lines:

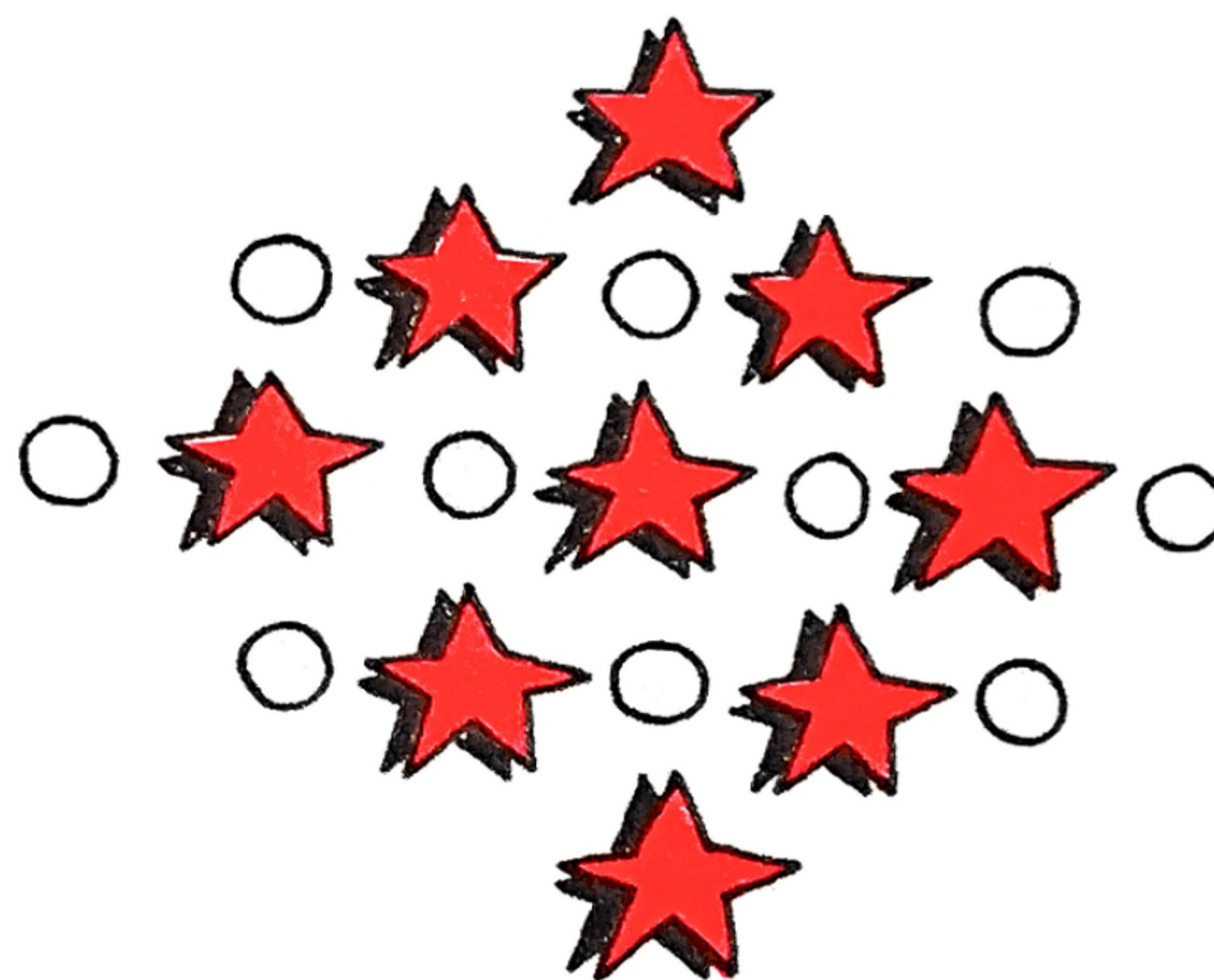
```
10 CLS  
20 PRINT "**** .."  
30 IF V=23 THEN END  
40 V=V+1:GOTO 20
```

Now RUN the program. Remember to press ENTER.

This time, the computer should end its task ready for further action! Can you see the prompt cursor (■) at the bottom of the screen? If you want to clear the screen and get back to the program listing, type the following commands:

```
CLS < ENTER >  
LIST < ENTER >
```

Try changing line 20. Make the computer print different patterns by using a different number of stars or dots. You might try using words instead of symbols.



### Remember

1. A program is a list of instructions, written in a way that your computer can understand.

2. Every line in a BASIC program must start with a *line number*. Line numbers usually go up in steps of ten, to leave room for additional lines to be inserted.
3. Type NEW before you enter a new program. This clears any old program lines from the computer's memory.
4. Press ENTER after typing each program line. This tells the computer that you have finished the line.
5. Be careful always to use the correct slashed ZERO key (0) for number zero (or nought) and O (without the slash) when you want *letter* 'O'. This is a very common cause of programs not working.
6. To start a program, type RUN, then press the ENTER key.
7. The command GOTO makes the computer continue with the program line number which follows the GOTO command.
8. You can usually stop a program by pressing the ESCape key once and you can continue with the program by pressing any other key. You can completely escape from the program by pressing the ESCape key twice. The computer is then ready to receive some more instructions.
9. The command LIST makes the computer print out the program on the screen.
10. Placing a *semi-colon* (;) after a word that is to be PRINTed, makes the computer print the next word immediately afterwards, usually on the same line. (The semi-colon must follow directly *after* the second set of quote marks. If you place it *inside* the quote marks, it will be PRINTed on the screen.)
11. If there is no semi-colon (;) or comma (,) after the second set of quote marks, the next item to be PRINTed will appear on a fresh line.



## Chapter 3

# Making Amstrad remember!

---

Inside your Amstrad is a group of microchips which can store electronic signals representing numbers and words. We call these chips its *Random Access Memory* – or RAM for short.

The best way to understand the Amstrad's random access memory is to think of it as thousands of empty boxes. You can fill any of these boxes with numbers, letters, symbols, words or sentences – in fact anything that you want the Amstrad to remember.

Before you can put something into a memory box, you have to 'write a label' for the box – otherwise the Amstrad won't know which of its thousands and thousands of boxes to put it in – and you won't know where to find it!

The memory box label can be almost any letter, or a descriptive word. For example, if you wanted the Amstrad to remember that you have 14 model cars you can type:

```
LET cars=14 < ENTER >
```

Try typing this, using capital letters for the

command LET and small letters for cars.

Note that there are no quote marks round the word cars. This is because you are using the word as a *label* for a memory box, and not as something you want PRINTing.

Now type:

```
PRINT cars < ENTER >
```

Like magic, the Amstrad responds with the answer:

```
14
```

What happened is this. First you stored the number 14 in a memory box labelled 'cars'. Then, when you asked the Amstrad to PRINT cars (without quote marks), it looked in the memory box labelled 'cars', found the number 14 inside and copied it onto the screen.

What would happen if you *were* to enclose the word "cars" in quotation marks? Try it and see:

```
PRINT "cars"
```

(Don't forget to press <ENTER> – we won't be reminding you any more!)

Now try it again, without the quote marks:

**PRINT cars**

Let's put something into another memory box. Suppose you want the computer to remember that you have 5 model trains. Just type:

**LET trains = 5**

Now you have put the number 5 into a memory box labelled trains. To check that it is there, clear the screen with

**CLS**

then type:

**PRINT trains**

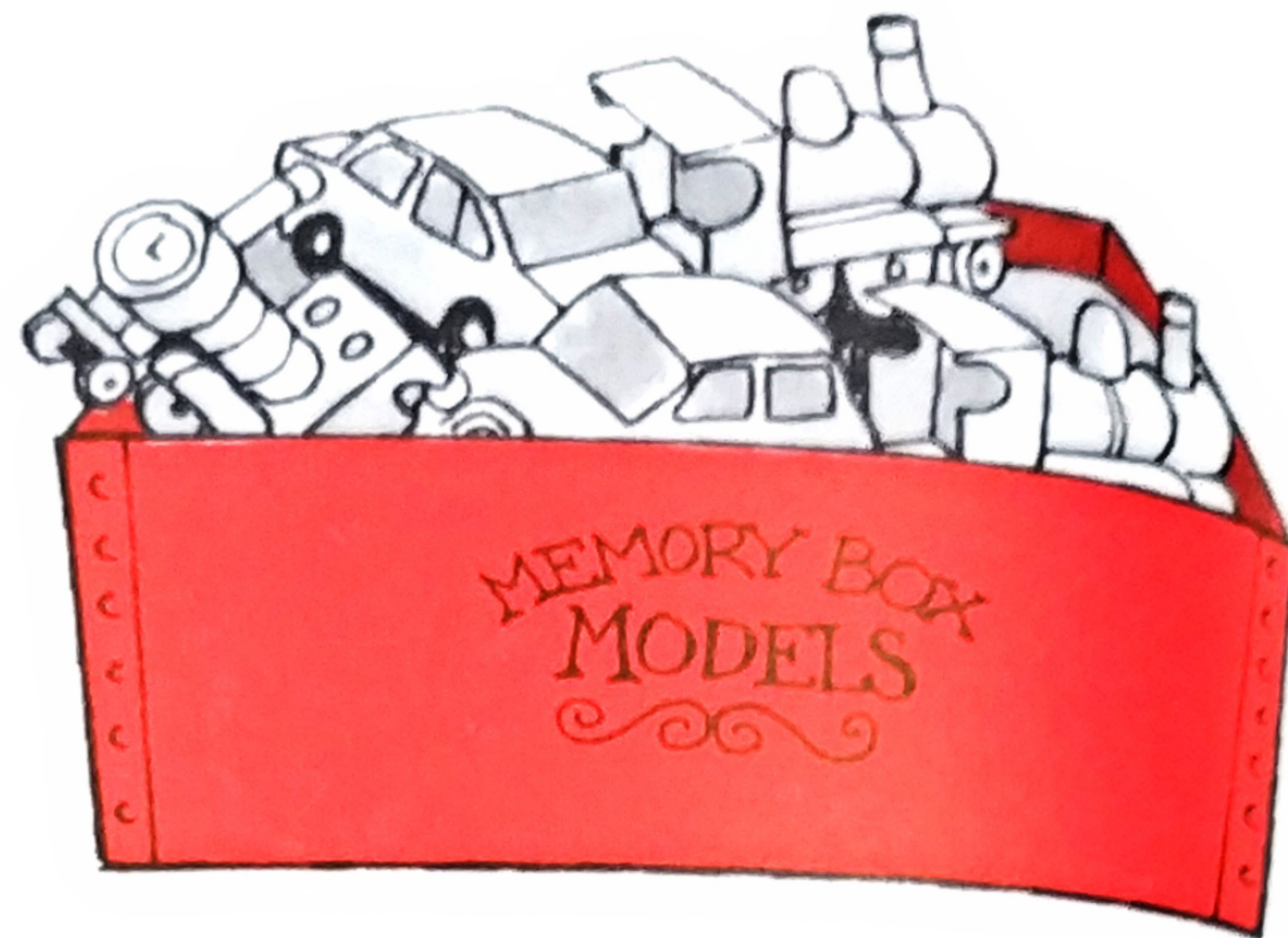
Did you get the answer 5? If not, you must have typed it in wrongly, so type it in again.

If you want to know how many toys you have altogether, you can get the computer to count them up for you. Type:

**LET models = cars + trains**

This *formula* line may seem a bit strange if you haven't done any algebra.

What it means to the computer is 'look in



the memory box labelled cars, then add the number you find there to the number you find in the memory box labelled trains. Put the result into a new memory box labelled models'.

What number will it find in the memory box marked 'cars'? And what number will it find in 'trains'?

So what number would you expect it to put in the box labelled 'models'?

To find out if you were right, type

**PRINT models**

Don't worry if you got it wrong the first time. *Numeric variables* as these numbers and labels are called, confuse most people to start with, but you will soon get used to using them.

Note that to save typing, you can leave out

the word LET and it will still work. So instead of typing

**LET trains = 5**

you can type simply:

**trains = 5**

to get the same result with less effort!

### **Mindbender 3.1**

To test whether you really understand what is happening, imagine that you have added three model planes to your collection. How would you get the computer to tell you how many models you have altogether? Try to work it out for yourself before looking at the answer at the back of the book.

HINT: planes = ?  
models = ?

### **Mindbender 3.2**

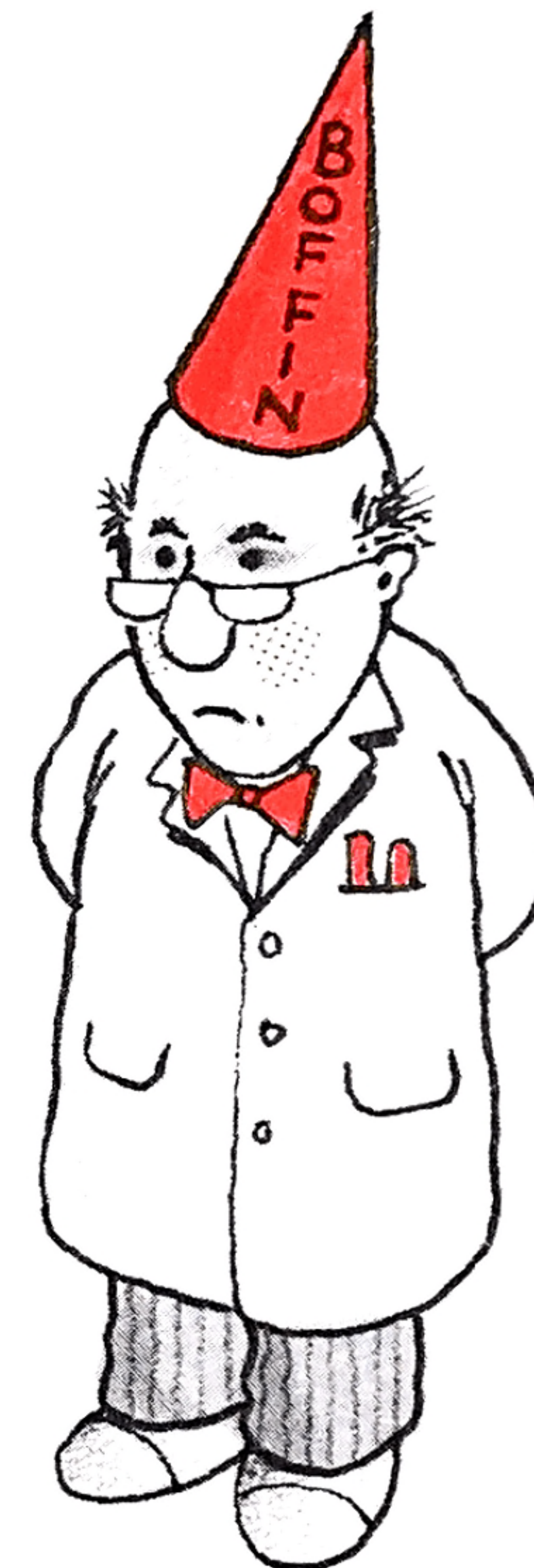
Now try this one. Let's pretend your record collection contains 12 singles and 8 albums (LPs). How would you get the Amstrad to tell you how many records you have altogether?

HINT: singles = ?  
albums = ?  
records = ?

### **Variable names**

In Amstrad BASIC, numeric variable names (or, as we have been calling them, 'memory box labels') can be:

- a single letter, or
- almost any combination of letters and numbers (as long as the name starts with a letter), or
- a (descriptive) word



Here are a few examples of numeric variable names that you could use:

age, a, A, A1, A2, size, s, S, length, L

### Boffin corner

The technical name for this type of 'memory box label' is *numeric variable*, because you can *vary* the number in the memory box.

To test this, type:

```
LET cars = 12  
PRINT cars
```

Now change (vary) the number in the memory box, by typing, say:

```
LET cars = 15  
PRINT cars
```

If you have kept your Amstrad switched on all through this chapter, you should now get a different answer when you type:

```
PRINT models
```

From these examples you can see that you can use capital letters as well as small letters. So instead of typing:

```
LET cars = 10
```

you could have typed:

```
LET CARS = 10
```

Provided you also type the word 'cars' in capital letters in the formula, this would work just as well.

### Remember

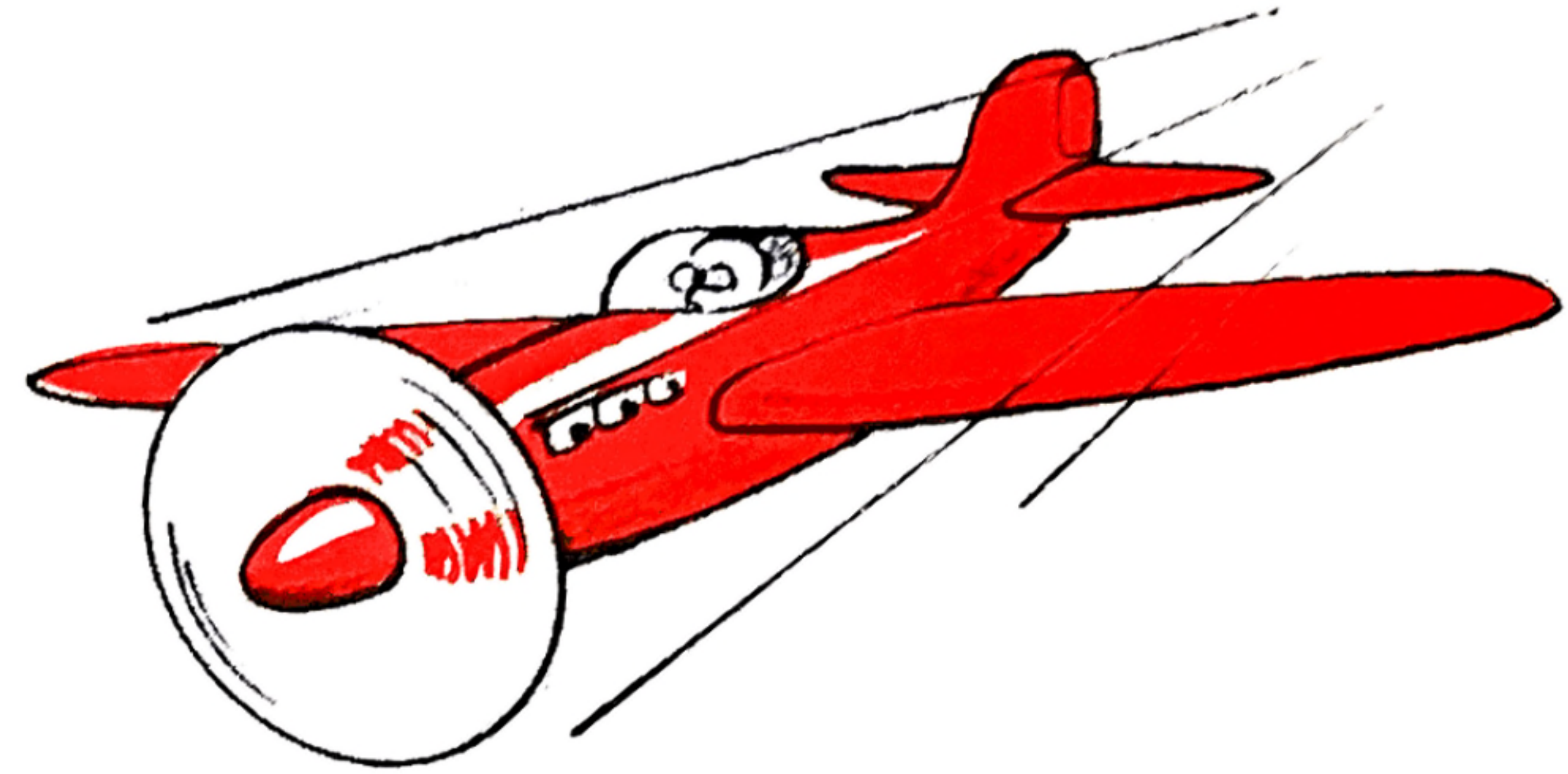
1. You can put any number into the Amstrad's memory by 'labelling a memory box', e.g. LET cars = 50. Technically, this is called assigning a *value* (50 in this case) to a numeric variable (which we have decided to name 'cars').
2. The name of the variable (or 'label') can be a letter, word or even a combination of letters and numbers (e.g. LET B3 = 5, LET A = 3, LET total = price + VAT).
3. It is best to stick to small letters for the labels or *variable* names. Never use words in *capitals* that could be confused with a BASIC command like TO or PRINT. (e.g. You must *not* use LET TOTAL = PRICE + VAT, because TOTAL and PRICE – in capitals – would be confused with the BASIC words TO and PRINT)
4. You can check what number the variable contains, using the PRINT statement, (e.g. PRINT B3 or PRINT total). Note that there are no quotation marks round the variable names.



5. You can use numeric variables to do calculations, by providing the Amstrad with a *formula* that tells it how to work things out (e.g. LET models = cars + trains + planes or LET area = length \* width)

6. If you assign a new value to a variable (i.e. 'put a new number in the memory box') the old value is lost (e.g., LET cars = 5 followed by LET cars = 10 gives the answer 10 when you type PRINT cars).

7. You can leave out the word LET in Amstrad BASIC, so instead of typing LET cars = 5, you can type just cars = 5 to get the same effect. Beginners often prefer to use the word LET to help remind them what the instruction means.



8. Although you can save typing and memory space by using single letters for variable names (e.g., **c=5: d=7: h=3: p=c+d+h**), it is far easier to understand what is happening in a program if you use descriptive words. For example, would you have guessed that the previous program meant LET cats = 5: LET dogs = 7: LET hamsters = 3: LET pets = cats + dogs + hamsters?



## Chapter 4

# Questions and answers

If you managed to struggle through the last chapter on numeric variables, and got the *Mindbender* questions right (or nearly right) –well done! That was probably the most difficult chapter in the book.

From here on, it's all plain sailing (well, nearly all!) and much more fun.

### Strings of things

As well as remembering numbers, the Amstrad can remember letters, symbols, words and sentences. You store these in 'memory boxes' in much the same way as you store numbers. But instead of typing, say:

```
LET cars = 5
```

which would put the number 5 into a memory box labelled 'cars', you type

```
LET name$ = "Brian"
```

which puts the word 'Brian' into a memory box labelled 'name\$' (pronounced 'name-string', or, if you prefer, 'name-dollar').

Now when you type

```
PRINT name$
```

the Amstrad responds with the reply:

```
Brian
```

Note that the quote marks have disappeared!

The only differences when you want to store letters or symbols are that:

a) The variable name must end with a dollar sign `<$>` – which you'll find on key 4. This tells the computer to store the information in an area of memory which is reserved for *string* variables. This is different to the area where it stores *numeric* variables.

b) The characters that you want to store must be enclosed in double quote marks, and must not be more than 255 characters long.

### String variable names

You can choose almost any name for a string variable, provided it won't be confused with a word in BASIC and ends with a dollar sign (\$).

For example:

name\$, n\$, friend\$, f2\$, answer\$.

You can even store numbers in a string (but, unlike numeric variables, you can't use string variables in calculations) e.g.:

```
LET address$="10 Railway Cuttings,  
East Cheam"
```

To get the hang of it, let's try a simple program:

```
10 LET name$ = "Brian Jones"  
20 LET age$ = " is 16 years old"  
30 LET message$ = " and owns an  
Amstrad"  
40 CLS  
50 PRINT name$  
60 PRINT age$  
70 PRINT message$  
80 END
```

Note that there is a blank space after the first set of quotation marks in lines 20 and 30. Remember to press <ENTER> after each program line, then type RUN <ENTER> to make the program work.

Alter lines 10 and 20 to make the computer print your own name and age.

### Adding strings together

When you are happy that the program is working correctly, LIST it, then type:



```
60 <ENTER>
```

followed by

```
LIST <ENTER>
```

Line 60 should now have disappeared. Now type a replacement for line 50, like this:

```
50 PRINT name$+age$ <ENTER>
```

and LIST the program again to check that the new line has replaced the old line 50. Then RUN the revised program.

The name and age messages should now appear on the same line, and you should see

now why a space was needed after the first set of quote marks in line 20. Adding strings together like this is called *concatenation*.

Test what happens if you delete line 70 from the program by typing

```
70 <ENTER>
```

and replace line 50 with

```
50 PRINT name$+age$+message$
```

#### Mindbender 4.1

Why does this program give a strange result?

```
10 car$="5"  
20 doll$="7"  
30 toy$=doll$+car$  
40 PRINT "total number of toys  
  = ";toy$  
50 END
```

How can you change this program to make it work correctly? If you can, go straight to the top of the class! If not, the answer is at the back of the book.

