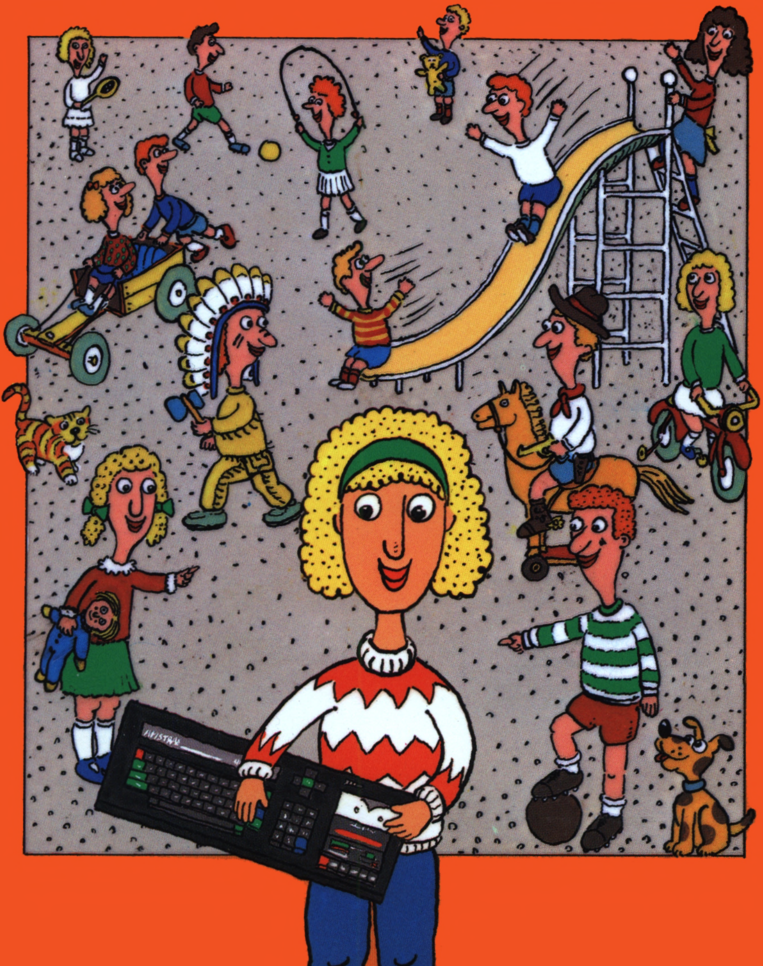
MY AMSTRAD CPC 464 AND ME

JACK WALKER



My Amstrad CPC 464 and Me

Jack Walker

Duckworth

First published in 1985 by
Gerald Duckworth & Co. Ltd.
The Old Piano Factory
43 Gloucester Crescent, London NW1

ISBN 0 7156 1964 0

British Library Cataloguing in Publication Data

Walker, J.
My Amstrad computer and me. — (My computer and me)
1. Amstrad Microcomputer — Juvenile literature
I. Title II. Series
001.64'04 QA76.8.E/

ISBN 0-7156-1964-0

Typeset by Commercial Colour Press,
Forest Gate, London E7.
Printed in Great Britain by
Redwood Burn Ltd., Trowbridge
and bound by Pegasus Bookbinding, Melksham

INTRODUCTION

This book is for children and for total beginners. It does not attempt to go beyond the absolute minimum necessary to understand the principles of programmming. I have tried to explain things as thoroughly and as simply as I can. I hope that I have succeeded in avoiding the assumption that after the first three or four pages the reader is miraculously transformed into an expert.

The difficulty with programming languages is not one of essence but of detail. There are a lot of words to understand and learn and put together. That is a good thing, because a good vocabulary can produce powerful sentences. But a collection of words is not necessarily a vocabulary. This book considers the most important words which happen (as is the case in any sort of language) to be the simplest. It deals not only with words but with the ideas that inform programming:

Actions can happen one after another, in sequence.

Actions can be repeated. They can be repeated a fixed number of times. Or they can be repeated until something happens to make them stop. If that something never happens they will keep repeating.

Actions can be aimed in one direction or another depending on some condition.

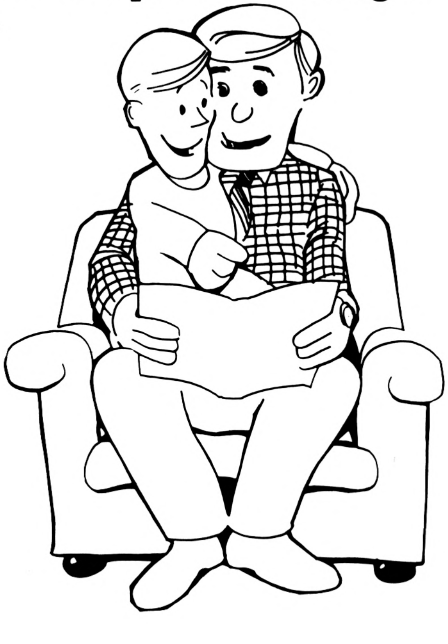
Actions act on objects.

A computer program is a collection of actions on objects. The objects are given to the actions. The actions process the objects to produce new objects. These objects may be processed by other actions in the program or they may be shown as the end results of the program.

One of the major ideas that this book does not deal with is that of recursion. It is not because this idea is particularly difficult, but because it comes into its own for programs that are more complex than any in this book.

An hour a day with the book, reading and acting on it, is a program for grasping how really simple and powerful the ideas of programming are.

The best possible readers I could hope for are children and parents reading the book together.