HINT: should we be using string variables or numeric variables?

#### Remember

1. Variables that store numbers are called *numeric variables* and variables that store strings of characters are called *string variables*.
2. To distinguish a string variable from a numeric variable, the name of the string variable must end with a *dollar sign* (\$) and the information to be stored in memory must be enclosed in *double quotation marks*.
3. As with numeric variables, you must not start the name with letters that could be confused with the start of BASIC keywords. It is therefore better to use *small letters* for all variable names.
4. The string of characters typed between the quotation marks can be letters, words, spaces, symbols, figures or even nothing at all! Empty strings like this:  

```
10 LET empty$ = ""
```

can be quite useful in certain programs.
5. You can store up to 255 characters in a string variable.

6. You can have numbers as well as letters and words in a string variable, but you cannot perform calculations on these numbers. If you want to do calculations, store the number in a numeric variable (i.e., without the dollar sign).

7. You can add strings together, using the + (plus) sign. This is called concatenation, e.g.

```
LET A$ = "Hello"  
LET name$ = "DAVID"  
LET space$ = " "  
LET D$ = A$ + space$ + name$
```

when you type PRINT D\$ <ENTER>, you will find it contains 'Hello DAVID'



## Chapter 5

# How to INPUT information

It would be rather restrictive if we could only put information into variables from inside a program, using LET statements.

In the BASIC language, numbers and words can be placed into variables from outside of the program, by using the INPUT command. This allows information to be typed in while the program is running.

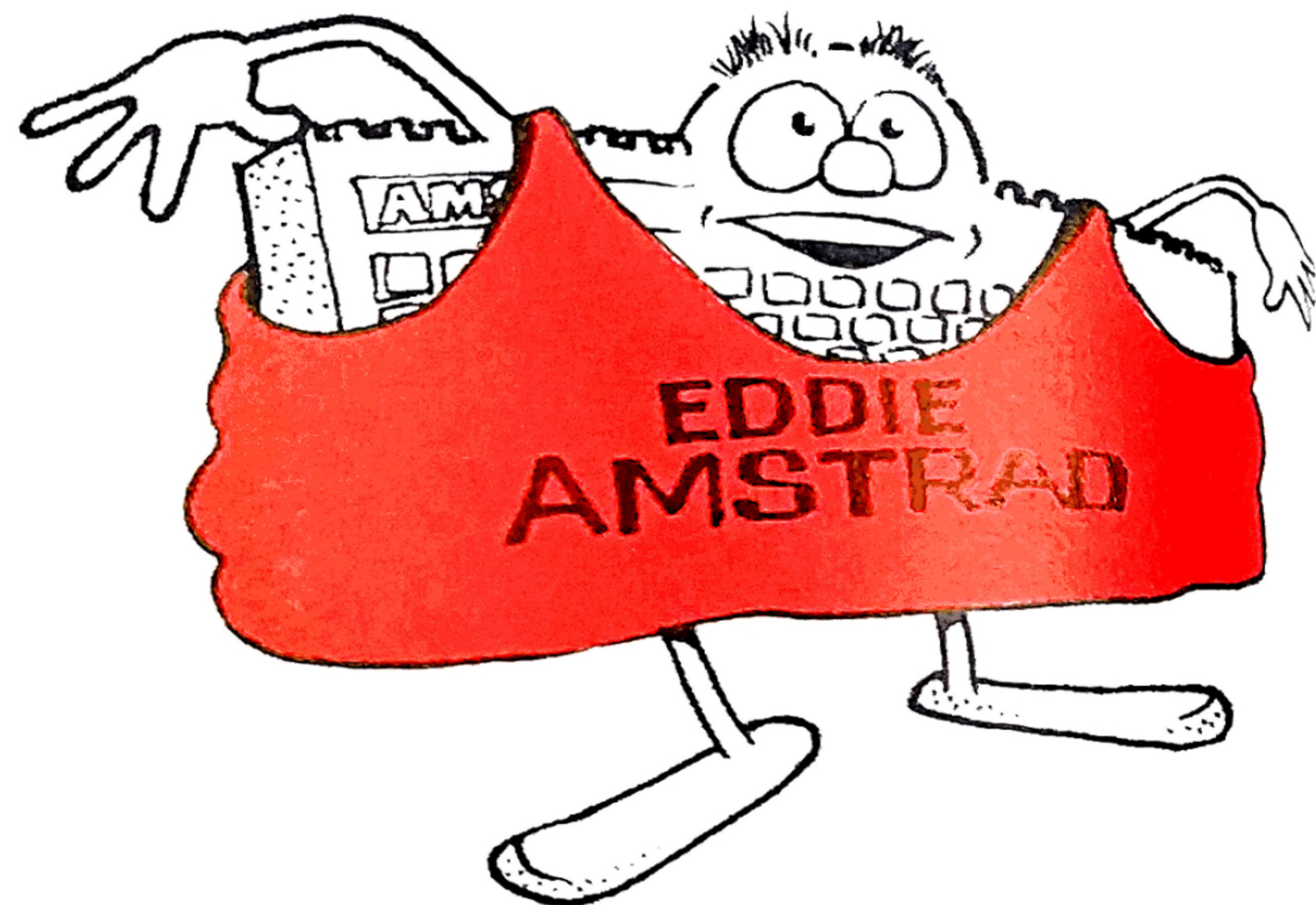
The best way to understand how INPUT works is to use it. Type

**NEW < ENTER >**

then try this:

```
10 LET computer$=" Eddy Amstrad"  
20 CLS  
30 PRINT "Please type in your name"  
40 INPUT name$  
50 PRINT "Hello ";name$  
60 PRINT  
70 PRINT "My name is ";computer$  
80 END
```

Try to guess what the program will do, then RUN it to see if you were right.



### How it works

Line 10 puts the words 'Eddy Amstrad' into a memory box labelled 'computer\$'. (Actually, it stores each letter in a series of consecutive memory locations, but you needn't let this worry you at present!). The dollar sign (\$) at the end of the variable name, tells the Amstrad that it is to store characters rather than numbers.

Line 20 simply CLears the Screen.

Line 30 prints a message on the screen,

asking you to type something in.

Line 40 uses the INPUT command to stop the program and wait for something to be typed in. A question mark appears on the screen to remind you that it is waiting for you to give it some information.

When you have finished typing, you have to press the ENTER key. The Amstrad then stores the information you have typed in a memory box labelled 'name\$' and carries on to the next program instruction – in this case, line 50.

Line 50 prints the word 'Hello', after which the Amstrad finds a semi-colon. This tells it to print the next item on the same line as the last.

But what is it that we want printing? 'Aha!', it thinks, 'the programmer wants me to print the contents of the memory box labelled name\$!'. So it prints whatever you typed in at the INPUT line.

Line 60 simply prints a *blank line*, which helps to improve the screen display. You can leave this line out or delete it if you prefer. Or you could add another blank line by adding:

**65 PRINT**

to the program.

Line 70 works in exactly the same way as line 50, but this time the message is different

and, instead of printing 'name\$' after the message, we now print the contents of the 'memory box' labelled 'computer\$'. You told the computer what to put in this box in Line 10.

Line 80 tells the program to END, so to make it RUN again, you need to type RUN <ENTER>.

### **Making changes**

Try changing line 70 to read:

**70 GOTO 30**

and see what happens. Type different names in when the computer asks you to.

You will need to press

**<ESC>**

to stop the program.

What would happen if you changed line 70 to read

**70 GOTO 20**

or

**70 GOTO 10**

If you can't work it out, try it and see! This produces a classic example of a bug in the program!

### Saving typing

There is a neat little trick you can do with INPUT to save yourself a lot of typing.

Take a look at this next program, but don't bother to type it in yet. It works the same way as the last program, but this time it asks you to input a *number* instead of a string of letters:

```
10 CLS
20 PRINT "How old are you?"
30 INPUT age
40 PRINT "You are ";age;" years old"
50 END
```

When the computer gets to line 30 it waits for you to type in a *number*. (The Amstrad knows it wants a number because there is no dollar sign (\$) after the variable name). Then it carries on with the rest of the program.

What happens if you type in *letters* instead of a number?

Line 40 looks a bit complicated, but is really quite easy. You'll see what it does later, if you can't work it out.

There is a neater way of writing this program, using INPUT to PRINT a message as well as waiting for information. This method joins lines 20 and 30 together. Type in and RUN this next program and see if you can spot the differences.

```
10 CLS
20 INPUT "How old are you ";age
```

```
30 PRINT "You are ";age;" years old"
40 END
```

Line 20 PRINTs the question as well as waiting for you to type a *number* to be stored in the numeric variable 'age'. This is a much better method than the first one you tried.

Line 30 tells the Amstrad to PRINT the words enclosed in quote marks ("You are"), followed *immediately on the same line* (that's what the semi-colon means) by whatever it finds in the *numeric variable* memory box labelled 'age'.

After printing the value of 'age', it comes across another semi-colon that tells it to continue printing on the same line. So it prints the next item enclosed in quote marks (" years old"), before carrying on with the next program line.

Note that there is a space after the word 'are' and another space before the word 'years' in line 30. What would happen if you were to miss out these spaces?

Line 40 – you know what END does! How could you change this line to make the program repeat itself?

### Pausing a program

INPUT can also be used to halt a program or game for purposes other than the input of information. For instance, you can use it to wait until someone has read the instructions before playing a game.



Here's a very simple example, based on the previous program. Note that this time, we are able to go back to line 10 – and clear the screen – which wouldn't be possible without this use of INPUT.

LIST the previous program, to check that it is still in memory, then type in just lines 40, 50 and 60 to make the program look like this:

```
10 CLS
20 INPUT "How old are you ";age
30 PRINT "You are ";age;" years old"
40 PRINT:PRINT
50 INPUT "PRESS ENTER KEY TO
CONTINUE", A$
60 GOTO 10
```

### Multi-statement lines

Did you notice that line 40 contains *two* PRINT instructions – separated by a *full colon*?

The Amstrad allows you to put several instructions on the same line, as long as you separate them with full colons (:), and provided the total length of the program line does not exceed 255 characters (including spaces).

Multiple statement program lines save memory space (and typing!) and help increase the speed at which BASIC programs work. This is important when you start writing arcade games.

### Question mark suppression

Did you notice anything unusual about line 50?

No, not the fact that everything is in capital letters – that is just for appearance. The important tiny detail is that we typed in a comma instead of a *semi-colon* before the variable name. This might seem a trivial difference, but its effect is quite interesting. It stops the question mark appearing!

If we had used a semi-colon instead of a comma between the second set of quotes and the variable A\$, a question mark would have appeared automatically at the end of the INPUT message. While this would not have done any harm, it would have looked a bit silly as we were using INPUT here to give an instruction – not to ask a question.



### Dummy variables

Another odd thing about line 50 is that we are not actually using the string variable A\$ to store information in as we normally would. So, in this instance, we call A\$ a '*dummy string variable*'. We are using it merely to check that the ENTER key has been pressed.

Though we named it A\$ – short for Answer\$ – you could just as easily have called it **answer\$**, or **reply\$** or **RS\$**.

After you have pressed <ENTER>, line 60 sends the program back to *clear* the screen and start again.

If we had not included INPUT to wait for a keypress in line 50, this would all happen so quickly that you might not have time to see anything! Test this by deleting line 50 and RUNning the program again.

### Number cruncher

Let's have a bit of fun with the *name* and *age* programs. Type

```
10 INPUT "Please type in your name ";
   name$
20 CLS: PRINT "Hello, "; name$
30 INPUT "Now tell me how old you
   are "; age
40 CLS
50 PRINT "My goodness, "; age; " years
   old!"
60 PRINT: PRINT "You are much older
   than ME..."
70 PRINT "I was only invented in
   1984!"
80 LET years = 50 - age
90 PRINT: PRINT "Do you realise
   that in only "; years; " years,"
100 PRINT "time you will be 50
   years old???!!!!"
```

RUN this program, and make sure that it

works, so far. Then be sure to read the next chapter to make sure you can SAVE it on cassette before switching the Amstrad off. We are going to add some more program lines to it later.

### Mindbender 5.1

Now for the 500 bonus points question. Using two additional program lines, how could you make the program print out how many *days* old you are?

HINT – there are 365 days in a year. (Don't worry about leap years).

– Your first line (110) should be a LET statement.

– Your second line (120) should be a PRINT statement.

The answer will be given in *Chapter 7*.

### Important

Is your Amstrad printing the messages too high on the TV screen? If so, add a couple of PRINT statements to the beginning of the program and after each CLS command like this:

```
8 PRINT: PRINT
20 CLS: PRINT: PRINT: PRINT "Hello,
   "; name$
40 CLS: PRINT: PRINT
```

## Remember

1. INPUT stops the program to allow information to be entered from the keyboard while the program is running. It waits for the user to type a number or a reply.
2. After typing in the information, you must press <ENTER> to signal the end of the INPUT.
3. INPUT must be followed either by a variable name, or by a *prompt message* followed by the variable name, e.g.

```
INPUT age
INPUT "Please type in your
age";age
INPUT name$
INPUT "What is your
name";name$
```

4. Use a *numeric* variable name (e.g. age) if you wish to INPUT a *number*. Use a *string* variable name (e.g. name\$) if you wish to input letters, symbols or words.
5. The semi-colon after the prompt message is optional (You can leave it out to save a byte of memory!).
6. A question mark (?) is included automatically at the end of a prompt,

provided you use a semi-colon after the closing quote marks.

7. If you type the comma after the closing quotes, the question mark will not appear. This can be useful when you are using INPUT with a dummy variable to halt a program under user-control, for example, while instructions are being read, e.g.

```
50 INPUT "PRESS <ENTER>
TO CONTINUE", R$
```

8. A *dummy variable* is one which you are not going to use to store information in. In the above example, the dummy string variable R\$ is being used purely to test whether <ENTER> has been pressed.
  9. To save memory and speed up program execution, you can include more than one instruction under the same program line number, provided you separate the different instructions with full colons (:), e.g.
- ```
10 CLS:INPUT "How
old are you";age
```
10. Program lines must not contain more than 255 characters including spaces.

# Chapter 6

## SAVE-ing and LOADing your programs



Now that you are starting to type in longer programs, it's time to learn how to save them on cassette tape, so that you don't have to type them in again each time you switch the Amstrad on.

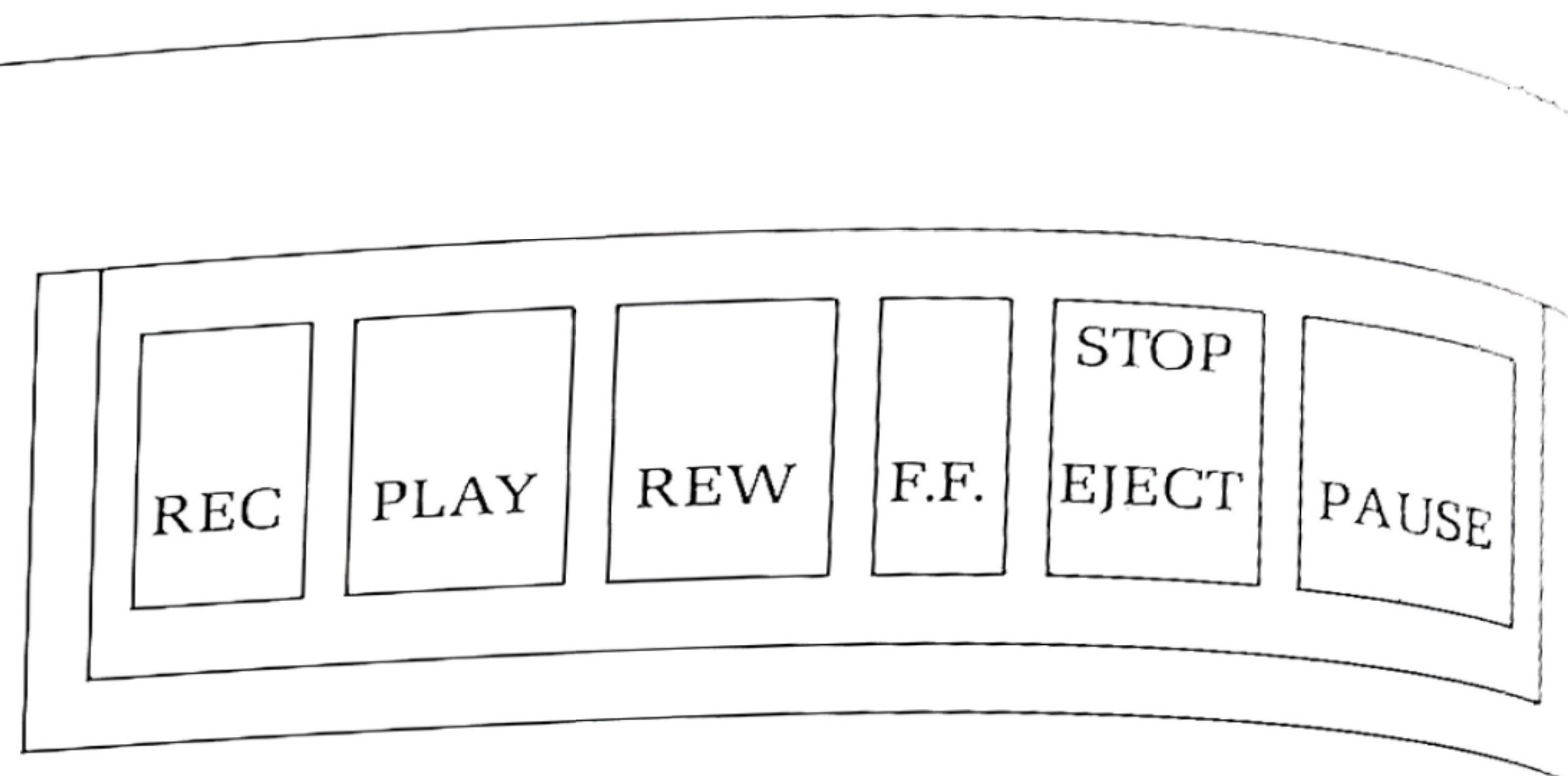
SAVE-ing and LOADing programs may seem a bit tricky at first, but really it's very easy.

### Cassette Recorder

At the right hand of the keyboard you will find the built-in Cassette recorder, the integral part of the Amstrad computer.

The recorder is essentially the same as found in any audio cassette recorder except that the Amstrad recorder offers two speeds for recording and reading programs. The Supersafe speed writes at 1000 baud (bits per second) and Speedload works twice as fast at 2000 baud.

The control keys are the same as found on the normal cassette recorder. You will find that these control keys need to be pressed considerably harder than the keys on the main keyboard.



[REC] = [REC]ord key pressed together with [PLAY] key will record data when either instructed by the program in the memory or by a SAVE command entered from the keyboard.

[PLAY] = [PLAY] will read the data when instructed by the program or read a program from the cassette into the memory by a direct command "LOAD" entered from the keyboard.

[REW] = [REW]ind will rewind the tape to the beginning.

[F.F.] = Fast Forward will advance the tape rapidly.

[STOP/EJECT] = When this key is pressed, the cassette will stop running. When this key is pressed the second time then the cassette lid



will open and allow you to remove or insert a cassette tape as required.

[PAUSE] = This key should not be used when recording or reading programs, as this could corrupt the program.

### Cassette Tape

Normal good quality audio cassettes usually work perfectly well with the Amstrad but use the shortest cassettes you can find.

It is much better to use a C10, C12 or C15 tape or C30 tape at the most than a C90 tape because this puts less strain on your recorder driving mechanism. It is also easier to find programs on short cassettes. *Don't* use C120 cassettes.

Try to avoid using cheap 'bargain price' cassettes of the type sold by garages. These

often have poor recording characteristics and so may not record your programs correctly. Tape that looks dark brown (e.g. super ferric) often records better than light brown (e.g. ferric) tape.

### SAVE-ing programs

Follow these instructions carefully.

1. Type LIST <ENTER> to check that there is a program in the Amstrad's memory. If not, type in a short program – any of the programs in this book will do.
2. You need to tell the computer if you wish to use the Speedload (2000 baud) rate for recording or the computer will default to the Supersafe speed.

For Speedload rate type:

**SPEED WRITE 1 <ENTER>**

The computer will automatically read the rate at which the program was saved on the tape originally.

3. Think of a name of not more than 16 characters for the program.
4. Place a blank tape in the recorder.
5. Rewind the tape to the beginning.
6. When the tape is fully rewound, type

**SAVE "progame" <ENTER>**

The Amstrad will respond with the message

**Press REC and PLAY then any key**

Instead of "progname", you can call the program anything you like in small or capital letters, provided the title is not more than 16 characters long and is enclosed in double quote marks

7. Now press PLAY and RECORD on the cassette recorder, count slowly to twelve - to make sure you are past the plastic leader on the tape - then press

**ENTER**

8. After a moment, the program name should appear on the screen with some numbers and letters.

The computer will tell you when it has finished SAVE-ing by showing the prompt and cursor (■).

9. Before pressing the STOP button on your cassette recorder, SAVE the program again for safety, perhaps with a slightly different name, for example:

**SAVE"progname2" < ENTER >**

Make sure that the RECORD button is still depressed on your recorder and wait a few seconds before you press <ENTER> for the second time in response to the screen message

**Press REC and PLAY then any key**

to make a short gap between the two recordings

### Verifying the SAVE

To check that your program has SAVED properly, press the REWIND key on your recorder, then type

**CAT - ENTER**

When the tape has rewound, press the PLAY button and watch what happens. After a short time the program name should appear on the screen, together with some numbers and perhaps letters, like this

```
progname      block1      $ OK
progname2     block1      $ OK
```

This tells you that both copies of the program have saved correctly, while leaving the original program in memory.

If an ERROR MESSAGE should appear then *stop* the tape, rewind and SAVE the program again.

If both copies of the program are OK, then press the ESCAPE key to get the cursor back.

The CAT command is very useful when you have forgotten the names of programs recorded on a particular tape.

## Loading programs

On the Amstrad, there are two ways to LOAD a program from tape. The LOAD "programe" command simply loads the program into memory so you can LIST, alter or RUN it. Alternatively, you can LOAD and RUN the program automatically, using the CHAIN "programe" command.

Let's try using LOAD first.

1. Switch your computer OFF then ON. Now it has a blank memory.

2. Place the tape, with the program you have just recorded on it, into the recorder. Make sure that the tape is rewound to the correct position, rewind the tape completely if you started at the beginning.

3. Type the following instruction:

**LOAD"programe" < ENTER >**

(where "programe" is the name you called your program when you saved it). You only need to specify the name if you want the computer to load a particular program. If you just want the next program recorded on the tape, you can type:

**LOAD"" < ENTER >**

making sure that there are no letters or spaces between the two sets of quote marks.

4. The message

**Press PLAY then any key:**

should appear on the screen.

5. Press the PLAY button on your cassette recorder and then press < ENTER >.

6. When the computer finds the program, the word

**Loading**

appears, followed by the program name and block number.

7. When the computer has successfully LOADED the program, it will show the prompt and cursor.

You can now LIST and alter the program or RUN it, in just the same way as if you had typed it in.

## CHAINing programs

CHAIN"programe" LOADs and automatically RUNs the program.

To use it, follow the same procedure as above, but instead of typing

**LOAD"programe" < ENTER >**

type

**CHAIN"programe" < ENTER >**

or just

**CHAIN"" <ENTER>**

which LOADs and RUNs the next program on the tape.

### Remember

1. The command SAVE"programe" is used to SAVE a named program.
2. The program name must not be more than **16** letters long and can contain spaces or symbols, e.g.  
**SAVE"programe"**  
**SAVE"prog - name"**
3. Be careful not to SAVE programs on top of other programs already recorded on a tape, unless you want to lose the programs you are taping over.
4. Be sure to wind past the plastic leader before you start to SAVE a program.

5. LOAD"" (with no space between the quote marks) tells the computer to LOAD the first program it comes to on the tape.
6. The command LOAD"programe" is used to LOAD a particular program. The computer will continue searching through the tape until it finds the program you have asked for.
7. CHAIN"programe" will LOAD and then automatically RUN a named program. If the program name is missing, i.e. CHAIN"", the computer CHAINs the first program it finds on the tape.
8. Keep a note of what is on your tapes by noting the number on the tape counter.
9. CAT will check that your program has saved correctly and tell you what programs are on a tape. After verifying a tape, press <ESC> to get the prompt and cursor back.





## Chapter 7

# REM, RENUMBER and EDITing

At the end of Chapter 5 we left you with a problem. How do you make this program print out how many days old you are?

```
10 INPUT "Please type in your name  
";name$  
20 CLS:PRINT"Hello, ";name$  
30 INPUT"Now tell me how old you  
are ";age  
40 CLS  
50 PRINT"My goodness, ";age;" years  
old!"  
60 PRINT:PRINT"You are much older  
than ME"  
70 PRINT"I was only invented in  
1984!"  
80 LET years = 50 - age  
90 PRINT:PRINT"Do you realise that  
in only ";years;" years"  
100 PRINT"time, you will be 50 years  
old???!!!!!"
```

Hopefully, you will have SAVED this program on tape, so you can now load it in from cassette and LIST it.

From the hints we gave you, you know that you need to add two lines. The first

should be a LET statement, multiplying the numeric variable 'years' by 365 (the number of days in a year) – and putting this number into a new memory box, which we'll call 'days':

```
110 LET days = age * 365
```

Ignoring leap years, this would mean that for someone who was exactly 10 years old (to make the maths easy!):

```
days = 10 * 365
```

therefore:

```
days = 3650
```

In other words, on your tenth birthday, you would already have lived for 3650 days!

All we need now, is one extra line to PRINT this information on the screen:

```
120 PRINT:PRINT"You have already  
lived for ";days;" days"
```

You can use the same, easy trick to convert days into hours – bearing in mind that

there are 24 hours in each day:

130 LET hours = days \* 24  
140 PRINT "which equals an  
amazing ";hours;" hours"

### Mindbender 7.1

How would you extend the program by four more lines to make it print out the number of *minutes* and the number of *seconds* you have lived?

HINTS: there are 60 minutes in an hour  
there are 60 seconds in a minute.

To make it a bit more interesting, you could use the phrases in your PRINT statements, like "or a staggering" and "which means a mind-boggling".

Our suggestion is at the back of the book, but try to work out your own solution first. It's easier than you think!

### REMARKS

Now that you are typing in longer programs and (we hope) SAVE-ing your programs on cassette tape, it's time to introduce the BASIC programmer's best friend, the REM statement.

Add this line to your Age program, then type LIST, 50 to see what happens:

```
5 REM ***** AGE ***** <ENTER>
```



REM – which is short for REMark – allows you to add titles and remarks to a program listing that REMind you what it is about.

You can put REM statements anywhere in the program, not just at the beginning. Try these extra lines:

```
25 REM * Find out age * <ENTER>  
105 REM * Work out how many  
days old * <ENTER>
```

Note that the stars aren't really necessary. We put them in to make the REM lines easier to spot.

You can also add a REM statement to the

end of a program line, like this:

```
130 LET hours = days * 24 : REM  
calculate the number of hours
```

provided you remember to separate the two statements with a full colon (:).

The computer ignores everything on a line after it comes to a REM statement and carries on with the next program line. So the following line would not have any effect:

```
6 REM clear the screen:CLS
```

but this one would (please type it in!):

```
6 CLS:REM clear the screen
```

The reason that the first version would not work is that when the Amstrad reaches line 6 it finds a REM statement, so it carries straight on to the next program line – not realising that there is a CLS instruction at the end of line 6. But if you put the CLS first, it carries that instruction out *before* meeting the REM statement and moving to the next program line.

### RENUM(ber)

Although our Age program should now work quite well, because we have added so many extra lines, the line numbers look a bit of a mess, e.g.

```
5 REM ***** AGE *****  
6 CLS:REM clear the screen
```

```
10 INPUT "Please type in your name "  
;name$  
20 CLS:PRINT"Hello, ";name$  
25 REM * Find out age *  
30 INPUT"Now tell me how old you  
are " age
```

Never mind! Your Amstrad has another magical command to sort this out – just type

```
RENUM <ENTER>
```

After a moment or two, the prompt and cursor should reappear. So type

```
LIST, 60 <ENTER>
```

and your program should now look like this:

```
10 REM ***** AGE *****  
20 CLS:REM clear the screen  
30 INPUT "Please type in your name "  
;name$  
40 CLS:PRINT"Hello, ";name$  
50 REM * Find out age *  
60 INPUT"Now tell me how old you  
are " age
```

and so on. The RENUM command has renumbered the program, so that it not only looks more 'professional', but also leaves you more room to add additional lines if you want to.

If, instead of typing RENUM <ENTER>, you were to type

**RENUM 20,5 < ENTER >**

Your program will be renumbered starting at line 20 (instead of the normal 10) and going up in steps of 5. Try it. Then use RENUM < ENTER > to get it back to normal.

Before you go any further, be sure to SAVE the latest version of your Age program.

### Holiday exchange

Now for some more Amstrad magic, that should save you a lot of typing. Type

**NEW < ENTER >**

then type in this new program, which could be handy if you are going to Spain for your holidays:

```
10 REM *** exchange ***
20 LET rate = 220
30 INPUT "How many pounds do you
  wish to change "; pounds
40 LET pesetas = pounds * rate
50 PRINT "For "; pounds; " pounds
  sterling"
60 PRINT "You will receive approx.
  ;pesetas; " pesetas in Spain"
70 PRINT "at the exchange rate of "
  ;rate; " pesetas"
80 PRINT "per pound sterling."
```

Now RUN the program and make sure it works properly. If it doesn't check for typing errors!

### The EDIT keys

If you've typed the program in properly, it should work, but it still looks a bit of a mess on the screen!

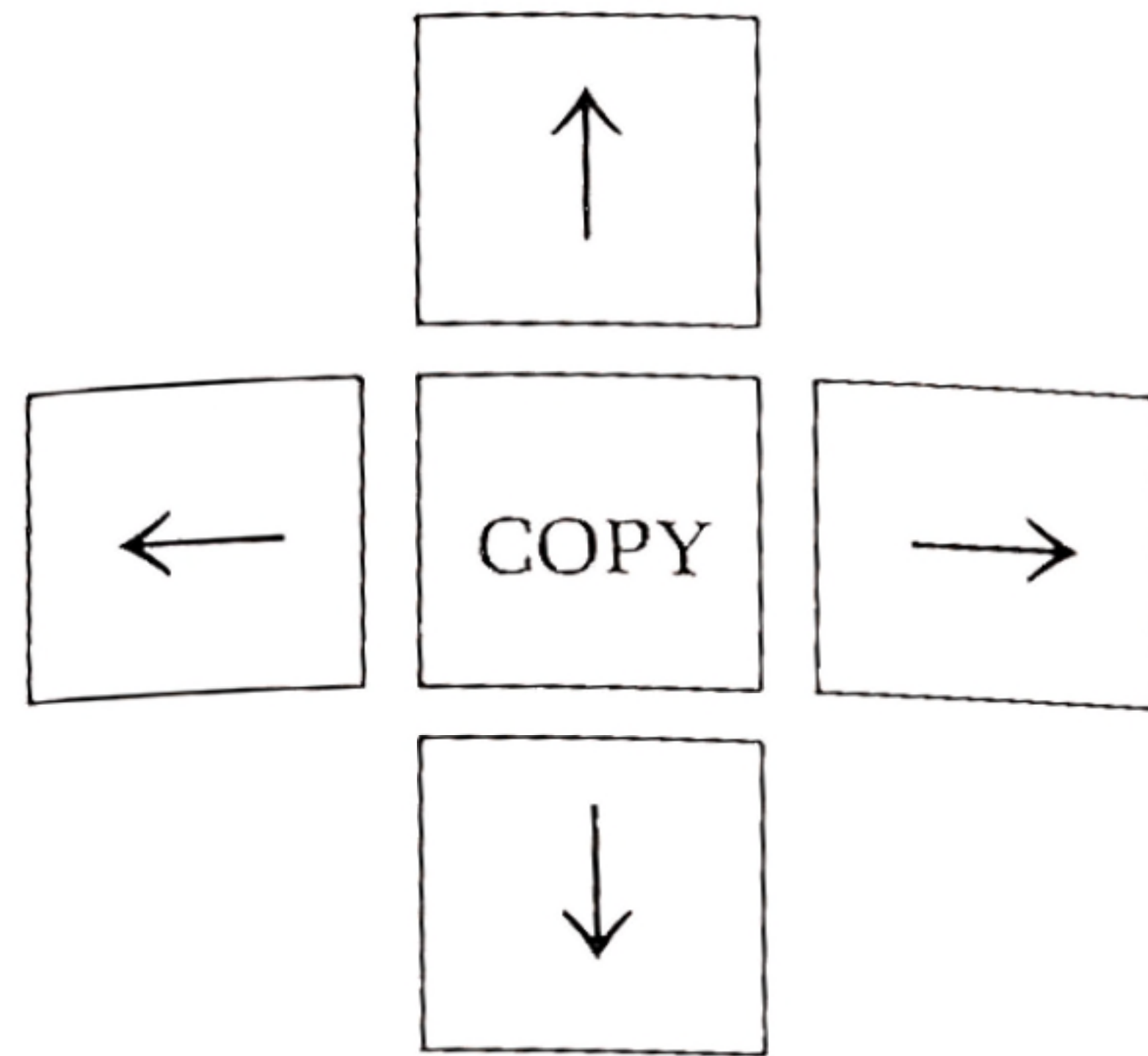
We could improve it a bit, by adding a CLS instruction at the beginning and another before it prints out the information. But this time, instead of adding, say,

**25 CLS < ENTER > and, say,  
45 CLS < ENTER >**

we'll EDIT a couple of CLS commands into the existing lines. Learning how to EDIT is going to save you a tremendous amount of typing later on!

Look at the top right hand corner of your keyboard. Can you see the keys with arrows printed on them? These are the CURSOR CONTROL keys that you use for EDITing programs.





LIST the program and follow these instructions carefully.

1) Hold down the SHIFT key and press the UP ARROW key twice. The 'COPY CURSOR' separates from the *main cursor* (they both look the same). The *copy cursor* is placed over the first character in the line i.e. 8, leaving the *main cursor* behind like this:

```
80 PRINT "Per pound sterling"  
READY
```

2) Continue holding the SHIFT key and pressing the UP ARROW key until the *copy cursor* is level with line 20 of the program, like this:

```
20 LET rate = 220
```

If you accidentally go too far, use the DOWN ARROW key to bring the copy cursor level with line 20.

3) Now use the key marked COPY to move the

cursor to the right hand end of line 20, so it looks like this:

```
20 LET rate = 220 ■
```

As the *copy cursor* moves along the line, notice that a COPY of the line appears at the bottom of the program:

```
80 PRINT "per pound sterling"  
20 LET rate = 220 ■
```

4) When a COPY of the whole line has appeared at the bottom of the program, type a full *colon*, followed by the command CLS.

You should find that while line 20 at the top of the screen remains as it is everything you type appears on the COPY line at the bottom of the program:

```
20 LET rate = 220:CLS ■
```

If you make a mistake, use the DELETE key to rub the wrong letter(s) out and re-type it correctly.

5) When you are sure that line 20 is correct, press

```
< ENTER >
```

and LIST the program again to check that the new line 20 has replaced the old one.

Now make exactly the same alteration to line 40, so that it reads like this:

```
40 LET pesetas = pounds * rate:CLS
```

Easy, isn't it? Now RUN the program again, to make sure that everything is still working OK.

### Inserting

You can use the *cursor control* keys and the COPY key to make practically any alteration you want – inserting, deleting or changing program lines.

Inserting letters or words is just as easy. Suppose you want to *insert* something in the middle of a program line. For example, let's insert the word 'Spanish' before the word 'exchange' in line 10.

First, CLear the Screen for action and list the program again, with

```
CLS < ENTER >  
LIST < ENTER >
```

Now follow these steps:

1) Use the SHIFT and **up arrow** to bring the *copy cursor* up level with line 10

```
10 REM *** exchange ***
```

2) Use the COPY key to move the cursor along the line until it is *in front of* the first 'e' of exchange. The copied line at the *bottom* of the program should now look like this:

```
10 REM *** ■
```

3) Now type the word, 'Spanish'; followed by space. The line at the bottom of the screen should accept each letter typed until it looks like this:

```
10 REM *** Spanish ■
```

Again, if you make a mistake, use the DELETE key and type the letter(s) again correctly.

4) Finally, use the COPY key to copy the rest of the old line 10 until it looks like this:

```
10 REM *** Spanish exchange *** ■
```

and then press < ENTER >. LIST the program to check that everything is as you want it to be, then RUN it again, just to be sure!

### Deleting

Deleting (or 'Editing-out') characters is even easier than inserting. For example, to remove the word 'exchange' from line 10, you would:

1) CLear the Screen and LIST the program.

2) Use the SHIFT and UP ARROW keys to get the *copy cursor* level with line 10:

```
10 REM *** Spanish exchange ***
```

3) Use the COPY key until the *cursor* is on the first 'e' of 'exchange'. The copied line at the bottom should now look like this:

```
10 REM *** Spanish ■
```

4) Now use the SHIFT and **right arrow** keys to move the *copy cursor* to the space following the word 'exchange'. Nothing will have happened on the bottom line, but the old line at the top should now look like this:

```
10 REM *** Spanish exchange ■ ***
```

5) Finally, use the COPY key again, to copy the rest of the line. The line at the bottom should then look like this:

```
10 REM *** Spanish ***■
```

Now press <ENTER> and LIST the program again. You have deleted the word 'exchange'!

Practice using the **cursor control** and COPY keys until you can use them to insert, change and delete characters without having to think about it.

### Improvements

Our screen display still looks a bit untidy. For one thing, the first line is too close to the top edge of your TV screen. So let's add a couple of blank lines before it starts printing:

```
25 PRINT:PRINT < ENTER >  
45 PRINT:PRINT < ENTER >
```

Now use the **cursor control** keys and the COPY key – exactly as you've learned above – to *edit* some of the PRINT lines, so that the messages fit on the screen more neatly:

```
50 PRINT "For ";pounds;" pounds
```

```
sterling, you will"  
60 PRINT "receive approx. ";pesetas;"  
pesetas in Spain"
```

If you can, be sure to SAVE the revised program before doing the Mindbenders.

### Mindbender 7.2

How could you alter this program (using the CURSOR CONTROL keys and COPY key) to make it calculate and print the number of Italian lire you would get for British pound notes?

HINT: At the time of writing, you would get 2395 lire for £1 sterling, so use rate =2395.

Don't forget to SAVE your answer before doing the next Mindbender.

### Mindbender 7.3

If the bank gives 11.9 francs per £1 sterling, how would you alter the program for a holiday in France? No hints this time because it's so easy!

Our suggested answers are given at the back of the book, but you should be able to do these yourself!

## Remember

1. REM enables you to add REMarks to a program, to help you REMember what you are doing. The Amstrad ignores everything else on the program line after it meets a REM statement.
2. You can use REM anywhere in the program, but if you use it in a multi-statement line, be sure to make it the last item in the line – like this:

**50 CLS : REM clear the screen**

3. RENUM (start at), (step) enables you to renumber your program from line number (start at), in increments of (step) lines, e.g.

**RENUM 100,5 < ENTER >**

will renumber the program, starting at line 100 and going up in steps of 5. So the program will start with line 100, then go on 105, 110, 115, 120, etc.

4. RENUM is very useful for 'opening up gaps between lines' when you are adding several more lines in between existing program lines.
5. The four arrow keys on the top right corner of the keyboard are called **cursor control** keys.
6. To *edit* a program, you first LIST the program (or just the line(s) you are interested in). Then use the *up* and *down arrow* keys together with SHIFT key to position the *copy cursor* line on the line you want to *edit*.
7. Use the COPY key to copy the parts of the line you want in the new line. Type in any new characters you want to insert and use the SHIFT and **right arrow** keys to skip over any characters that you don't want.
8. Press < ENTER > to confirm that you have finished editing the line.





## Chapter 8

# Colour MODEs

Before we look at some more important BASIC commands we shall take a break and experiment with some of the modes and colour commands on the Amstrad.

Your Amstrad is able to work in three different MODEs, numbered from 0 to 2. These MODEs affect what you can do on the screen, the colours you can use and what size letters can be printed. Only one mode can be used at any one time.

Up to now, you have been working in MODE 1. This is the default mode – which means the mode you get when you first switch on the computer and you haven't asked for anything in particular.

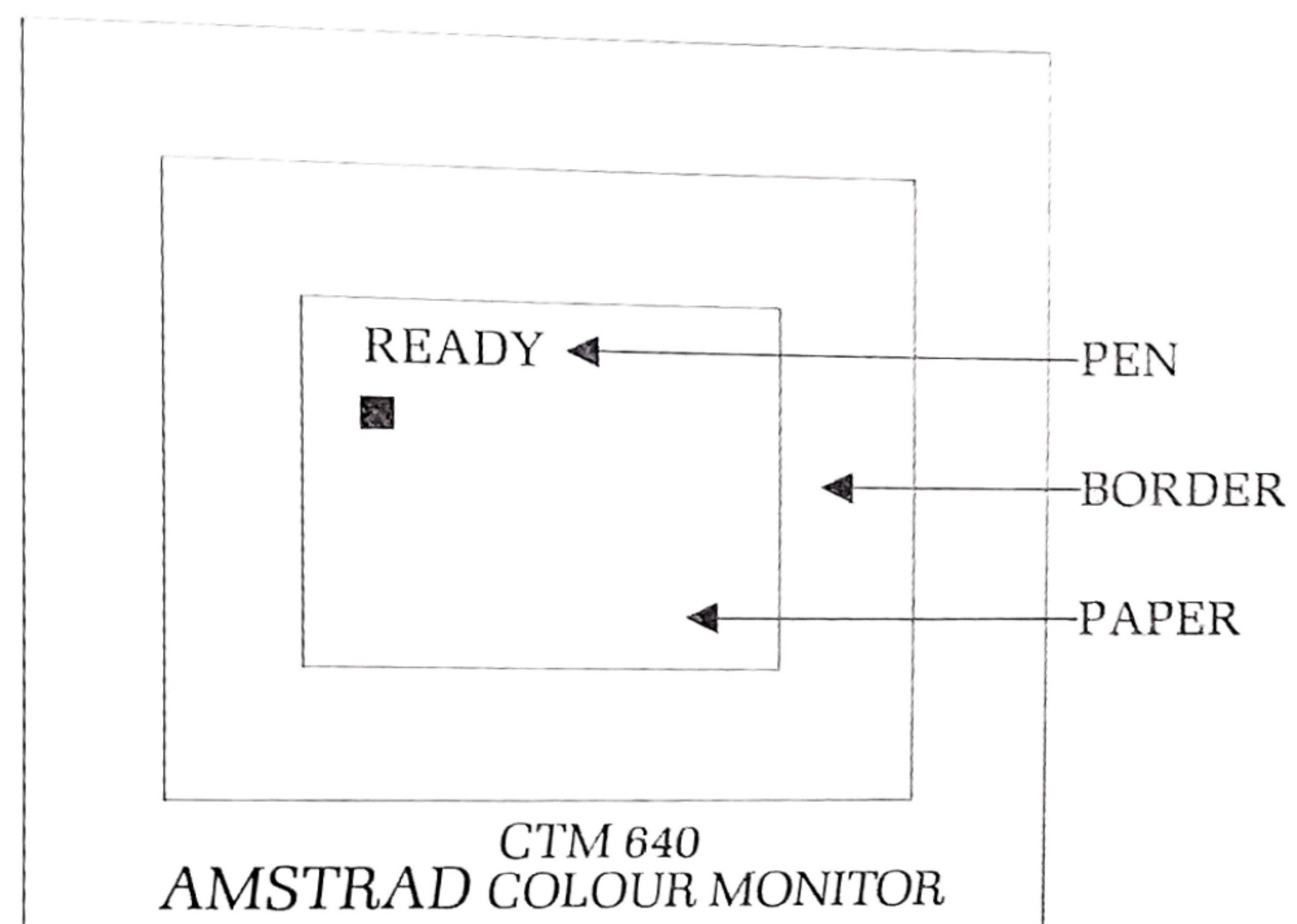
In MODE 1 up to 4 of the 27 colours can be used at any time. This mode uses 'normal' sized letters – 40 of these will fit into one line across the screen and it gives you 25 lines.

In MODE 2 up to 2 of the 27 colours can be used at any time. This mode uses very small letters, so 80 characters will fit across the screen and it gives you 25 lines.

In MODE 0 up to 16 of the 27 colours can be used at any time. This mode uses very big letters, so only 20 characters will fit across the screen and it also gives you 25 lines.

There is a choice of 27 ink colours. See Figure 1 for a complete list of ink colours and the related ink numbers.

You can change the colour of the BORDER, the PAPER or the PEN all independently of each other. The BORDER is the area surrounding the PAPER. The PAPER is the area inside the BORDER where



characters can appear. The PEN is the colour of the characters.

When you first switch on the computer you will find the BORDER and PAPER colours are both blue and characters in yellow (PEN).

You can change the border colour by using the BORDER command and any of the ink colours shown in Figure 1 (page 50).

Now type in:

**BORDER 16 and press < ENTER >**

You will see the border colour is changed from blue to pink. If you refer to Figure 1 you will see that ink number 16 is pink.

You can also make the border colours flash alternatively between the two colours.

Now type in:

**BORDER 16,26 and press < ENTER >**

The ink number 16 is pink and 26 is white, so the computer is flashing pink and white border colours alternatively.

You can change the paper or pen colours by using PAPER or PEN commands. There is a total of sixteen pens/papers available on your Amstrad. They are numbered 0 to 15. These numbers are not the colour numbers shown in Figure 1.

Figure 2 (page 50) shows the matrix of pen/paper numbers and the associated ink (numbers) colours available on each of the three modes. You will find it easier to understand by using the PAPER and PEN commands.

Type in:

**PAPER 2 and press < ENTER >**

If you refer to Figure 2, you'll see 2 under the PAPER/PEN column and if you now read along the same line in MODE 1 column you'll see the ink number is 20. If you now look at Figure 1, the ink colour chart, you will find 20 is bright cyan.

If you clear the screen (CLS command) you'll find the paper colour has changed from blue to bright cyan, leaving the pen (character) colour still in yellow.

Now type in:

**PEN 3 and press < ENTER >**

If you can again refer to Figure 2, you'll see 3 under the paper/pen column and if you read along the same line in MODE 1 column you'll see the ink number is 6. If you now look at Figure 1, the ink (number) colour 6 is bright red.

Type in the program below, to see what the different MODEs are like and to see all the available colours in each of the three modes.

You are still using CLS and NEW, aren't you?

```
10 REM *** MODES & COLOURS ***
20 MODE 1
30 INPUT "Which MODE (0 to 2 only)
   ";choice
40 MODE choice
50 FOR pennumber = 0 TO 15
60 PEN pennumber
70 PRINT "This is PEN number
   ";pennumber;" in MODE";choice
80 PRINT
90 NEXT pennumber
```

Now RUN the program using each of the three modes.

You can see that the letter sizes are different in each mode.

**MODE 1 uses normal sized letters**  
**MODE 2 uses very small letters**  
**MODE 0 uses very big letters**

You should have also seen the available colours under each of the modes.

### How it works

Let's see how the Modes & Colours programs works.

Line 20 – CLears the screen and puts the computer in MODE 1. There is no need for a CLS instruction here, because whenever you change the mode the screen clears automatically.

Line 30 – Waits for you to type in a number. The number is stored in a variable 'choice'.

Line 40 changes the MODE to the number chosen in line 30. Notice how the variable choice is used.

Lines 50 and 90 are part of FOR/NEXT loop, we'll have a look at loops in more detail later on. It feeds numbers from 0 to 15 into a memory box labelled pennumber which is used in line 70 to set the pen colour.

Line 60 – Selects the pen colour.

Line 70 – Prints a message in a colour set by the PEN command in line 60 within a selected MODE.

Line 80 – Prints a blank line.

The 27 colours available are listed in Figure 1, each with their INK reference number.

### Mindbender 8.1

Why did all the lines in MODE 2 print in one colour (i.e. in Yellow)?

And why did it miss out all the even pen numbers i.e. 2,4,6 etc?

HINT: How many ink colours in MODE 2?  
(see Figure 2 then Figure 1.)

### Mindbender 8.2

Try writing your own short program to show the Amstrad colours in MODE 0. See example at the back of the book.

### Remember

1. The Amstrad has 3 screen MODES. These are numbered from 0 to 3.
2. MODE 0 is the only mode which uses

the most colours (16 of the 27 available colours) including the flashing ones.

3. The BORDER command is used to change the border colour to any of the 27 colours shown in Figure 1.
4. The PAPER/PEN Number is used to change either the paper or the pen colours.
5. Changing mode in a program clears the screen – just as if you've typed CLS.

### MAIN COLOUR CHART

| <i>Ink Number</i> | <i>Ink Colour</i> | <i>Ink Number</i> | <i>Ink Colour</i> |
|-------------------|-------------------|-------------------|-------------------|
| 0                 | Black             | 14                | Pastel Blue       |
| 1                 | Blue              | 15                | Orange            |
| 2                 | Bright Blue       | 16                | Pink              |
| 3                 | Red               | 17                | Pastel Magenta    |
| 4                 | Magenta           | 18                | Bright Green      |
| 5                 | Mauve             | 19                | Sea Green         |
| 6                 | Bright Red        | 20                | Bright Cyan       |
| 7                 | Purple            | 21                | Lime Green        |
| 8                 | Bright Magenta    | 22                | Pastel Green      |
| 9                 | Green             | 23                | Pastel Cyan       |
| 10                | Cyan              | 24                | Bright Yellow     |
| 11                | Sky Blue          | 25                | Pastel Yellow     |
| 12                | Yellow            | 26                | Bright White      |
| 13                | White             |                   |                   |

Figure 1: The INK numbers and colours

### INK COLOUR

| <i>Paper/<br/>Pen No.</i> | <i>Mode 0</i>  | <i>Mode 1</i> | <i>Mode 2</i> |
|---------------------------|----------------|---------------|---------------|
| 0                         | 1              | 1             | 1             |
| 1                         | 24             | 24            | 24            |
| 2                         | 20             | 20            | 1             |
| 3                         | 6              | 6             | 24            |
| 4                         | 26             | 1             | 1             |
| 5                         | 0              | 24            | 24            |
| 6                         | 2              | 20            | 1             |
| 7                         | 8              | 6             | 24            |
| 8                         | 10             | 1             | 1             |
| 9                         | 12             | 24            | 24            |
| 10                        | 14             | 20            | 1             |
| 11                        | 16             | 6             | 24            |
| 12                        | 18             | 1             | 1             |
| 13                        | 22             | 24            | 24            |
| 14                        | Flashing 1,24  | 20            | 1             |
| 15                        | Flashing 16,11 | 6             | 24            |

Figure 2: PAPER/PEN/MODE/INK reference



## Chapter 9

# Round and round we go

Now we get back to some simple BASIC work. It is very important that you try to understand this chapter, where we are going to look at loops.

In previous chapters, you have used GOTO to make the computer repeat a section of program. The computer has then continued repeating the program until the ESCAPE key was pressed, twice.

There are better ways to make a program repeat, without using GOTO. In fact, using GOTO in big programs makes them difficult to follow, so try to use GOTO as little as possible.

### FOR/NEXT loops

This first command to help us repeat a section of program uses two special lines which allow you to control how many times you want the section repeated. Instead of using a long-winded way of talking about 'repeating sections' we shall use the word *loop* from now on.

Try this example of a *loop*. Remember to type NEW and CLS. You are used to it by now but might be getting a bit lazy!



```
10 REM *** Loop ***  
20 MODE 1  
30 FOR a=1 TO 20  
40 PRINT "Loop ";a  
50 NEXT a  
60 PRINT "Finished the loop."
```

With a FOR/NEXT loop, you have complete control over the number of times something will happen. This program prints the word 'Loop' (with a number after it) 20 times.

At the very top of the loop – line 30 – the special FOR command is used. At the bottom of the loop – line 50 – the special command NEXT is used. FOR and NEXT always go together, so such loops are called FOR/NEXT loops.

Line 30 really means, 'do the program up to the NEXT line 20 times'. The numeric variable 'a' keeps count of how many times the loop has been done, so it is called the *count variable*. You can see that 'a' is printed in line 40 after the word Loop.

The BASIC word TO is used to tell the computer how many times to do the loop. FOR a=1 TO 50 means, 'do the loop 50 times'. Note also that because computers often start counting at number 0 (zero) instead of 1, the line:

**FOR count = 0 to 50**

means, in effect, 'do this loop 51 times', because there are actually 51 numbers between 0 and 50. So if you wanted it to do 50 loops, starting at 0, you would have to type:

**FOR count = 0 to 49**

### Mindbender 9.1

Now you try writing a program to PRINT out the word 'Hello' and your name 15 times.

HINT: You will need to alter lines 30 and 40 in the above program.

### Times tables

This next short program, prints out the 'times tables'. You choose which table is to be printed from outside the program using our old friend INPUT. Try it first, then we'll have a look at how it works.

```
10 REM *** TIMES TABLES ***
20 MODE 1
30 PRINT "TIMES TABLE PROGRAM"
40 PRINT
50 INPUT "Which 'X times table' do you
   want ", number
60 CLS
70 PRINT "This is the "; number; "
   times table."
80 FOR n=1 TO 12
90 PRINT n; " x "; number; " = ";
   n*number
100 NEXT n
110 PRINT
120 PRINT "Press a key to restart."
130 G$=INKEY$
135 IF G$="" THEN GOTO 130
140 RUN
```

### How it works

This program uses many of the BASIC commands that you have studied so far. Line 50 stores your choice of times table in the numeric variable 'number'. Line 80 tells the computer to do the loop 12 times. You can try making it more than 12 if you like.

Line 90 looks complicated but if you study it carefully you will see that it prints both the sum and its answer. The answer is the 'n \* number' part, 'n' is the loop number and the 'number' is the value you INPUT at line 50.

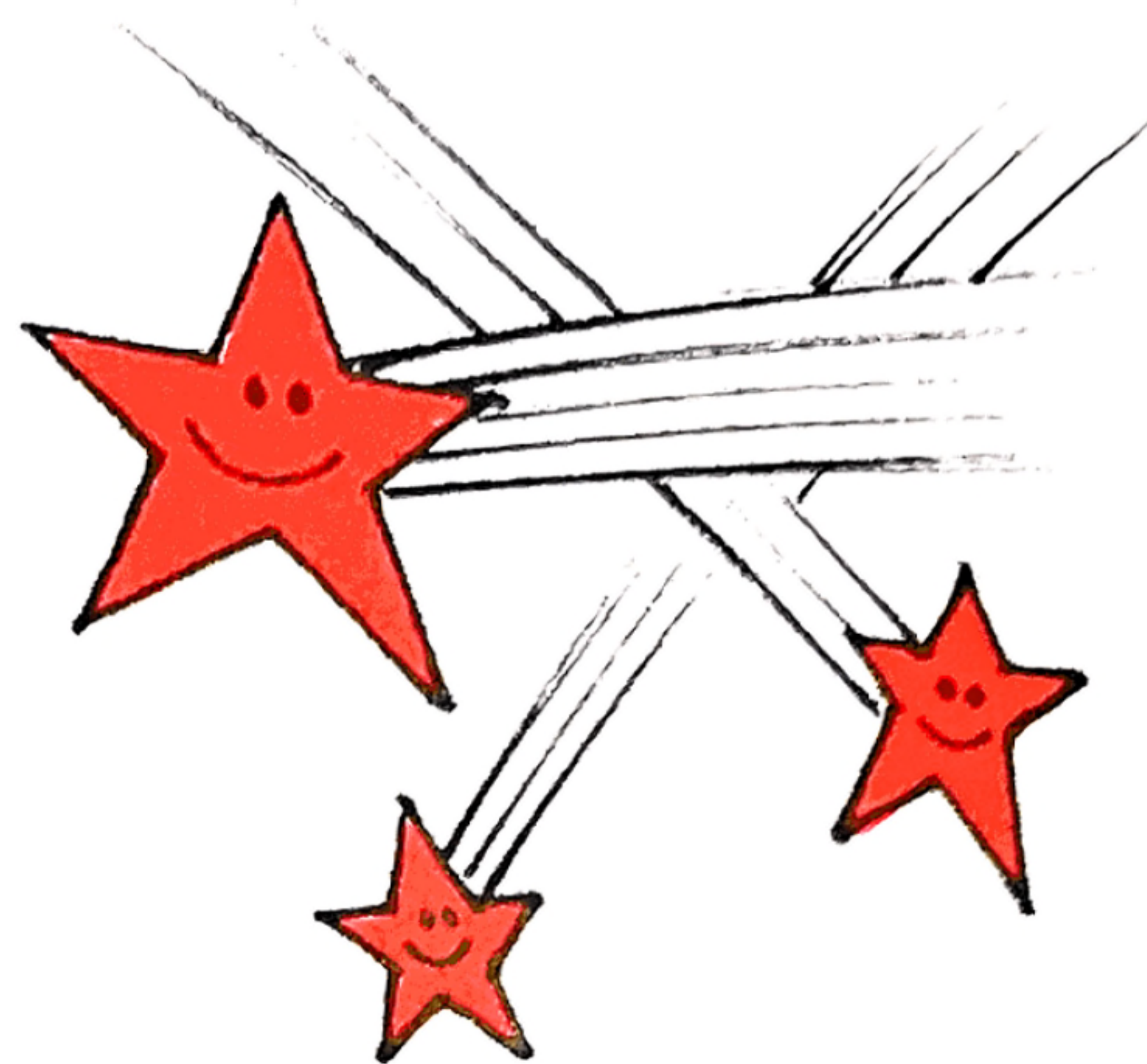
Let's imagine that you had selected the 5 times table (making 'number'=5) and the loop had reached, say, its 6th time round (making n = 6). Line 90 would then first PRINT the number 6 (the value of 'n'), followed by 'x', followed by 5 (the value of 'number'), followed by '=', followed by 30 (which is the result of 'n \* number'). When n = 6 and number = 5 'n \* number' becomes 5 \* 6 – which equals 30!

Next time round, 'n' will become 7 and 'number' stays the same (5). So 'n \* number' will equal 7 \* 5 = 35 – which is what Line 90 will print.

Perhaps you could change the program so that it uses some colour. By now, you should know how to use the PEN/INK commands in the 3 MODES available.

### Shooting stars

This program below uses a loop to move a star across the screen. Try it.



```
10 REM *** Shooting stars ***
20 MODE 1
30 CLS
40 FOR column = 2 TO 40
50 LOCATE column,12
60 PRINT "*"
70 LOCATE column-1,12
80 PRINT ""
90 NEXT column
```

Can you spot how this program works? What happens is that Line 60 prints a star, 12 lines down from the top of the TV screen.

The position of the star across the screen is set by the FOR/TO statement in line 40, which starts by making 'column' = 2, then 3, 4, 5, etc until it reaches 40.

So the value of 'column' in lines 50 and 70 also starts at two, then increases to 3, 4, 5, etc. The effect (in line 60) is that a star is printed

first in column 2, then in columns 3, 4, 5, etc until it reaches column 40 – which is the furthest right-hand column in MODE 1. (In other modes, the maximum column number might be 20 or 80).

Line 80 prints a blank space where the previous star had been printed. If you try leaving line 80 out you will end up with a line of stars printed across the screen. By rubbing out the previous star after the new one has been printed, you get the effect of a star moving across the screen.

### Now you see it ...

The star is rather fast! You can slow it down by putting a delay loop inside the first loop. This goes after the star has been printed and rubbed out from its last position, so add the line:

```
85 FOR delay = 1 TO 100 : NEXT delay
```

This will slow down the star. Experiment with different values in line 85. Instead of 100 you might try 50 then 200 and so on.

Placing one loop inside another like this is OK, but the two loops must not overlap. The second loop must start and finish inside the first.

### Experiments in TIME

When you start programming arcade games, the speed at which things move across the screen will become very important. Although BASIC probably seems very fast at the

moment, in computer terms it is really rather slow!

So you will often need to see if you can find a way of speeding things up. Here's a scientific way of testing whether one method is really faster than another – using the TIME command.

Built into your Amstrad is a pretty accurate timing mechanism which is updated every 300th of a second. It keeps a record of the elapsed time since the computer was switched on.

You can note the time you started and finished an operation within your program. The calculated difference between the two noted times should tell you how many three hundredths of seconds the specific operation took.

Let's try it in your *Shooting stars* program, by adding these four lines:

```
35 STARTTIME = INT(TIME/300)
100 ENDTIME = (TIME/300)
110 ELAPSEDTIME = ENDTIME -
    STARTTIME
120 PRINT ELAPSEDTIME;"seconds"
```

Note that to turn the time into seconds, you need to divide it by 300. The effect is to time how many seconds it takes for the star to move across the screen. RUN the program a couple of times and note the time down.



Now remove the delay loop, by deleting line 85. RUN the program again a few times (the timings may vary fractionally).

Is there any way we could speed the movement up even more? Well, yes – if you don't mind doing a bit more work.

Firstly, in the same way that you don't really have to include the word LET in lines like:

```
35 LET STARTTIME = (TIME/300)
```

(so we didn't bother to include it above) you can also leave out the name of the count variable after a NEXT statement. So see if it makes any difference to the timing if you change line 90 to read just:

```
90 NEXT
```

Is it any faster? In a small FOR/NEXT loop like this one the difference is probably very small, but in a big loop – say from 1 to 1000, it could make an enormous difference.

The other thing worth trying is to use the cursor and copy keys to EDIT line 40 to make it one long multi-statement line, containing the whole of the FOR/NEXT loop, and separating each of the statements with *full colons* (:) like this:

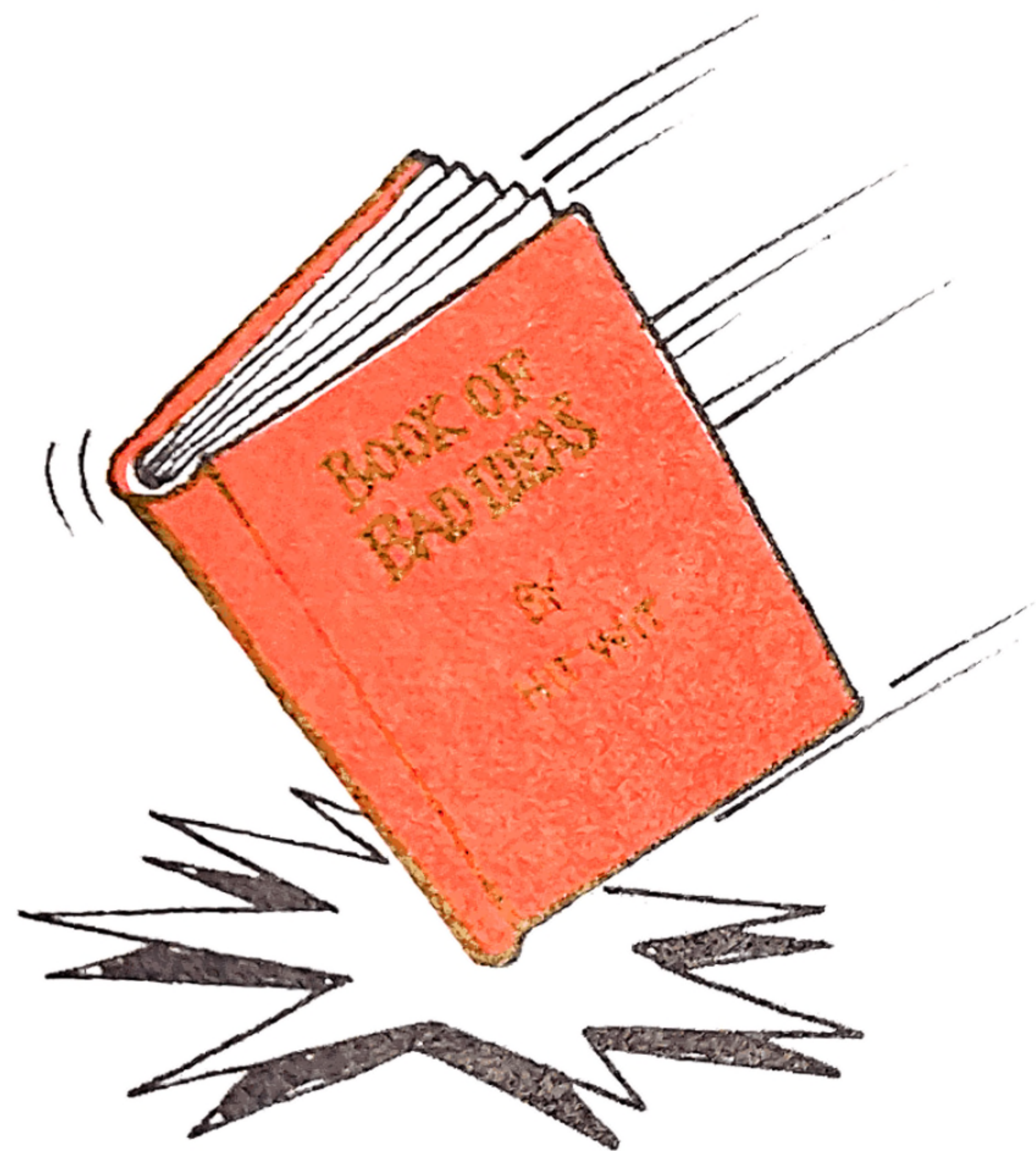
```
40 FOR column = 2 TO 40:  
LOCATE column,12:  
PRINT "*":
```

```
LOCATE column-1,12:  
PRINT ""
```

Don't forget to delete lines 50, 60, 70 and 80 before you RUN the program!

**Warning – 'experts' can be wrong!**

As you progress in computing, you will hear a great deal of nonsense talked by other programmers – and even written in books and magazines – about how you can speed something up by doing one thing or another. Some people's ideas work; others don't.



To be sure – be scientific! Make up a little test program to find out whether it is really worth doing... don't let even the greatest boffins on the Amstrad persuade you that SUBROUTINES are better than GOTOs or

GOSUBs unless you have tested it for yourself.  
Like all adults, they are sometimes wrong!

### WHILE ... WEND

Now, there is another way of making a loop.  
The Amstrad has some commands that many  
other BASICS do not possess.

You may have been asked to do  
something while ...like 'look after the baby  
while I am at the shops...'

Well, the Amstrad can be told to do things  
while a given condition is true, using a  
WHILE/WEND loop. The computer carries on  
doing a loop until a specified condition is met.  
The next program will help you to see what  
we mean.

```
10 REM *** WHILE/WEND loop ***
20 MODE 1
30 a=0
40 WHILE a < 10
50 a=a+1
60 PRINT a
70 WEND
80 PRINT "Finished"
```

Notice that we have stopped using the  
word LET. You can add LET to the beginning  
of lines 30 and 50 if you prefer.

At the beginning of the program the  
variable 'a' is made equal to zero. The  
computer is then asked to repeat all lines  
between the word WHILE and the command  
WEND while 'a' is less than 10.

Line 50 (which is inside this loop) adds 1  
to the value of 'a'. So when the loop starts  
(a=0), it makes a=0+1, which equals 1. The next  
time round, 'a' already equals 1 so it becomes  
a=1+1, which makes it 2. The next time, as it  
already now equals 2, 'a' becomes a=2+1,  
which makes it 3, and so on.

Line 60 simply PRINTs the value of 'a', so  
that you can see what is happening.

Remember – when you use WHILE, it  
must have some condition after it.

Try changing line 40 to –

```
40 WHILE a < 20
```

or any number you fancy.

### Cacophonics

Below is a program that makes a horrid noise  
until you press the ESCAPE key. It uses the  
SOUND command.

Although we will not have the space to  
study the SOUND command in depth in this  
book, we shall use the command to make more  
sounds later in the book.

Be careful not to make any typing  
mistakes.

```
10 MODE 1
20 WHILE TIME > 1000
30 SOUND 1,15,RND(255),5
40 WEND
```

WHILE/WEND loops are very useful because you can make the computer do something until something else happens. This gives you a lot of control over the program.

### **Mindbender 9.2**

How could you alter the *Shooting stars* program, so that it uses a WHILE/WEND loop, instead of FOR/NEXT?

HINT: You will need to combine *Shooting stars* with some of the ideas in the 'WHILE/WEND' program.

### **Mindbender 9.3**

Most 'experts' seem to believe that WHILE/WEND loops are far superior to the humble FOR/NEXT loop. Use the TIME command to test whether your WHILE/WEND loop is in fact faster than the FOR/NEXT version.

Which version of the program would you think uses the least memory? Each character in your program (including spaces) occupies approximately 1 byte of memory.

### **Mindbender 9.4**

Make the computer print the 12 times table. Use a loop.

### **Mindbender 9.5**

Use the loop to print your name 200 times

on the screen, with one space between each name.

### **Mindbender 9.6**

Make the computer count up to 1000 and print every number. Time the computer. How long does it take?

You will find some of the possible answers to these problems at the end of this book.

### **Remember**

1. A program can be made to repeat a section as many times as you want by using a LOOP.
2. One type of loop is the FOR/NEXT loop.
3. The FOR/NEXT loop uses a numeric variable to count how many times it repeats the required program instructions. This is called the *count variable*.
4. The FOR command is the start of the loop. All program lines after the FOR and before the NEXT are repeated.
5. The count variable can be omitted after the NEXT command. The loop runs faster.

6. FOR/NEXT loops normally increase the count variable by 1 each time the loop goes round. But you can make it go up by any other number, by specifying a STEP size e.g. FOR pitch = 1 TO 255 STEP 8.

7. FOR/NEXT loops can be placed inside each other. These are called 'nesting' loops. You must be careful to close the inner loop with a matching NEXT, before you close the outer loop, e.g.

```
10 FOR x = 1 to 10
20 PRINT "outer loop running"
30 FOR y = 1 to 3
40 PRINT "inner loop running"
50 NEXT y,x
```

Note that the inner loop (**y**) has been closed first in the NEXT statement. If you put the letters the other way round, the program would crash. Try it!

8. You can use the TIME command to count the number of three hundredths of a second that have elapsed since you stored the start time.

9. Another type of loop is the WHILE/WEND loop. This makes sure that a section is repeated while a particular condition is true.

10. In a line such as

```
10 SOUND 1,15,200,5
```

the numbers mean:

SOUND channel, Tone Period duration, and Volume, where channel is usually 1 to 7 (see *User Instructions*), volume is normally 15 (loudest) to 0 (7 is half volume), Tone can be from 0 to 4095 (highest note), and duration can be from 1 to 255 in 100th of a second. See Chapter 6 of your *User Instructions* for details.



## Chapter 10

# Decisions . . . decisions

Every day you make decisions. That is, you decide to do something. Computers can also make decisions, which is what makes them so powerful.

When we make a decision we often say that IF something happens, THEN I will act. Computers use the words IF and THEN as a part of the BASIC language. An everyday example might be IF you have permission, THEN you may go out to play. In BASIC the IF/THEN statement means IF something is true THEN do something.

A short program will show you how IF/THEN statements work.

```
10 MODE 1
20 INPUT "Press 1 or 2", number
30 IF number=1 THEN PRINT "Number
  1"
40 IF number=2 THEN PRINT "Number
  2"
50 END
```

This program is very simple and should be easy to understand. Line 20 uses INPUT to wait for a number to be typed. The number is

stored in the variable 'number'.

Lines 30 and 40 use IF/THEN to make a decision. If the number stored in the variable 'number' is equal to 1 or 2, then the computer prints a message. Try typing a number that is greater than 2, what happens? The computer only acts in lines 30 and 40 if the number is 1 or 2 (after all, the lines tell it to do so!)

### Logical operators

In the IF/THEN statement you used an = sign. There are other signs you can use. Look at the following signs, and try to remember what they mean.

- < less than, smaller than
- > greater than, bigger than
- < > not equal to, not the same as

Signs like these are called *logical operators* because they help the computer work out the logic of what you want it to do.

### Random selection

The IF/THEN statement is used in a lot of computer programs, especially in games. You can use all of the commands learned so far to

make a simple guess-the-number game. Before you type in the program let's have a look at one more command – the command RND.

RND is used to make the computer pick a random number. This is very much like you pulling a piece of paper out of a bucket of pieces with numbers written on them. You are never sure which number will turn up. You may know the lowest and the highest number, but it would be difficult to predict which number will come out of the bucket next.

You call this type of number a 'random' number – a number picked out entirely by chance. To make the Amstrad choose a random number, we use the RND(X) command, where (X) specifies the range of numbers it is to choose from.

Have a look at the program below. It shows how the computer can pick a random number.

```
10 MODE 1
20 FOR go=1 TO 10
30 PRINT INT(RND(1)*10)+1;
40 NEXT go
50 END
```

When we ran the program, it printed the following numbers 1 7 5 7 6 4 9 8 8 5. The numbers chosen when you run the program will be different.

Notice how the limit of 10 is placed after the RND command. This means that the

computer can choose any number from 1 to 10. Try changing the program to make the computer pick larger numbers.

What happens if you leave out the semi-colon at the end of line 30?

### Guessing game

Now type in the next program. It is a bit longer than those you have tried so far. The program uses RND and IF/THEN. Try to work out how it 'works'.



```

10 REM ** Guess the number **
20 MODE 1
30 goes=0
40 number=INT(RND(1)*20)+1
50 PRINT "Guess-the-Number"
60 PRINT "I have chosen a number."
70 PRINT "It is between 1 and 20."
80 WHILE GUESS <> number
90 INPUT "Your guess is "; guess
100 goes=goes+1
110 IF guess > number THEN
    PRINT "Sorry, it is smaller.":
    SOUND 1,15,15,10
120 IF guess < number THEN
    PRINT "Sorry, it is bigger.":SOUND
    1,30,15,10
130 WEND
140 PRINT "WELL DONE!! You took
    ";goes "; guesses."
150 PRINT "The number was ";number
160 FOR sou=4 TO 200 STEP4:
    SOUND 2, sou,15,1:NEXT sou
170 END

```

RUN the program. If it does not work, check your typing.

### How it works

You should be able to see how the program works. The computer chooses a random number in line 40 and stores it in a memory box labelled 'number'.

The WHILE/WEND loop makes sure that the computer does not print the 'Well done' message while you have not guessed correctly. The IF/THEN lines check your

answer then print a clue. The numeric variable 'goes' keeps count of how many goes you have had.

There are some SOUND commands on lines 110, 120 and 160. The rising sound at the end of the program is produced by a FOR/NEXT loop. Note that this uses the command STEP to make the pitch of the sound rise in steps of 4 numbers at a time. So, first it plays pitch 4, then pitch 8, then pitch 12, etc. If you want to experiment, try making the STEP size a different number. You could even make it a random step size, using

```
STEP 1=INT(RND(1)*10)+1
```

or any other number in the brackets.

This is quite an easy program to improve. Can you change the colour? How about a larger spread of numbers?

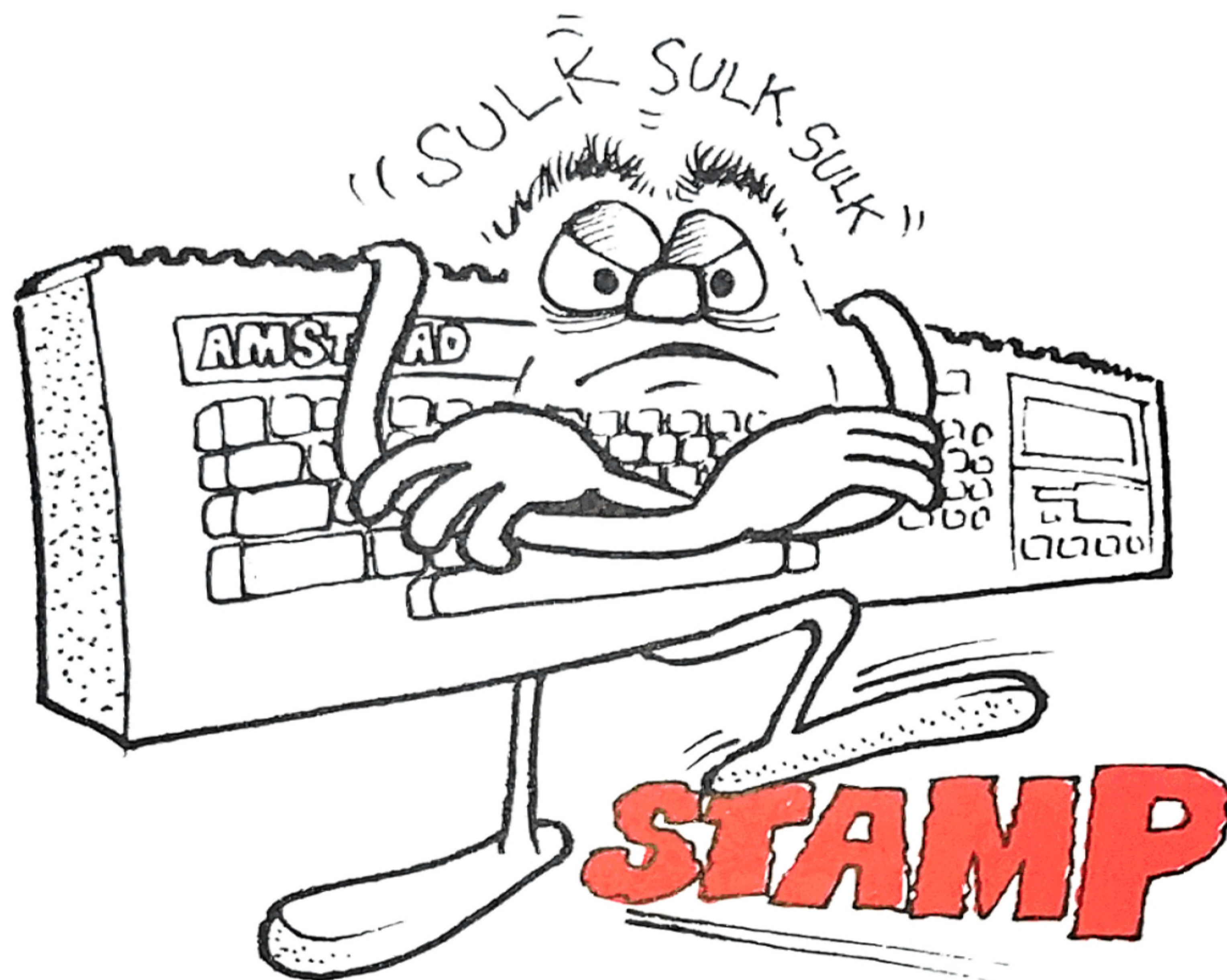
### Computer sulks

So far we have only used numbers in IF/THEN statements. It is also possible to use letters or words. Have a look at the program below. This is a simple example of an IF/THEN statement using a string variable.

```

10 REM *** Sulker ***
20 MODE 1
30 WHILE answer $ <> "YES" AND
    answer $ <> "NO"
40 INPUT "Do you like the Amstrad
    (YES OR NO) ";answer$
50 WEND

```



```

60 IF answer$="YES" THEN
    PRINT "You are my friend for life."
    ELSE PRINT "See if I care! I shall
    sulk."
70 END

```

This little program has a few special lines in it. Look at lines 30 to 50. What do you see?

A WHILE/WEND loop is used to make sure that the computer only accepts 'YES' or 'NO' as an INPUT. This is very useful and makes sure that the computer will keep asking the question until the person using it types in a *valid* answer. In this program either 'Yes' or 'No' would be valid answers.

Line 60 is an IF/THEN statement, using a *string variable*. Notice that the word is inside quote marks(""), just like when you PRINT a word or assign it to a variable.

There is something special that you must have spotted in line 60 – the word ELSE. ELSE is a BASIC command that can be used if an IF/THEN statement. It means that IF the condition is not true, THEN do something ELSE (or different).

Look at the following program. It uses IF/THEN ELSE to check an INPUT

```

10 REM *** Sums test ***
20 MODE 1
30 goes=0
40 PRINT "ARITHMETIC TEST"
50 WHILE goes < 10
60 number1=INT(RND(1)*50)+1
70 number2=INT(RND(1)*50)+1
80 PRINT "What is the answer to ";
    number1;" + ";number2
90 INPUT "Type your answer ";
    answer
100 goes=goes+1
110 IF answer=number1+number2
    THEN PRINT "CORRECT" ELSE
    PRINT "WRONG"
120 PRINT "Press a key for the next
    sum"
130 G$=INKEY$: IF G$="" THEN 130
140 WEND
150 PRINT "This is the end of the
    test."

```



This is a straight-forward program, so you should be able to follow how it works. Notice how all of the decisions are made in line 110.

What is missing is a way of counting how many sums are correct. So make these changes to the program.

```
30 goes=0:right=0
110 IF answer=number1+number2
    THEN PRINT "CORRECT":right=
    right+1 ELSE PRINT "WRONG"
160 PRINT "You scored ";right;"
    out of 10."
```

### Mindbender 10.1

The program still needs lots of improvements. You can make a few yourself. For instance, stop it printing the message on line 120 when the test has ended. Put in a line to make it display the score out of 10 after each answer.

**HINT: PRINT "Your score so far is  
";right;" out of ";goes**

If you cannot work out where the line goes, have a look at the back of the book.

How about making the screen clear after each sum? Where would you place CLS to make it clear the screen?

You could even have the messages appearing in colour. Look back to Chapter 8 to remind yourself how it is done.

Try to make the program into your own program by making as many improvements as possible. For example, why not add the SOUND routines that you used in *Guess-the-number*? You can learn a lot by changing programs.

### Remember

1. The IF/THEN statement is used to make decisions.
2. IF/THEN is a test. IF the statement is true, THEN the rest of the line is acted upon.
3. The word ELSE can be used in an IF/THEN statement. It makes the computer do something different IF the condition is NOT true.
4. The usual symbols to use for testing are > (greater than), < (less than), = (is equal to) and <> (is not equal to). You could also use <= (is less than or equal to) or >= (is greater than or equal to).

5. IF/THEN can also be used with AND and OR to test whether *two* things are true, e.g.

```
IF bullet = invader _ position AND fuel  
> 0 THEN GOSUB....  
IF answer$ = "Y" OR answer$ = "YES"  
THEN GOTO...
```

6. RND chooses a RaNDom number. A number must be placed in brackets after RND, e.g. RND(20) means choose a number between 1 and 20.

7. WHILE/WEND can be used to make sure an INPUT is what is wanted, e.g.

```
WHILE answer$ <> "YES"  
AND answer$ <> "NO"  
--  
--  
--  
WEND
```

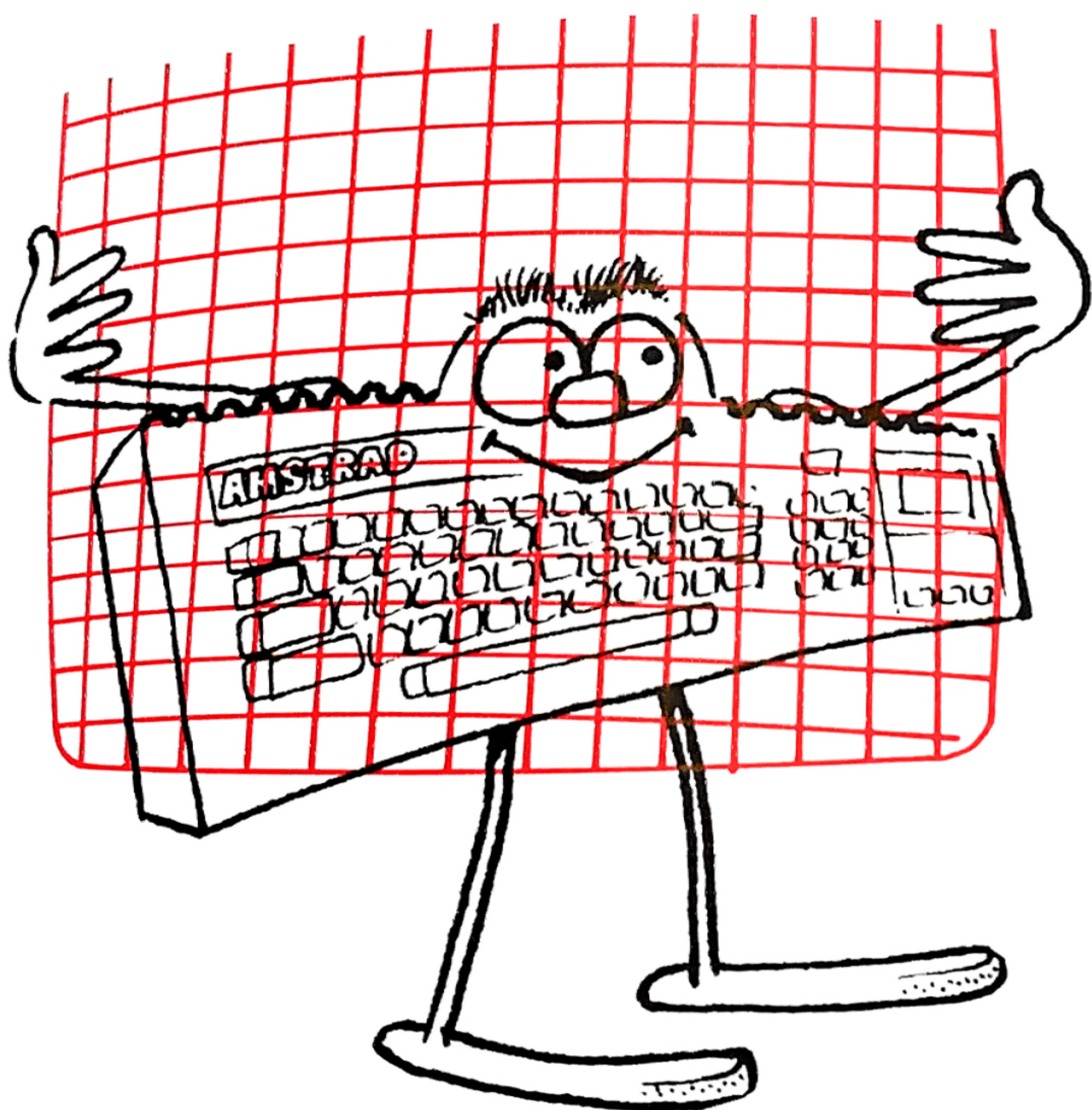


## Chapter 11

# Using the LOCATE command

Your computer divides the TV screen into an invisible grid, the size of which depends upon the MODE you use.

In MODE 1 the grid is 40 blocks across the screen by 25 blocks down. Note that the blocks



across are numbered 1 to 40, while the blocks down the screen are numbered 1 to 25 in this MODE, as shown in the diagram below:

To tell the computer where to start printing is very easy. You give it the position of the block (or rectangle), using co-ordinates. First you give it the *top* number (the distance across the top of the TV screen), then you give it the *side* number (the distance down the screen).

|                   |   |   |   |   |   |   |   |   |   |    |     |       |
|-------------------|---|---|---|---|---|---|---|---|---|----|-----|-------|
|                   | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | ... | TO 40 |
| 1                 | X |   |   |   |   |   |   |   |   |    |     |       |
| 2                 |   |   |   |   |   |   |   |   |   |    |     |       |
| 3                 |   |   |   |   |   |   |   |   |   |    |     |       |
| 4                 |   |   |   |   |   |   |   |   |   |    |     |       |
| 5                 |   |   |   |   |   |   |   |   |   |    |     |       |
| 6                 |   |   |   |   |   | Y |   |   |   |    |     |       |
| 7                 |   |   |   |   |   |   |   |   |   |    |     |       |
| 8                 |   |   |   |   |   |   |   |   |   |    |     |       |
| 9                 |   |   |   |   |   |   |   |   |   |    |     |       |
| 10                |   |   |   |   |   |   |   |   |   |    |     |       |
| .                 |   |   |   |   |   |   |   |   |   | Z  |     |       |
| .                 |   |   |   |   |   |   |   |   |   |    |     |       |
| .                 |   |   |   |   |   |   |   |   |   |    |     |       |
| To 25 (In MODE 1) |   |   |   |   |   |   |   |   |   |    |     |       |

Have a look at the letters printed on the grid to see how this works, using the LOCATE command:

Position X : LOCATE 1,1  
Position Y : LOCATE 6,5  
Position Z : LOCATE 9,10

Did you notice how the numbers are placed separated by a comma? This is how it should always be done.

### Using the LOCATE command

The program below will allow you to use LOCATE to place words or symbols on the screen. Be careful with the numbers you type in, as the computer will not check them.

```
10 REM *** LOCATE Positions ***
20 MODE 1
30 INPUT "The first locate position
   is ";x
40 INPUT "The second locate position
   is ";y
50 INPUT "The symbols or words
   are ";word$
60 CLS
70 LOCATE x,y: PRINT word$
80 LOCATE 1,23: PRINT "The locate
   position is ";x," ";y
90 G$=INKEY$: IF G$="" GOTO 90
100 RUN
```

This program uses commands that you have used before. The *three variables X, Y and word\$* are used in lines 70 and 80. See what happens if you try to PRINT words\$ on line 23. The message in program line 80 will write over it.

To continue, you simply press a key. The

effect of line 90 is to make the program wait until a key is pressed. You may like to use this method in your own programs.

What happens if you try to input invalid LOCATE positions – more than 40 across or more than 25 down the screen? And what happens if you try to print on the very bottom of the screen, e.g. LOCATE 30,25? The problem is caused by the cursor re-appearing at the bottom and causing everything to scroll up a line. This can be a nuisance.

### LOCATE in other modes

If you use other MODEs, you will find that the number of columns across and lines down the screen are different. Below is a chart to help you remember the screen size for the different MODEs

| Mode | Size                              |
|------|-----------------------------------|
| 0    | 20 columns across x 25 lines down |
| 1    | 40 columns across x 25 lines down |
| 2    | 80 columns across x 25 lines down |

### FRE(0)

Would you like to know how many bytes of memory remain unused after you program?

You can find out the length of memory space available at any time by typing:

**PRINT FRE(0)**

## Remember

1. LOCATE **across,down** enables you to print information at a particular point on the screen.
2. First you give the number of columns across the screen, then the number of rows down the screen. The two numbers must be separated with a comma.
3. The row and column numbers start at 1, 1 in the top left hand corner of the screen.
4. The number of columns across the screen will be 1 to 20, 40, or 80 depending upon the MODE you are using. In MODE 1 there are 40 columns across the screen, numbered from 1 to 40.

5. The number of rows down the screen will be numbered from 1 (at the top) to 25, depending upon the MODE you are using. In MODE 1 there are 25 rows down the screen, numbered from 1 to 25.
6. To find out how much free space is available.

## PRINT FRE(0)

7. You cannot use up all the memory just for your program or it won't work. All programs need 'head-room' or 'workspace' in which they store variables and strings, and carry out calculations.



## Chapter 12

# Your first arcade game!

Now you know how to print things anywhere on the screen, the time has come to develop your first arcade-style game!

Although it may look a little complicated, there is nothing in the next program that you haven't done before – or can't work out with a little thought and a few experiments.

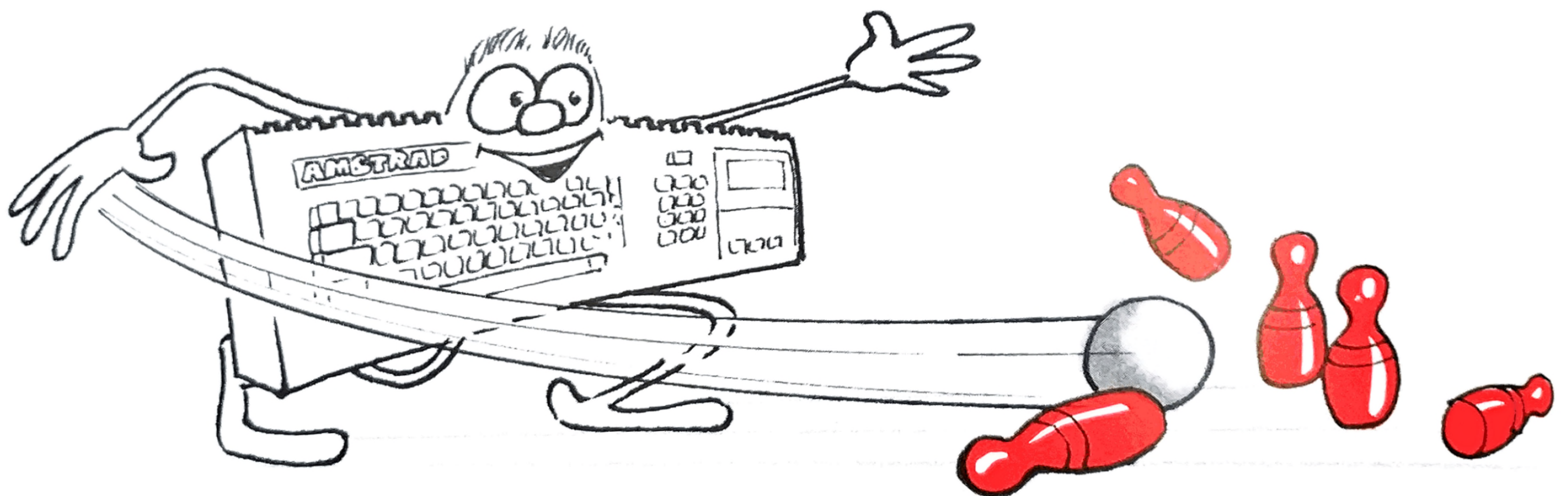
If you want to be able to design your own arcade games, it's worth spending a bit of time studying how this program works. Almost every arcade game uses precisely the same principles as this simple program.

Two things to remember when typing it in. First, you don't need to type in the REM statements, if you don't want to – they are there simply to help you follow what each line does.

Secondly, to make it easier to follow the listing, we have put in a lot of blank lines. You must leave these out when typing in the program.

```
10 REM *** WORDZAP ***
```

```
30 MODE 1
```



```

40 X=18
70 REM *** get target word ***
90 INPUT "Type in a long
word";word$
100 CLS:LOCATE 15,1:PRINT word$
120 REM *** Is any key pressed? ***
140 IF INKEY(1)=0 THEN x=x+1:IF
x>35 THEN x=35:GOTO 200:
REM Right Arrow
150 IF INKEY(8)=0 THEN x=x-1:IF
x<1 THEN x=1:GOTO 200:REM
Left Arrow
160 IF INKEY(47)=0 THEN GOSUB
240
180 REM *** Print gun ***
200 LOCATE x,24:PRINT "↑"
210 GOTO 140:REM Check key press
again
220 END
240 REM *** Fire bullet ***
260 FOR bullet=25 TO 1 STEP-1
270 LOCATE X+1,bullet:PRINT "!"
280 LOCATE X+1,bullet:PRINT ""
290 NEXT:RETURN

```

To play the game, type in, say, your name when you are asked for a long word. Then use the LEFT and RIGHT ARROW keys to move the 'gun'. Press the SPACEBAR to fire the gun.

To stop the game, press the ESCAPE key twice. Then type

**CLS < ENTER >**

to clear the screen and LIST the program.

### How it works

Line 30 – selects MODE1 and clears the screen. From the chart in the previous chapter, you can see that MODE1 gives you a screen 40 characters wide, by 25 lines down. There are 4 of the 27 colours available in this MODE (see Chapter 8).

Line 40 – sets the initial value of X approximately half-way across the screen (see later why not precisely half-way (20)).

Line 90 – uses INPUT to stop the program until you have typed in a word, which will be used as a target.

Line 100 – clears the screen and prints the target word LOCATED at 15,1. This means column 15 across the screen on the top row of line (1). If you want to make the game easier, change the 1 to a number, say 10, and see what happens. If you want to use only short words as targets, you also could try changing the 15 to, say, 16 or 17.

### Using INKEY(n)

Line 140 – is really 3 or 4 program lines rolled into one! First, it uses INKEY(1)=0 to check whether the RIGHT ARROW key has been pressed. If so, it adds 1 to the previous value of X.

When the game starts, the value of X is 18 (line 40). Now it becomes  $18 + 1 = 19$ . Next time round, X is already 19, so if the RIGHT ARROW is still pressed, X becomes  $19 + 1 = 20$ , etc. The effect of this will be to move the gun one position to the right each time the right arrow key is pressed.

The next part of the line checks whether the value of X has reached 35. IF it has, the instruction 'THEN X = 35' re-sets it to the maximum value of 35. It can never grow to 36, 37, 38, etc which would take it off the right hand side of the screen, with disastrous results!

The reason that a maximum value of 35 has been used (and not 40 as you might expect), is to allow for the width of the gun – which is effectively 5 characters wide.

To check the effect, replace line 140 with

```
140 IF INKEY(1)=0 THEN X=X+1 :  
    GOTO 200
```

then be sure to put the line back to our way again!

Line 150 – checks whether the **left arrow** key has been pressed using `INKEY(8) = 0`. There is an explanation of what these INKEY numbers mean in Chapter 8, page 20, of your User Instructions.

IF the **left arrow** key has been pressed, THEN the value of X is decreased by one. The

effect of this will be to move the gun one place to the left.

As with line 140, it is necessary to prevent the gun moving off the side of the screen, but this time, the left side. The lowest column number for a LOCATE statement is one (1), so we stop X becoming 0, -1, -2, etc, by saying that IF X is less than (<) one, THEN (LET) X=1.

Line 160 – yet another INKEY statement! This time we check to see if the SPACEBAR has been pressed, so we use INKEY (47) which Appendix III, page 16, of your *User Instructions* tells you is the number for the spacebar. If you were to change the number inside the brackets to, say, (53) – which is the INKEY number for the letter 'F' key, the bullet would fire only when you press the 'F' key.

As you can see, though the INKEY command looks complicated, it is really extremely easy to use. It has the great advantage over INPUT, that it doesn't stop the program if no key is pressed, and you don't have to press <ENTER>. This makes it a really useful command in fast-moving games!

### How the gun moves!

Line 200 – prints one keyboard symbol, with a space on each side. The symbol is an upward pointing arrow head sign. You can find it beside the CLR key.

Of course, you could use any other characters, if you think they might look a bit more like a gun!



Placing a blank space on either side of the gun is a cunning programming trick that might take you a month of Sundays to work out for yourself, but which, like most really bright ideas, is beautifully simple!

To see what it does – and you really must try this – RUN the program again, after removing the two spaces. Don't bother to fire the gun, just try moving it right and left... do it right now, before reading on.

Did you see what happens? You get a trail of old bits of gun! So now you know what the spaces do – they rub out any bits of the gun from the previous X position by printing a blank space over the end of the last position.

As you've probably guessed by now, we move the gun by altering the X value in the LOCATE X24 statement. When the program starts, X = 18 (line 40) which is roughly halfway across the screen, allowing for the length of the gun.

You might wonder why we didn't set the gun even lower down, for example, on the bottom line – line 31. If you try it, you'll see why – you run into rather annoying scrolling effects! You may also discover a minor problem if you set it on line 30, but you can overcome this one if you set the maximum movement left to 1 less than the present 35 (line 140).

To make the game easier, you could reduce the second number in the LOCATE to,

say, 25, which would print the gun higher up the screen and nearer to the target. This is one of the ways that SKILL levels can be set in arcade games.

### Using a SUBROUTINE

Line 260 – is the start of a subroutine that comes into operation the moment you press the SPACEBAR.

A subroutine is simply a miniature program within a program.

Briefly, the computer is sent to a subroutine by the GOSUB command (see line 160). This is similar to GOTO except that after GOing to a SUBroutine, the computer will remember where it came from in the program and RETURN there after carrying out the instructions in the subroutine and meeting the RETURN instruction.

When you press the SPACEBAR, the computer is sent to the subroutine which fires the bullet. Then it RETURNS to the main program to print the gun again.

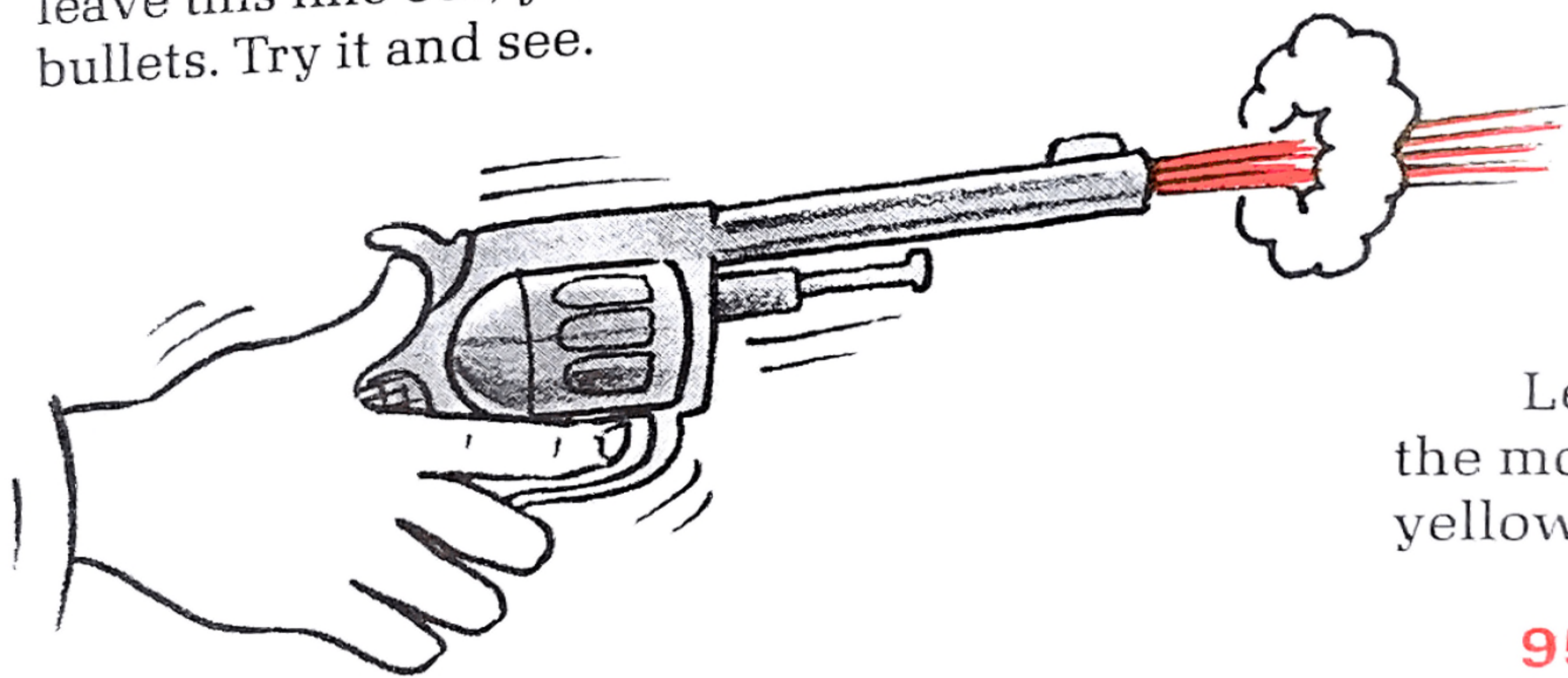
You would normally only use a subroutine (rather than GOTO), when you want to repeat the same set of commands at several different places in the program. So in this particular case, where we are only going to the routine from one point in the program (line 160) we could just as easily have used GOTO, instead of a GOSUB. We chose to use a subroutine here, mainly to introduce you to the methods.

### Firing the gun

Line 260 – uses a FOR/NEXT loop to count backwards from 25 to 1. The STEP-1 command means 'reduce the number by 1, each time the loop goes round'. If you don't like the way the centre of the gun disappears each time it is fired, change the number from 25 to 24.

Line 270 – prints an exclamation mark (!) to represent a bullet. If you prefer, you could use a star (\*) instead. Note that the bullet is printed one position to the right of X. This makes it appear to come out of the centre of the gun.

Line 280 – prints a blank space over the previous bullet position to rub it out. If you leave this line out, you will get a stream of bullets. Try it and see.



Line 290 – uses NEXT to check whether the FOR/NEXT loop has finished. In effect, it asks 'have I finished counting down from 25 to 1?' If it hasn't (i.e., the numeric variable 'bullet' is still greater than zero), NEXT sends the computer back to line 260 which reduces the value of bullet by one again.

When the value of 'bullet' reaches one (1), the computer moves on to the following statement on the line, which is RETURN. This sends it back to where it came from (i.e. line 160, immediately after the GOSUB instruction), so that it can carry on with the next instruction in the main program.

### Adding colour

From what you learned in Chapter 8, you may remember that the mode we are using (MODE 1) allows us to use a total of 4 colours. So, if you have a colour monitor, you don't have to stick to boring old black and white!

The colours available in MODE 1 are:

| Colour        | Pen/Paper |
|---------------|-----------|
| Blue          | 0         |
| Bright Yellow | 1         |
| Bright Cyan   | 2         |
| Bright Red    | 3         |

Let's stick to the yellow PEN colour for the moment and keep the target (word\$) yellow. Just add this line:

**95 PEN 1: REM yellow**

How do you think we can PRINT the gun in red? Easy, just find out which line prints the gun, then add the appropriate colour command (this time 3 for red) immediately before it. You can see from the REM lines that the gun is printed after line 180 – Ah! There it is at line 200... So now add:

**195 PEN 3 : REM red**

Now you see how useful those nicely spaced out REM statements can be when it comes to making alterations to a program.

Finally, let's change the bullet to cyan:

**255 PEN 2 : REM white**

RUN the program again, to check that you are happy with the colours so far.

How about a nice, bright yellow paper? No sooner said, than done! A yellow paper is PAPER 1 according to the chart above, so let's add the instruction just before we print the target – word\$ – say, at line 93:

**93 PAPER 1 : REM yellow  
background**

Now RUN the program again. OOOPS!!! What's happened to the target? Obviously, our bright yellow words, doesn't show up too well on a bright yellow paper, so let's change it to blue:

**95 PEN 0 : REM blue**

That should do it! If you find that the cyan bullets don't show up very well on your screen, change them to red by typing:

**255 PEN 3 : REM red**



# Chapter 13

## Lines and pictures

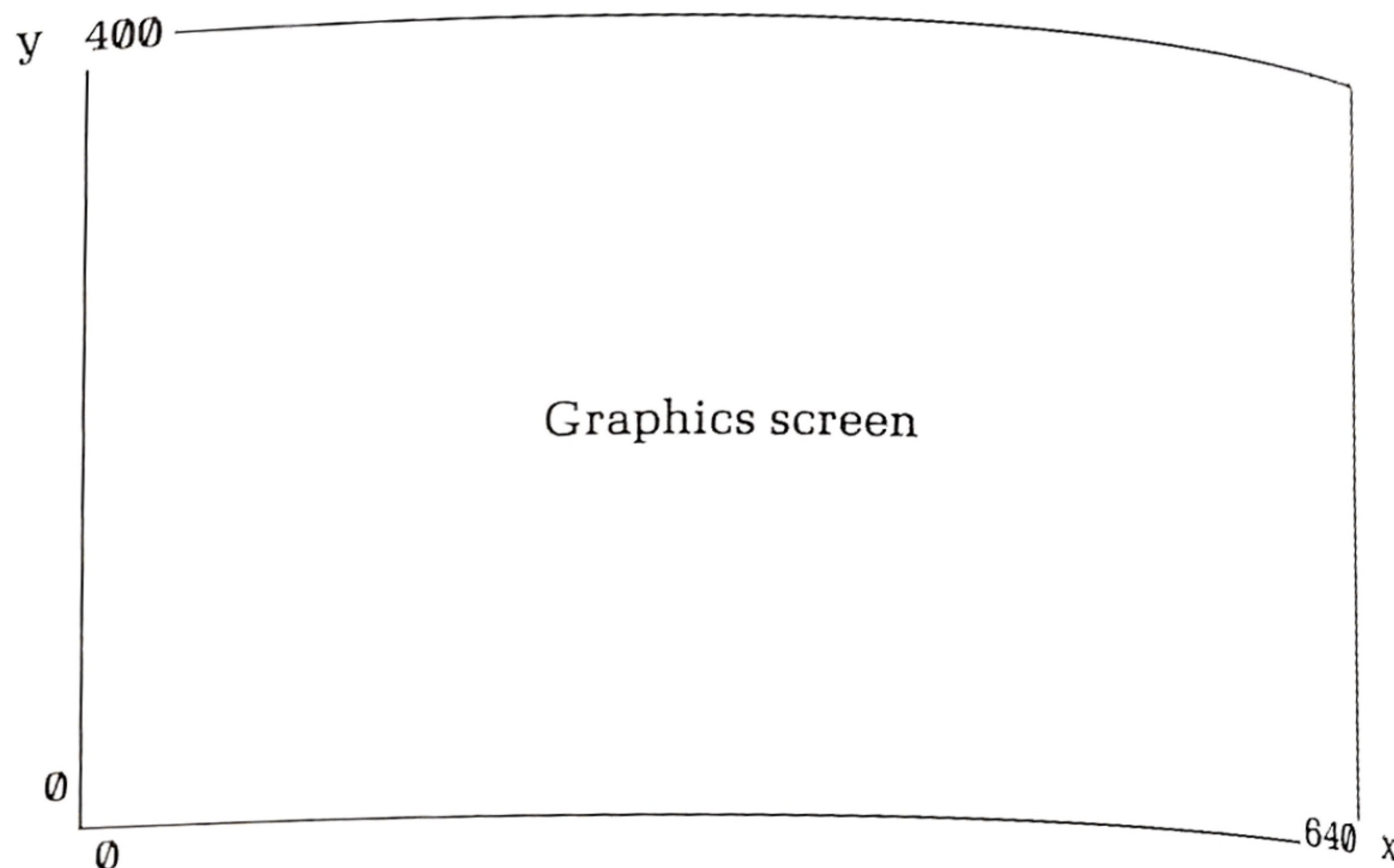
Most home computers can make lines on the screen and fill areas in with different colours. We call the pictures made by computers *graphics*.

The Amstrad can produce very good graphics, although some of the commands are very difficult to use. Already you have made your own characters, but these are only a small part of the Amstrad's graphics capabilities. MODEs 0, 1, and 2 are graphics modes. That means you can draw lines and block in areas of colour using these MODEs.

### Graphics grid

The *graphics* screen grid is different to the *text* grid which you have been using to print letters and symbols on.

Look at the screen plan below. This is Mode 2 graphics screen plan. Can you see that there are 640 units along the top and bottom and 400 units up the sides? Another important difference is that the 0,0 position is at the *bottom* of the screen, not the top as it was when you used the LOCATE command.



### Choosing a graphics MODE

The Amstrad uses up a lot of its memory in the graphics modes. So the finer the lines you want to be able to draw, the more memory is required.

For drawing really fine lines MODE 2 is best, MODE 1 is perhaps the best for making patterns with lines. In MODE 1 you can only use 4 colours; in MODE 2 you have only two colours available, but you have more memory to use. MODE 0 draws with thicker lines but allows you to use 16 of the 27 colours.

## Drawing lines

If you understand a little bit about co-ordinates you will find this chapter quite easy. One thing to note is that the start of the co-ordinates for the graphics screen is at the bottom left hand corner. Look at the screen plan in Appendix VI of your Amstrad User Instructions there are much more detailed screen plans. It is worth getting this out and having a look.

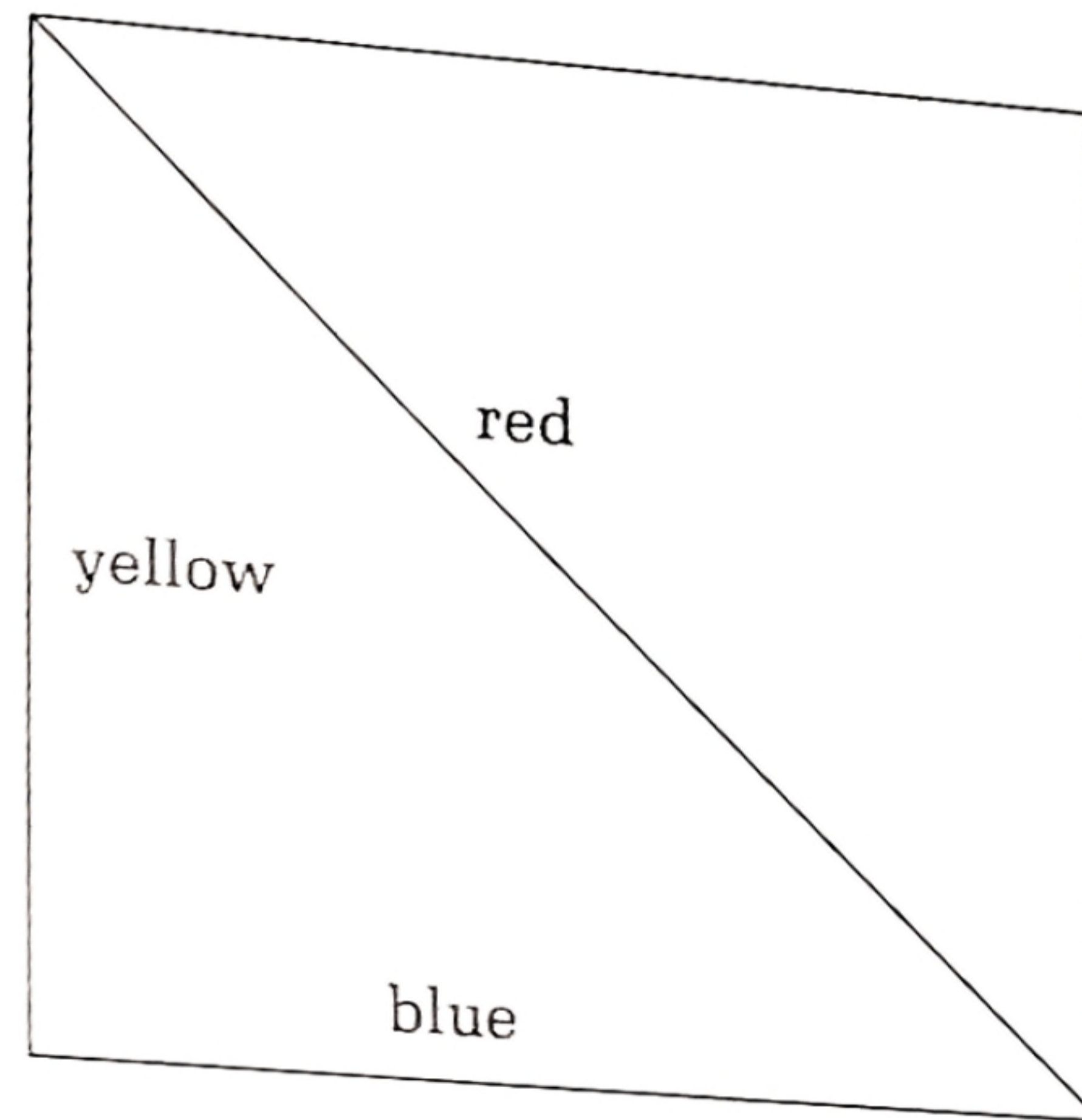
Now, to get the computer to draw a line we have to do three things.

1. Enter a graphics MODE e.g. MODE 1.
2. Tell the computer where to DRAW from.
3. Tell the computer where to DRAW to.

We use the commands MOVE and DRAW to move to a place and to draw to a place on the screen. The picture below will help you to understand how to use DRAW command. After each command we place two numbers separated by a comma. The numbers tell the computer where to DRAW to on the screen. Enter the following program, RUN it and look at the diagram below.

```
10 REM *** Triangle ***
20 MODE 1:REM any MODE 0 to 2
30 DRAW 0,400,1:REM straight line
  from bottom left to top left in
  yellow colour
40 DRAW 640,0,3:REM straight line
  from top left to bottom right in
  red colour
50 DRAW 0,0,2:REM straight line from
```

bottom right to bottom left in  
blue colour  
60 DRAW = 1 to  
1000:NEXT:REM waits for 3 1/2  
seconds before the prompt cursor  
is displayed

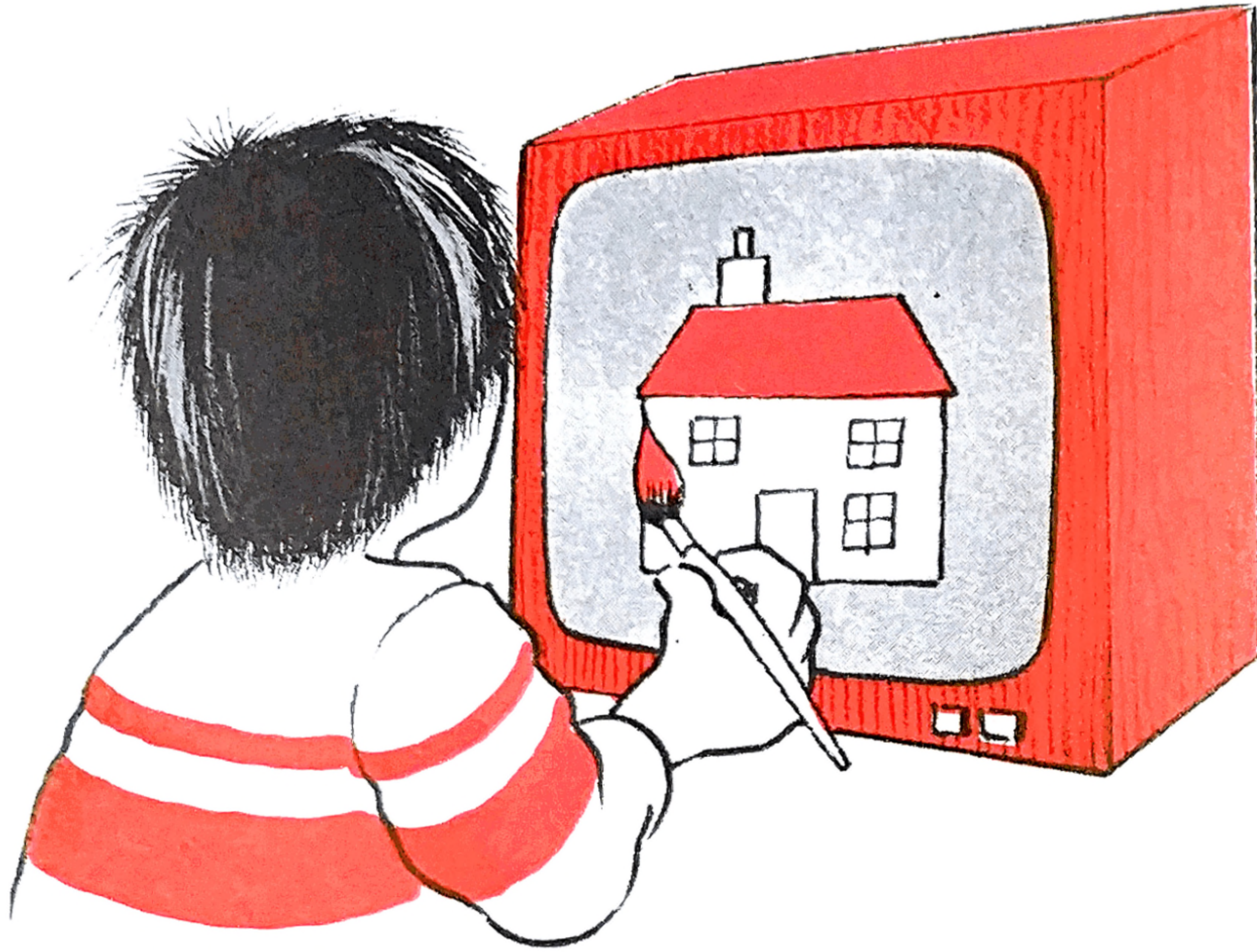


The Amstrad should draw a triangle in the lower half of the screen.

The other useful command is MOVE. The MOVE command will let you move the graphic cursor to any point on the screen.

## Drawing pictures

You can see that it is very easy to build up interesting drawings using the MOVE and DRAW commands. The following program draws a simple house. Try to follow where the computer is drawing to on the screen plan in your *Amstrad User Instructions*. If you understand how the co-ordinates are being



used, try to write your own program that will draw a picture.

```

10 REM *** Draw a house ***
20 MODE 1
30 REM Draw a block
40 MOVE 150,100
50 DRAW 150,300
60 DRAW 450,300
70 DRAW 450,100
80 DRAW 150,100
90 REM Draw roof
100 MOVE 150,300
110 DRAW 300,400
120 DRAW 450,300
130 REM Draw a door
140 MOVE 200,100
150 DRAW 200,200
160 DRAW 250,200

```

```

170 DRAW 250,100
180 DRAW 200,100

```

Now you carry on with the program. Make the computer draw a couple of windows. It is easier if your first draw it on graph paper or a screen plan.

### Saving space

You may be thinking that a lot of program lines will be used up if the drawing is complicated. This can be avoided by writing several instructions on the same program line. Don't forget that each instruction must be separated by a *full colon* (:).

The short program below uses the colon method, it saves space and computer memory.

```

10 MODE 1
20 DRAW 640,400:MOVE
640,0:DRAW 0,400:
MOVE 320,0:DRAW 320,400:
MOVE 0,200:DRAW 640,
200

```

Try the program. If you miss out the colon (:), the computer may become confused and give you an ERROR message. There are other ways of drawing things on the screen but they use commands that are too complicated for beginners.

### Patterns

The next three programs are a bit more complicated than those you have seen so far. They use MOVE and DRAW to make patterns.

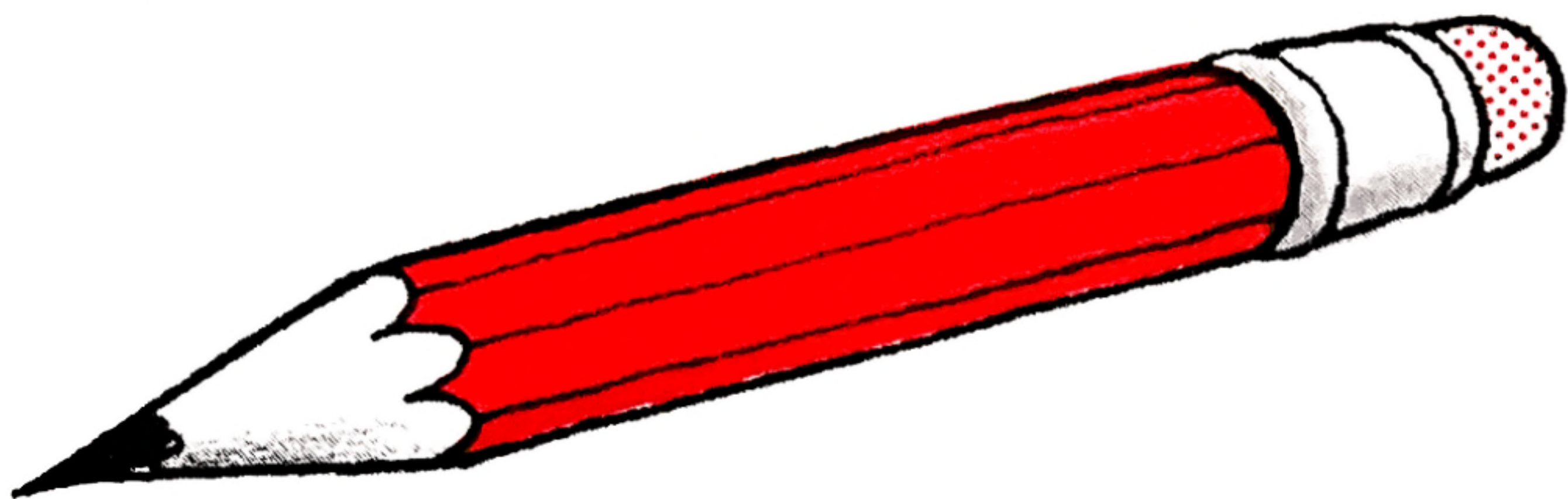
Type them in and see what they do. Number 3, *DRAW IT*, is a program that helps you draw pictures on the screen; read the instructions carefully.

```

10 REM *** Pattern ***
20 MODE 1
30 INK 0,20
40 X=640:Z=0:Y=400
50 FOR go=0 TO 2
60 FOR n=Z TO X STEP 40
70 MOVE n,Z:DRAW X,n
80 NEXT n
90 FOR n=X TO Z STEP-40
100 MOVE n,Y:DRAW Z,n
110 NEXT n
120 X=300:Z=150:Y=300
130 NEXT go
140 GOTO 140

```

A program that uses straight lines to make curves! You may have done the same sort of thing with a pencil and ruler.



### Gunshot!

The next program is rather different . . .

```

10 *** Gunshot ***
20 MODE 1
30 INK 1,0,26:SPEED INK 7,7
40 FOR n=1 TO 60

```

```

50 MOVE 320,200
60 DRAW INT(RND(1)*640)+1,
  INT(RND(1)*400),1
70 NEXT N
80 FOR S=15 TO 0 STEP -1
90 SOUND 1,300,40,S,7,12,2:NEXT S
100 FOR W=1 TO 500:NEXT W
110 IF S > 0 THEN GOTO 50

```

Well, almost like a bullet going through a window! You should be able to follow the part of the program that makes the lines. Line 30 changes the colour white to black, then changes white back to white. What effect do you think this has? Try missing out the line.

### DRAW it!

Our next program enables you to DRAW on the screen.

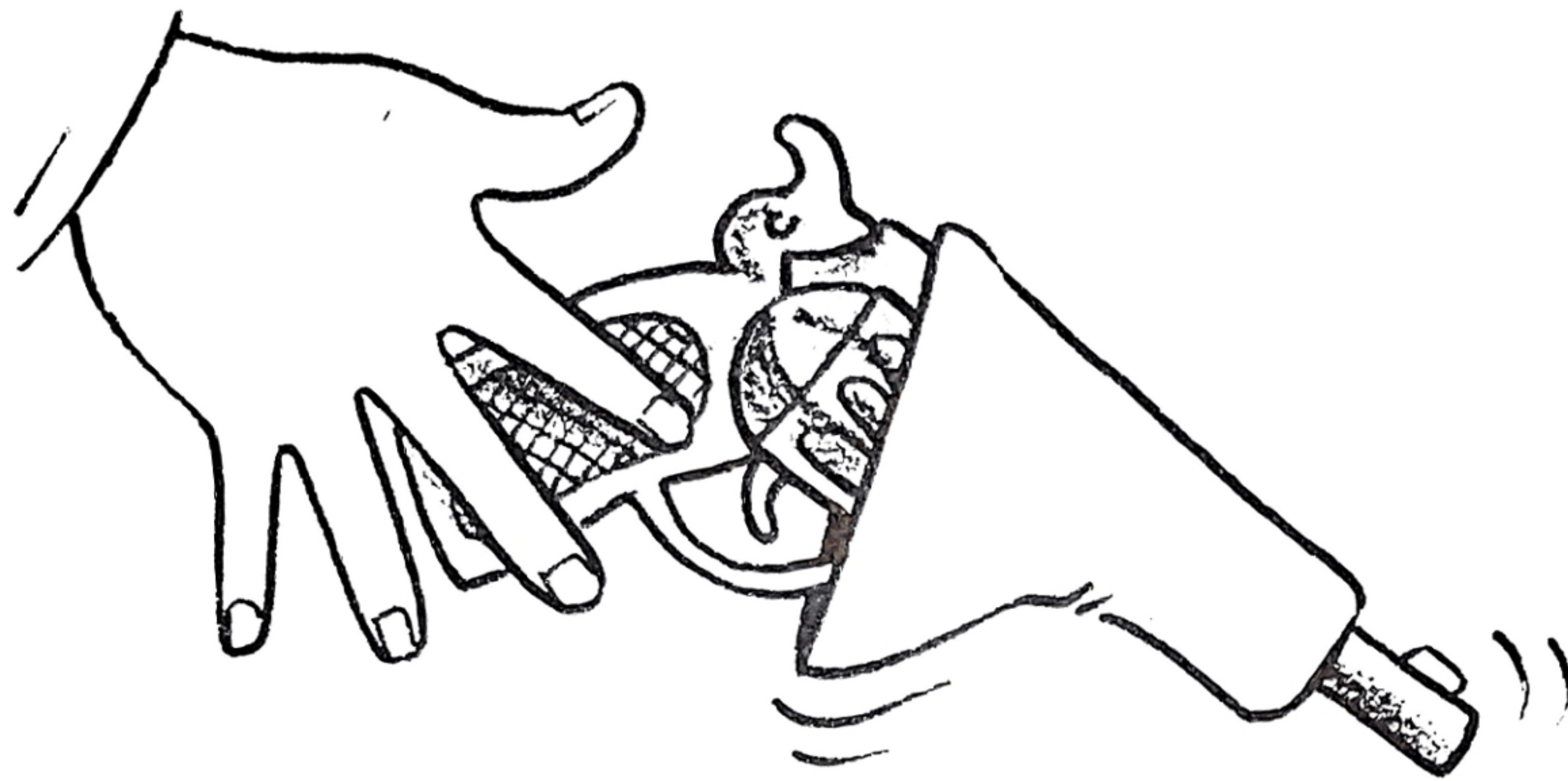
Use the cursor control keys (the four arrow keys) to DRAW. If you hold down the SPACE BAR while drawing, the line will disappear. You can press two keys at once to move diagonally. Try not to go off the edge of the screen or you may not find your way back onto it!

Remember that you do not have to include the REM statements, they are to help you understand the program.

```

10 REM *** Draw it ***
20 MODE 1
30 X=320:y=200:REM Centre of screen
40 MOVE x,y:REM Start at centre of
  screen
50 IF INKEY(0)=0 THEN Y=Y+1:REM

```



```

Move up
60 IF INKEY(2)=0 THEN Y=Y+1:REM
Move down
70 IF INKEY(8)=0 THEN X=X+1:REM
Move left
80 IF INKEY(1)=0 THEN X=X+1:REM
Move right
90 IF INKEY(47)=0 THEN C=3 ELSE
C=1:REM
Space bar alters colour to red
100 PLOT x,y,c:REM Plot line in colour
C to position x,y
110 GOTO 50

```

### How it works

The drawing program uses INKEY to test for a particular key being pressed. The INKEY key numbers are given in your *User Instructions* on Appendix III, page 16.

PLOT X,Y,C is only another way of telling the computer to draw – ‘X’ and ‘Y’ are variables, controlling the position on the screen. Can you see where X and Y are changed? ‘C’ is a variable that changes the

colour of the line from blue to red, when you hold down the SPACEBAR (key 47). This line is drawn in red.

To move round the screen, you start at the centre, where ‘X’=320 and ‘Y’=200. If you press the RIGHT ARROW key (key 1) line 80 makes the variable ‘X’ become  $320 + 1 = 321$ , so a line is drawn one small square to the right by line 100. If you continue to press a key ‘X’ or ‘Y’ becomes smaller or larger. This uses precisely the same methods as you used to control the gun position in the previous chapters.

### Changing colours

So far, we have only used yellow and red as colours for our drawings. It is possible to change the colour of the lines but remember that only MODE 0 allows you to use 16 colours. You will use the command to change graphics colours in the next chapter.



## Remember

1. MODES 0,1,2 are used for drawing lines on the screen. MODE 2 gives the finest lines but has only two colours, MODE 0 gives thicker lines but uses all 16 colours.
2. The graphics screen is on a grid 640 across x 400 down.
3. The command to move to a position on the screen is MOVE **x,y**, where **x** and **y** are numbers telling the computer the co-ordinates to move to.
4. The command to draw to a position on the screen is DRAW **x,y**.
5. More than one instruction can be placed in a program line, provided each instruction is separated by a colon (:).
6. INKEY(**number**) is used to test for a particular key being pressed. Details are on Appendix III, page 16, of the Amstrad User Instructions.



## Appendix

The following two programs are very long, compared to those you have seen in this book. When typing them in you must take extra care not to make a mistake. Finding mistakes can take a long time if you are not used to looking for 'bugs' in a program. It might be best if you type in part of the program at a time, SAVE it, then LOAD it later so you can add more of the listing. In this way you will not become too bored with typing them out.

The first program is based upon the board game, Mastermind. You have to guess the sequence of coloured pegs under a cover at the top of the games board. The computer displays a board plus your attempts to discover the colour code. Full use of colour is made in this program; the game is very attractive and fun to play. You will find full instructions in the program code.

### Masterbrain

```
10 REM *** Masterbrain ***
20 DIM b$(10),y(10),z(10),
    c(5),g(4),r(4),c%(4), colour(4)
```

```
30 MODE 1
40 GOSUB 1490: GOSUB 1640:
    GOSUB 1680
50 MOVE 255, 1%: DRAW 320, 1%
60 GOSUB 1540
70 CLS: b%=4: w%=5: f%=0: g$="":
80 GOSUB 970: PRINT
90 c(0)=4
100 FOR n=1 TO 4
110 c(n)=INT(6*RND(1))+1)
120 IF c(n)=6 THEN c(n)=7
130 NEXT n
140 FOR n=1 TO 4
150 x=c(n)
160 GOSUB 750
170 c(n)=x
180 NEXT n
190 P$=""
200 FOR x1=1 TO 4
210 P$=P$+CHR$(c(x1))
220 NEXT x1
230 PEN 11: PRINT
    "MASTERBRAIN": PEN 4:
    PRINT " "
240 FOR P=1 TO 10
250 PEN 4
260 PRINT "GUESS: ": count=0:
    g$=""
```

```

270 FOR count=1 TO 4
280 c$=INKEY$: IF c$="Y" OR c$="
"W" OR c$="R" OR c$="G" OR
c$="B" OR c$="P" THEN 290
ELSE GOTO 280
290 PEN 3: g$=g$+c$
300 PRINT c$;
310 NEXT count
320 SOUND 1,30,10,15: SOUND
2,60,5,15
330 GOSUB 1110
340 GOSUB 1270
350 b$ (P)=g$
360 GOSUB 510
370 IF b=4 THEN GOSUB 910: f%=1:
GOSUB 1110: GOSUB 1320:
GOSUB 1430
380 GOSUB 600
390 PAPER 1: LOCATE 1, b%+1:
PRINT: TAG: PLOT (6*64),
378-(b%*16), 4: PRINT
STRING$ (b, CHR$ (224));:
TAGOFF
400 y(P)=b
420 TAG: PLOT (6*64),
378-(w%*16), 1: PRINT;
STRING$ (w,CHR$(224));:
TAGOFF: b%=b%+2: w%=w%+2:
PEN 3
430 z(P)=w
440 NEXT p
450 CLS: LOCATE 1,10: PRINT
"SORRY "; LOCATE 1,12:
PRINT "YOU "; LOCATE
1,14: PRINT "HAVE ";:
LOCATE 1,16: PRINT "LOST!"
460 PEN 3: PRINT "THE CODE

```

```

WAS "": PRINT
470 PEN 15: PRINT SPC(3): p$
480 f%=1: GOSUB 1110: GOSUB
1320
490 FOR ti=1 TO 500: NEXT:
LOCATE 1,24: GOSUB 1430
500 REM Black Pegs
510 FOR x1=1 TO 4
520 g(x1)=ASC (MID$ (g$, x1, 1))
530 NEXT x1
540 b=0
550 FOR k=1 TO 4
560 IF g(k) < c(k) THEN 580
570 b=b+1
580 NEXT k
590 REM White Pegs
600 FOR x1=1 TO 4
610 r(x1)=ASC (MID$(p$, x1, 1))
620 NEXT x1
630 w=0
640 FOR i=1 TO 4
650 FOR j=1 TO 4
660 IF g(i) < > r(j) THEN 710
680 w=w+1
690 r(j)=0
700 GOTO 720
710 NEXT j
720 NEXT i
725 IF b > w THEN GOTO 740
730 w=w-b
740 RETURN
750 REM change
760 IF x < > 1 THEN 790
770 x=82
780 RETURN
790 IF x < > 2 THEN 810
800 x=71: RETURN

```

```

810 IF x=3 THEN 840
820 x=89
830 RETURN
840 IF x=4 THEN 870
850 x=66
860 RETURN
870 IF x=5 THEN 900
880 x=80
890 RETURN
900 x=87
910 RETURN
930 LOCATE 1, b%+1: PRINT
    SPC(12)
940 PEN 14: LOCATE 1,5: PRINT
    "YOU WIN"
950 PRINT "code:": PEN 15: PRINT
    p$
960 RETURN
970 REM Graphics
980 CLS
990 FOR i%=378 TO 502 STEP 31
1000 MOVE i%,0: DRAW i%,320,5
1010 NEXT i%
1020 FOR i%=28 TO 340 STEP 32
1030 MOVE 385, i%: DRAW 492, i%,5
1040 NEXT
1060 REM More graphics
1070 TAG: PLOT 425,345,6: PRINT
    "?": TAGOFF
1100 RETURN
1110 REM aski
1120 IF f%=1 THEN g$=p$
1130 FOR x1=1 TO 4
1140 COLOUR (x1)=ASC (MID$
    (g$,x1,1))
1150 NEXT x1
1160 FOR x1=1 TO 4

```

```

1170 c%=COLOUR (x1)
1180 NEXT x1
1190 RETURN
1200 REM Make colours
1210 IF c%=89 THEN c%=1: RETURN
1220 IF c%=82 THEN c%=3: RETURN
1230 IF c%=80 THEN c%=11: RETURN
1240 IF c%=87 THEN c%=4: RETURN
1250 IF c%=71 THEN c%=12: RETURN
1260 IF c%=66 THEN c%=8: RETURN
1270 REM Display
1280 FOR x1=1 TO 4
1290 TAG: PEN (x1): PLOT (x1*64)
    *28,378-(b%*16),5: PRINT
    PIN$; TAGOFF
1300 NEXT x1: TAGOFF
1310 RETURN
1320 REM Winner
1330 LOCATE 16,2: PRINT SPC(4):
    LOCATE 16,3: PRINT SPC(4)
1340 FOR n=512 TO 640 STEP 31
1350 MOVE n, 488: DRAW n, 448
1360 NEXT n
1370 MOVE 512,448: DRAW 640,448
1380 FOR s%=30 TO 200 STEP 10:
    SOUND 3, s%,20,15: NEXT
1390 FOR x1=1 TO 4
1400 TAG: MOVE (x1+15)*32,
    510-(2*16): PRINT PIN$;
    TAGOFF
1410 NEXT x1
1420 RETURN
1430 REM Again
1440 PEN 4
1450 LOCATE 1,28: PRINT
    "ANOTHER TRY?"
1460 a$=INKEY$: IF a$ <> "Y" AND

```

```

a$ <> "N" THEN 1460
1470 IF a$="y" THEN 70
1480 PEN INT (RND(1)*15)+1: PAPER
      INT (RND(1)*15)+1: PRINT
      "    GOODBYE ": GOTO 1480
1490 REM Intro
1500 MOVE 640,1: REM Plot
1510 PAPER 4:: CLS
1520 RETURN
1530 REM Codes
1540 PAPER 1
1550 SOUND 1,510,120,14
1620 RETURN
1640 REM CHARs
1650 SYMBOL AFTER 224: SYMBOL
      224,0,0,24,60,24,0,0,0: SYMBOL
      225,230,255,0,0,0,0,24,24:
      SYMBOL 226,80,126,23,226,
      126,60,24,24
1660 PIN$=CHR$(225)+CHR$(
      8)+CHR$(10)+CHR$(226)
1670 RETURN
1680 REM INSTRUCT
1690 CLS: PEN 2: LOCATE 13,1:
      PRINT "MASTERBRAIN":
      PEN 3
1700 PRINT: PRINT "This game is
      similar to the board game":
      PRINT "MASTERMIND": PRINT
      "You have to guess the correct
      order": PRINT "of the coloured
      pegs under the cover at the top of
      the board"
1710 PRINT: PRINT "You have to
      input the guess as a CAPITAL
      letter": PRINT: PRINT "e.g.
      RRBW (red, red, blue, white)":

```

```

PRINT
1720 PRINT "A WHITE peg indicates
      a correct colour in the right
      place": PRINT: PRINT "A
      YELLOW peg indicates a correct
      colour in the wrong place"
1730 PRINT: PRINT "The computer
      will not use blank spaces only
      coloured pegs."
1750 PRINT: PRINT: PRINT
      "    Press a letter to start"
1760 a$=INKEY$: IF a$="" THEN
      1760
1770 MODE 0: CLS
1780 RETURN

```

## Tower of Hanoi

This game is also a game of strategy and thought. It is based upon an ancient game played with rings on three pegs. All you have to do is to move the tower of rings from one peg to another. Sounds easy? You will find that to complete the Tower of Hanoi in under seventy moves is very difficult. A colourful game that will test your powers of concentration and intelligence to the full.

## Tower of Hanoi

```

10 REM *** TOWER OF HANOI ***
20 REM *** A game of logic ***
30 ENV 1,120,2,30
60 MODE 0: GOSUB 1240: FOR n=1
      TO 300: NEXT

```

```

80 DIM s$(6), a(3,7), a$(7)
90 GOSUB 1340
100 CLS: WHILE g$ <> "y" AND
    g$ <> "Y" AND g$ <> "n" AND
    g$ <> "N": LOCATE 1,10:
    PRINT "INSTRUCTIONS":
    LOCATE 1,12: PRINT "(Y or
    N)?: g$=INKEY$: WEND: IF
    g$="n" OR g$="N" THEN 120
110 MODE 1: GOSUB 840
111 PAPER 10: CLS
120 MODE 0
121 PAPER 10: CLS
130 LOCATE 1,20: PRINT STRING$(
    20, CHR$(225))
140 PEN 12
150 LOCATE 3,21: PRINT "1"
    LOCATE 9,21: PRINT "2":
    LOCATE 15,21: PRINT "3"
160 GOSUB 1460
170 FOR z=7 TO 1 STEP-1
180 a(1,z)=1
190 a(2,z)=z
200 a(3,z)=1
210 NEXT z
220 c=1: z=0
230 z=z+1
250 y=8
260 WHILE y <> 1: y=y-1
270 IF a$(a(z,y))=a$(6) THEN PEN 3
280 IF a$(a(z,y))=a$(7) THEN PEN 8
290 IF a$(a(z,y))=a$(1) THEN PEN 4
300 IF a$(a(z,y))=a$(2) THEN PEN 3
310 IF a$(a(z,y))=a$(3) THEN PEN 1
320 IF a$(a(z,y))=a$(4) THEN PEN 0
330 IF a$(a(z,y))=a$(5) THEN PEN
    11

```

```

340 LOCATE z*6-5, y+12: PRINT
    a$(a(z,y))
350 WEND
360 IF z < 3 THEN 240
370 PEN 11
380 LOCATE 10,6: PRINT "MOVE:
    ";c
390 IF a(1,2)=2 OR a(3,2)=2 THEN
    760
400 LOCATE 1,4: PRINT SPC(20):
    PEN 3
410 LOCATE 1,4: PRINT
    "FROM   ":PRINT
420 PEN 1
430 INPUT j: LOCATE 1,6: PRINT
    SPC(4)
440 PEN 3: LOCATE 1,4: PRINT j;
    "TO"
450 PRINT: PEN 1
460 INPUT k: LOCATE 1,6: PRINT
    SPC(5)
470 PEN 3
480 LOCATE 1,4: PRINT j;" TO
    ";k;" "
490 SOUND 1,100,5,15
500 IF j > 3 OR j < 1 OR k > 3 OR k < 1
    THEN GOSUB 1550: GOTO 400
510 PEN 12: GOES=GOES+1
520 IF k=j THEN 750
530 FOR I=1 TO 50: NEXT
540 d=0
550 d=d+1
560 IF a(j,d)=1 THEN 600
570 p=d
580 q=a(j,d)
590 IF p=d THEN d=7: GOTO 620
600 IF d < 7 THEN 550

```

```

610 GOSUB 1550: GOTO 400
620 d=0
630 d=d+1
640 IF a(k,d)=1 THEN 670
650 IF a(k,d) < q THEN GOSUB 1550:
    GOTO 400
660 IF a(k,d) > 1 THEN 680
670 IF d < 7 THEN 630 ELSE 680
680 d=d-1
690 a(k,d)=a(j,p)
700 a(j,p)=1
710 c=c+1
720 f1=0
730 d=7
740 z=0: GOTO 240
750 REM Come here at end of game
760 FOR s=10 TO 250 STEP 10
770 SOUND 1,s,5,15: SOUND
    3,s,5,15
780 NEXT
790 PEN 1
800 LOCATE 1,22: PRINT "You have
    completed";: PRINT "the
    TOWER"
810 FOR I=1 TO 400: NEXT: g$=""
820 GOSUB 950
830 IF g$="Y" OR g$="y" THEN
    CLEAR: GOTO 120 ELSE
    GOSUB 1070: END
840 REM Instructions
850 PEN 3: CLS
870 b$=rat$+"TOWER OF HANOI"
880 RESTORE 1220
890 FOR sen=1 TO 6
900 READ s$(sen),a
910 LOCATE 1,a: PRINT s$(sen)
920 NEXT sen

```

```

930 IF INKEY$="" THEN 930
940 RETURN
950 REM End
960 CLS:GOSUB 1240
970 FOR I=1 TO 100: NEXT
980 PEN 8: LOCATE 1,3: PRINT
    "You took"; GOES; "goes";:
    PRINT "to build the tower"
990 PEN 1: IF GOES < 66 THEN
    LOCATE 1,8: PRINT "A
    BRILLIANT SCORE!!!"
1000 IF GOES > 90 THEN LOCATE
    1,8: PRINT "Try harder kid!"
1010 IF GOES > 66 AND GOES < 90
    THEN LOCATE 1,8: PRINT "O.K!
    An average score"
1020 WHILE g$ <> "Y" AND
    g$ <> "y" AND g$ <> "n" AND
    g$ <> "N"
1030 LOCATE 1,20: PRINT "Another
    game?";: PRINT "(Y or N)"
1040 g$=INKEY$
1050 WEND
1060 RETURN
1070 REM Bye
1080 CLS
1090 FOR b=1 TO 30
1100 PRINT "Good Bye";
1110 IF count > 15 THEN PRINT
1120 NEXT b
1130 END
1220 DATA Your task is to transfer the
    rings, 6, one by one onto one of
    the other, 8, pegs. You cannot
    place a bigger ring, 10, on a
    smaller one and the computer,
    12, will tell you so if you try it, 14

```

```

1230 DATA Press any letter to start
      the game, 20
1240 REM Title
1260 title$="TOWER OF HANOI":
      CLS: c=3
1270 FOR p=1 TO LEN (title$)
1280 PEN c: c=c+1: IF c > 15 THEN c=3
1290 LOCATE p+2,6: PRINT MIDS$
      (title$,p,1)
1300 SOUND 1,p*8,4,15
1310 FOR I=1 TO 91: NEXT
1320 NEXT
1330 RETURN
1340 REM Char
1350 SYMBOL AFTER 225
1360 SYMBOL 225,255,255,255,
      255,0,0,0,0
1370 SYMBOL 226,60,60,60,60,60,
      60,60,60
1380 SYMBOL 227,255,255,255,
      255,255,255,255,0
1390 SYMBOL 228,240,240,240,
      240,240,240,240,0
1400 SYMBOL 229,15,15,15,15,15,
      15,15,0
1410 SYMBOL 230,15,15,15,15,15,
      15,15,15
1420 SYMBOL 231,63,63,63,63,63,
      63,63,0
1430 SYMBOL 232,192,192,192,
      192,192,192,192,0
1440 SYMBOL 240,48,91,255,63,4,

```

```

      8,48: SYMBOL 241,0,0,226,
      253,249,33,16,8: rat$=CHR$
      (240)+CHR$ (241)
1450 RETURN
1460 REM Rings
1470 Sp$=" ": a$(1)=sp$+sp$+
      CHR$ (230)+STRING$ (3,sp$)
1480 a$(2)=sp$+sp$+CHR$
      (231)+CHR$ (232)
1490 a$(3)=sp$+sp$+CHR$
      (227)+CHR$ (228)
1500 a$(4)=sp$+CHR$ (229)+CHR$
      (227)+CHR$ (227)
1510 a$(5)=sp$+STRING$ (3, CHR$
      (227))+CHR$ (228)
1520 a$(6)=CHR$ (229)+STRING$
      (4, CHR$ (227))
1530 a$(7)=STRING$ (5,CHR$ (227))
      +CHR$ (228)
1540 RETURN
1550 REM invalid
1560 FOR L=1 TO 100: NEXT
1570 LOCATE 1,4: PRINT "YOU
      CANNOT DO THAT!"
1580 SOUND 1,3,25,15
1590 FOR s%=170 TO 60 STEP -15
1600 SOUND 1,s%,1,15
1610 NEXT
1615 LOCATE 1,4: PRINT"
      "
1620 RETURN

```





# Glossary of commands

This section of the book contains a list of all the commands used in the book, plus a few more you might find useful. Some explanations include a program line or even a short program to help you understand what the commands does.

**AUTO** tells the computer to AUTOMATICALLY print line numbers. AUTO on its own starts at 10 and goes up in tens. AUTO 200,20 starts at line 200 and goes up in steps of 20. Press ESCAPE to stop AUTO renumbering.

**AUTO < ENTER >**

**CHAIN""** this command LOADs a program from cassette, then RUNs it. CHAIN"" chains the next program on the tape, or the name of the program can be put inside the quote marks, e.g.

**CHAIN"GUNSHOT"  
< ENTER >**

**CHR\$** pronounced 'character string' is

used to PRINT a symbol, or character on the screen, e.g., CHR\$(97) is the letter A. PRINT CHR\$(13) is the same as pressing the ENTER key. CHR\$ can also be used to print user-defined characters.

**10 PRINT CHR\$(97)**

**CLS** CLears the Screen and sends the cursor back to the top of the screen, e.g.

**50 CLS**

**CURSOR** is not a command, but the coloured square that tells you where the computer is to PRINT next on the screen.

**DRAW** draws a line on the screen from the last point visited to the point given after the command.

**10 MODE 1  
20 WHILE x=0 AND y=0  
30 x=INT (RND(1)\*400+1):**

```

      y=INT(RND(1)*640+1)
40 DRAW x,y
50 WEND

```

ELSE

is used in an IF/THEN statement. IF the IF/THEN statement is UNTRUE the information after the ELSE will be used by the computer.

```

10 IF n=0 THEN PRINT "Do not
   use zero" ELSE PRINT "That
   was O.K."

```

END

is used to tell the computer that the program is to END and return to BASIC.

```

100 PRINT "GOODBYE"
110 END

```

FOR...  
NEXT

a loop which makes the computer repeat the program lines between the FOR and the NEXT a set number of times. In Amstrad BASIC it is not necessary for the NEXT to have the VARIABLE after it.

```

10 FOR n=1 TO 30
20 PRINT "Loop ";n
30 NEXT

```

GOTO

tells the computer to jump to a certain line number. Try not to use GOTO too much because it makes the program difficult to follow.

When you learn how to use GOSUB you will find GOTO is not needed so much.

```

130 MODE 1
140 PRINT "Again"
150 PRINT "and again!"
160 GOTO 150

```

IF...THEN

tells the computer to make a decision. If the answer is true (Yes) then the computer obeys the rest of the IF/THEN line. If the answer is false (No – not true) the computer moves onto the next program line. Can be used with ELSE (See ELSE).

```

10 INPUT "Please type in a
   number ";number
20 IF number=10 THEN
   PRINT "HELLO"

```

INK

Depending on the Screen Mode, a number of INKs are available. The colour or colours used for an INK may be changed by an INK command, according to the table of colour values in Chapter 8.

INKEY(n)

tests the keyboard to see if any keys have been pressed. The number in the brackets relates to a key. See page 16 Appendix III of your User Instructions. For example, INKEY(47) tests for the space bar.

```

10 MODE 1
20 IF INKEY(47)=0 THEN
    SOUND 1,200,11,15
    ELSE GO TO 20

```

INKEY\$

waits for a key to be pressed on the keyboard.

```

100 PRINT "Do you want to
    play again?"
110 PRINT "Press Y or N"
120 WHILE g$ <> 'Y' AND
    g$ <> 'N' AND g$ <> 'y'
    AND g$ <> 'n'
130 g$=INKEY$
140 WEND
150 RUN

```

INPUT

stops the program and waits for something to be typed in. INPUT requires <ENTER> to be pressed before continuing.

Messages can be included in INPUT lines. The number or letters typed in are stored in a STRING VARIABLE (e.g. 'A\$') or NUMERIC VARIABLE (e.g. 'age') following the INPUT line 'prompt' message (if any).

```

10 MODE 1
20 INPUT "How old are you ",
    age
30 IF age > 2 PRINT "Cool!
    That's old. I am not even 1
    year old." ELSE PRINT

```

```

"I was invented in 1984."
40 PRINT "I wonder if I shall
    look like you when I
    am ";age

```

LEN

works out how many characters there are in a string (or how many letters are in a word stored in memory)

```

10 MODE 1
20 INPUT "Type any word ",
    word$
30 length=LEN(word$)
40 PRINT "That word was ";
    length;" letters long."

```

LIST

prints the program in memory onto the screen. LIST 100 will list line 100 only. LIST 100-200 will list lines 100 to 200. LIST 300- will list all lines after 300. It does not need a line number. To stop a program whizzing up the screen you can press the ESC key once. Press any key to continue listing. Press ESC key twice to stop listing.

```
LIST 1000-1050 <ENTER>
```

LOAD

is used to load a program from cassette or floppy disc. LOAD"" will load the first program found; this will not work with a disc drive. LOAD"program name" loads the named program. This

|               |                                                                                                                                                                                                                                               |       |                                                                                                                                                                     |
|---------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|               | will work with cassette or disc drive. It does not need a line number.                                                                                                                                                                        |       | position across and down the screen. See <i>Chapter 13</i> .                                                                                                        |
| LOCATE<br>x,y | is used with PRINT and INPUT to move words to where you want them on the screen. The column number across the screen (x) comes first, followed by a comma and the number of lines down the screen (y). See <i>Chapter 11</i> on using LOCATE. |       | <pre> 10 MODE 1 20 MOVE 320,200:REM centre   of screen 30 PLOTR 0,0 </pre>                                                                                          |
|               | <pre> 10 MODE 0 20 PRINT "WHAT IS   YOUR NAME?" 30 INPUT NAMES\$ 40 CLS 50 LOCATE 8,10 60 PRINT NAMES\$ 70 LOCATE 20,20 80 PRINT "WHAT A   NICE NAME!" </pre>                                                                                 | NEW   | removes the old program from the computer's memory.                                                                                                                 |
|               |                                                                                                                                                                                                                                               |       | <b>NEW &lt; ENTER &gt;</b>                                                                                                                                          |
|               |                                                                                                                                                                                                                                               | PAPER | sets the background ink for characters. When characters are written to the text screen, the character is filled with the PAPER ink before the character is written. |
| MODE          | The Amstrad has 3 MODEs, numbered 0 to 2 which select the type of screen display and number of colours available. See <i>Chapters 8</i> and <i>12</i> .                                                                                       |       | <pre> 10 MODE 0 20 FOR P=0 TO 15 30 PAPER P: CLS 40 PEN 15-P 50 LOCATE 6,12: PRINT   "PAPER" 60 FOR t=1 TO 500:   NEXT 70 NEXT P </pre>                             |
|               | <b>20 MODE 2</b>                                                                                                                                                                                                                              |       |                                                                                                                                                                     |
| MOVE          | is used in the graphics MODEs to move to a point on the screen. It is followed by two numbers separated by a comma, giving the                                                                                                                |       | <pre> 10 MODE 0 20 PEN 2 30 PRINT "AMSTRAD" </pre>                                                                                                                  |
|               |                                                                                                                                                                                                                                               | PEN   | sets the ink to be used when drawing at the given screen stream.                                                                                                    |

PLOT

is the same as DRAW except that only the pixel at the specified co-ordinate is written.

PRINT

tells the computer to write something onto the screen. If two printed messages, or numbers, are separated by a semi colon (;) they will be printed next to each other.

```
10 MODE 1
20 PRINT "Hi there"
30 PRINT "Hi there"
40 PRINT "Hi ";" there"
```

REM

lets you place comments in the program. The computer ignores everything on a program line after it comes across the REM command, so you must not add commands that you want it to carry out on the same line after a REM.

```
10 REM This is my program.
20 GOTO 20
```

RND

selects a random number up to the limit set in the brackets.  $\text{INT}(\text{RND}(1)*10)+1$  gives you a whole number between 1 and 10.

```
10 WHILE number < 20
20 number=INT(RND(1)
   *20)+1
30 PRINT number
40 WEND
```

RUN

makes the computer carry out the program in the memory. It does not need a line number.

```
RUN < ENTER >
```

SAVE

saves the program in memory on cassette or disc. You may give the program a name of up to sixteen letters (only seven if you are saving onto a disc). The program name must have quote marks (") around it. It does not need a line number.

```
SAVE "Invaders 2"
< ENTER >
```

SOUND

is used to make sounds. It is followed by four numbers. The first one is the *sound channel*, the second the *volume* (15 to 0), 15 is the loudest. The third number is the *pitch* and the fourth is the length of the sound, or '*duration*'.

```
10 SOUND1,15,INT(RND(1)
   *255)+1,5
```

SPACE\$

It prints a number of spaces on the screen.

```
10 MODE 1
20 PRINT "Hi";SPC(32);"there"
```

**TAG** this command allows text and symbols to be mixed with graphics.

**TAGOFF** cancels the TAG for a given stream.

**WHILE/WEND** is another way of making a loop. The computer repeats certain program lines until a condition is true.

```

10 MODE 1
20 WHILE number < 57
30 number=INT(RND(1)*
100)+1

```

```

40 PRINT number
50 WEND
60 PRINT "I have finally
selected number
";number

```

**WINDOW** sets a text window for a given screen stream.

```

10 MODE 1
20 WINDOW 10,30,8,20
30 PAPER 2: PEN 3
40 CLS
50 PRINT "WINDOW ONE"
60 GOTO 60

```



## Index

|            |                                      |        |                                   |            |                                                 |
|------------|--------------------------------------|--------|-----------------------------------|------------|-------------------------------------------------|
| CHAIN      | 27, 38                               | INK    | 49, 53                            | PEN        | 47-50, 53                                       |
| CLS        | 9, 12, 16-7, 32, 41-2, 44, 48-51, 63 | INKEY  | 69-70, 78-9                       | PRINT      | 9-12, 15-9, 22, 30, 32, 39, 53, 62, 72          |
| CURSOR     |                                      | INPUT  | 28-31, 33, 52-3, 59, 62, 64, 69   | RAM        | 19                                              |
| CONTROL    | 42, 45-6                             | LIST   | 9, 18, 25, 31, 35, 37, 39, 44, 46 | REM        | 40-1, 46, 68, 72-3, 77                          |
| DELETE     | 7-9, 11, 14, 43-4                    | LOAD   | 34, 37-8, 80                      | RND        | 60, 64                                          |
| DRAW       | 75-79                                | LOCATE | 65-67, 69-71, 74                  | RUN        | 148, 25, 28, 31-2, 38, 44, 49, 54-5, 71, 73, 75 |
| END        | 9, 30                                | MODE   | 47-9, 54, 65-6, 69, 72, 74-5, 78  | SAVE       | 32, 34, 36-40, 42, 45, 80                       |
| ESCAPE     | 15-6, 18, 36, 51                     | MOVE   | 75-6, 79                          | SHIFT      | 7, 10-1, 43-6                                   |
| FOR...NEXT | 49, 51-2, 55, 57-8, 72               | NEW    | 14, 18, 49, 51                    | SOUND      | 61, 63                                          |
| GOTO       | 15-8, 51, 55, 71                     | PAPER  | 47-8, 50, 73                      | WHILE/WEND | 56-8, 62, 64                                    |
| IF...THEN  | 59-64, 70                            |        |                                   |            |                                                 |



# Answers

## Mindbender 3.1

Assuming that you have haven't switched the machine off since typing in the commands:

**LET cars = 10**

and

**LET trains = 5**

you simply type in the additional information:

**LET planes = 3**

If you now type

**PRINT models**

the computer will give you the wrong answer! This is because it still thinks that

**models = cars + trains**

whereas your collection now includes cars, plus trains plus planes. So you also have to change the formula by typing in:

**LET models = cars + trains + planes**

Now, when you type:

**PRINT models**

you should get the correct answer— 18.

## Mindbender 3.2

Easy, this one! Just type

**LET singles = 12**

**LET albums = 8**

**LET records = singles + albums**

then, to check the answer, type

**PRINT records**

If you've used the same numbers as me, you should get the answer 20. But you could just as easily substitute your own numbers to get a different answer.

Note that you could also use different names for the variables and still get the right answer. For example, you could call the seven-inch singles 'sevens' and the twelve-inch albums 'twelves', provided you use the same names for the formula line, e.g.:

```
LET sevens = 12
LET twelves = 8
LET records = sevens + twelves
PRINT records
```

If you are really lazy (or hate typing), you could even abbreviate the variable names to a single letter, e.g. s for 'singles', a for 'albums', and r for 'records'.

You can also leave out the word LET, which isn't necessary on the Amstrad (though it is on some computers, including the Sinclair Spectrum). Your answer would then be:

```
s=12      < ENTER >
a=8       < ENTER >
r=s+a     < ENTER >
PRINT r   < ENTER >
```

While this gives a perfectly correct answer with very little effort, I don't recommend you to do it this way just yet as you can easily forget what the single letters stand for. Stick to full words that help to explain what you are doing.

#### Mindbender 4.1

There were a couple of red herrings in this one to trick you! The answer is to use *numeric variables* instead of string variables, because you can't do calculations, with string variables.

```
10 cars = 5
20 dolls = 7
30 toys = dolls + cars
```

```
40 PRINT "Total number of toys = ";
    toys
50 END
```

All you had to do was to delete the dollar signs (\$) and the quote marks round the numbers. There is no need to use the word LET, though it would not be wrong to do so. The semi-colon in line 40 is not strictly necessary, so you are not wrong if you missed it out.

In line 30 it makes no difference if you put 'cars' before 'dolls'.

If you deleted line 20 and made line 10 read

```
10 cars = 5:dolls = 7
```

you are a programming genius and can award yourself 200 bonus points!

#### Mindbender 7.1

Just add these lines (You may need different line numbers to those shown, depending on how you ended up after using RENUM.)

```
200 LET minutes = hours * 60
210 PRINT "or a staggering ";
    minutes;" minutes,"
220 LET seconds = minutes * 60
230 PRINT "which works out at a
    mind-boggling "
240 PRINT; seconds;" seconds!!!"
```

Don't worry if the wording of your PRINT statements isn't the same as ours – it's the



PRINCIPLE (how to do it) that matters!

Note that for anyone over 32 years old, the number of seconds is too big for the Amstrad to handle, so it has to use EXPONENTIAL NOTATION!

### Mindbender 7.2

```
10 REM *** LIREX ***
20 LET rate = 2395:CLS
25 PRINT:PRINT
30 INPUT "How many pounds do you
  wish to change ";pounds
40 LET lire = pounds * rate:CLS
45 PRINT:PRINT
50 PRINT "For ";pounds;" pounds
  sterling, you will "
60 PRINT "receive approx. ";lire;" lire
  in Italy"
70 PRINT "at the exchange rate of ";
  rate;" lire"
80 PRINT " per pound sterling."
```

### Mindbender 7.3

For France, change the following lines in the above program to read:

```
10 REM *** FRANCEX ***
20 LET rate = 11.9:CLS
40 LET francs = pounds * rate:CLS
60 PRINT "receive approx. ";francs;
  " francs in France"
70 PRINT "at the exchange rate of ";
  rate;" francs"
```

You can now see how much easier it is to EDIT programs, rather than having to re-type whole lines when only minor changes are needed!

### Mindbender 8.1

The lines in MODE 2 only printed in yellow because the other INK colour (blue) is the same colour as the PEN (background). Therefore only the yellow line can be seen. This is also the reason why all the even pen numbers are missed out.

### Mindbender 8.2

```
10 MODE 0
20 PRINT "AMSTRAD CPC 464"
30 AMSTRAD = AMSTRAD + 1
40 PEN AMSTRAD
50 LOCATE 1,20: PRINT AMSTRAD:
  LOCATE 1, AMSTRAD
60 FOR N = 1 TO 500: NEXT
70 IF AMSTRAD < 15 GOTO 20
```

### Mindbender 9.1

```
10 MODE 1
20 FOR a=1 TO 15
30 PRINT "Hello, Jim"
40 NEXT a
```

### Mindbender 9.2

```
10 REM *** Shooting stars ***
20 MODE 1
30 column = 2
40 WHILE column < 39
50 LOCATE column,12: PRINT "*"
60 LOCATE column-1,12: PRINT " "
70 column = column + 1
80 WEND
```

### Mindbender 9.3

Add the following lines to the above program:

```
35 START = TIME
90 PRINT "Time = "; (TIME-start)/
300;" seconds"
```

When you RUN the program and compare the TIME with that of the FOR/NEXT loop, you may find the difference surprising.

However, you can't draw absolute conclusions from only one test. There are times when one type of loop will be the obvious choice.

### Mindbender 9.4

```
10 MODE 1
20 PRINT "The twelve times table"
30 FOR n=1 TO 12
40 PRINT n;" x 12 = ";n*12
50 NEXT n
```

### Mindbender 9.5

```
10 MODE 2
20 FOR n=1 TO 200
30 PRINT "Fred ";
40 NEXT n
```

### Mindbender 9.6

```
10 MODE 1
20 START = TIME
30 FOR n=1 TO 1000
```

```
40 PRINT n
50 NEXT n
60 PRINT "TIME= ";(TIME-START)/
300;" seconds"
```

### Mindbender 10.1

The new 'Score so far' line should be inserted between existing lines 110 and 120 – say, Line 115. It has to appear after 'right=right+1' has been worked out, e.g.

```
115 PRINT "score = "; right;"
out of "; goes
```

To stop the 'Press any key' message appearing, you need something like:

```
112 IF goes = 10 THEN GOTO 150
```

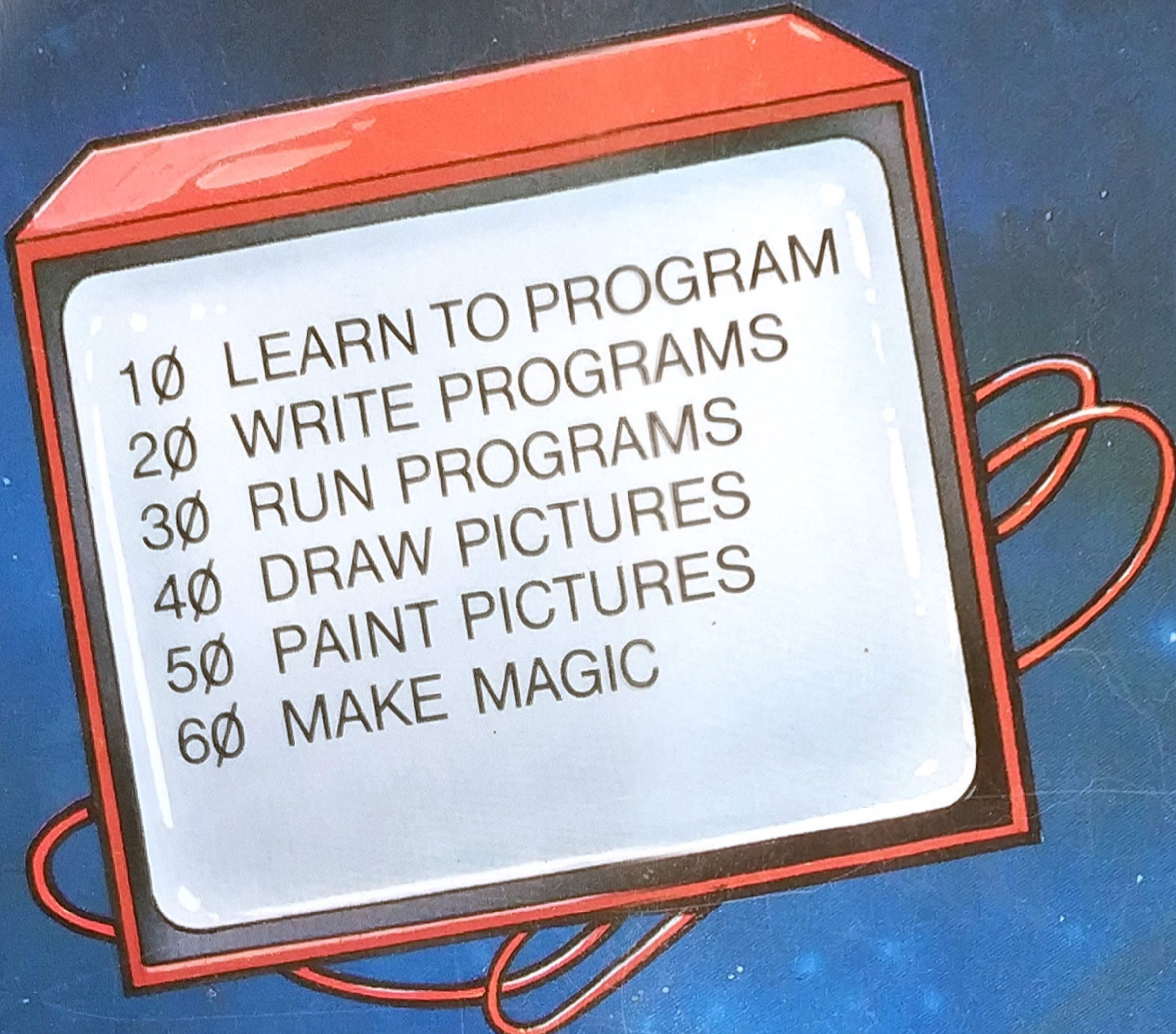
You could place it after line 115, say at line number 117 – but this wouldn't be quite as effective. On the Amstrad, you should really close loops before you 'jump out' of them. So, strictly speaking, the line should read GOTO 140, rather than GOTO 150.

A CLS should go between lines 130 and 140 to clear the screen after you have read your result on the screen and pressed a key to indicate that you want the next problem, e.g.

```
135 CLS
```



£5.95  
net

- 
- 10 LEARN TO PROGRAM
  - 20 WRITE PROGRAMS
  - 30 RUN PROGRAMS
  - 40 DRAW PICTURES
  - 50 PAINT PICTURES
  - 60 MAKE MAGIC

ISBN 0-572-01297-7



9 780572 012977

AMERICAN STANDARD FIRST PRINCIPLES OF BOOK BINDING