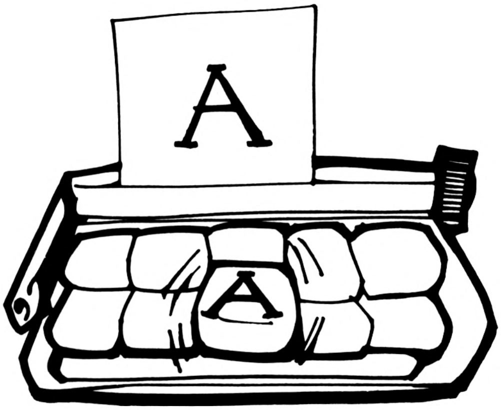


IT LOOKS FAMILIAR!

When you first look at your Amstrad, you might tell yourself, "This looks just like a typewriter!"

Have you ever used a typewriter? You first put a blank sheet of paper in it. You type and you see that words get PRINTED on the paper. Does the sheet of paper have to be white? Not really. Does the typewriter ribbon have to be black? Not really. You could have red ink on black paper, black ink on white paper, and so on. What would happen if you had ink the same colour as the paper? If you typed your name in red ink on red paper, would you be able to see your name?

Have you thought what happens when you type the letter A on a typewriter?

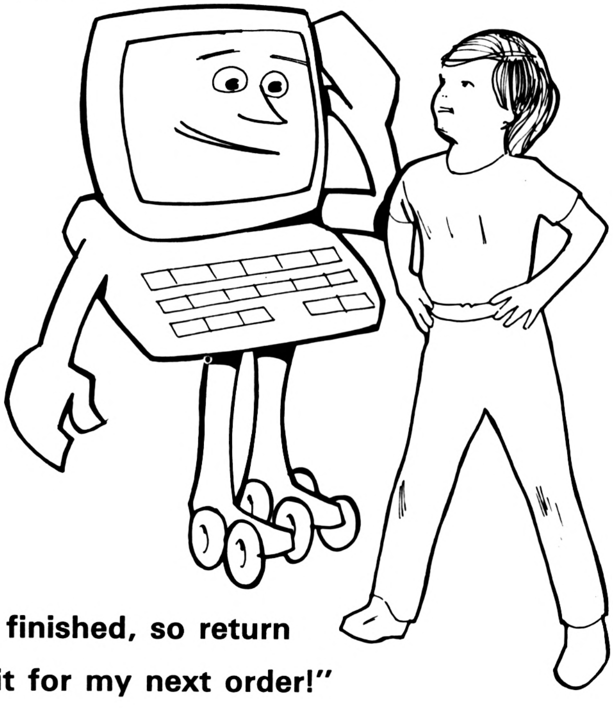


When you hit the A key, this makes the typewriter move levers and springs, an arm comes up and strikes the ribbon and the letter A is printed on the paper.

If you type the letter A on the keyboard of your Amstrad, the Amstrad obeys your command to print the letter A on the screen. But, of course, it uses a different method to the one used by the typewriter. The Amstrad has electronic chips inside it instead of springs and levers. It shows things on the screen in the same way as a TV set does.

Is your Amstrad switched on and connected?

Notice the Ready sign. It tells you that the Amstrad is ready for you to type instructions and orders into it.



"I've finished, so return and wait for my next order!"

Type in your first name. Now, the way you tell the Amstrad that you have finished an instruction is by pressing the ENTER key. So press ENTER.

The Amstrad gives you a message:

```
Syntax error.
```

That's its rather cheeky way of telling you that it doesn't understand what you want it to do.

That's not really surprising, when you think that your Amstrad is really only a machine. Your TV set gives marvellous moving pictures and sound, but it needs to be switched on by you. You have to change the channels. You have to turn the brightness, volume, contrast and sound controls.

So you also have to give your Amstrad instructions in a special language that it can understand. This language is called BASIC.

One of the words in this language is PRINT.

Type in:

```
PRINT"Amstrad"
```

Press ENTER to tell the Amstrad that you've finished your instruction. What do you see on the screen? What would you do if you wanted to PRINT your first name instead of AMSTRAD?

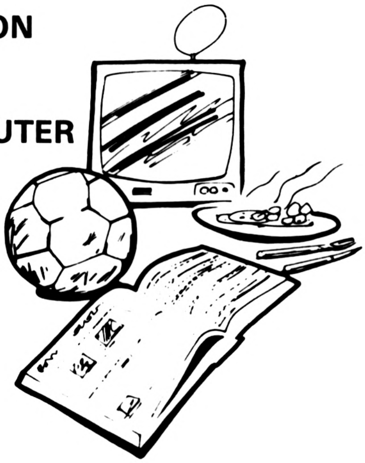
At this stage, press CAPS LOCK and keep this key in operation for the rest of the work you do with this book.

DOING THINGS AT ONCE AND DOING THINGS LATER

When you are home at the weekend, you may suddenly decide to go out and play football. After that, you may decide to read a book. You can keep on deciding to do things as you think of them, without having a definite plan. But you can instead think ahead and make a list of what you want to do, one thing after another. If you keep a list, you can look at it later and see what you have to do.

Suppose this is your list:

```
10 PLAY FOOTBALL
20 READ A BOOK
30 WATCH TELEVISION
40 HAVE DINNER
50 PLAY WITH THE
   AMSTRAD COMPUTER
60 END
```



Why do you think the numbers go up ten at a time? Let's try and think why.
Suppose you want to have a shower after playing football. Then you can write:

```
15 HAVE A SHOWER
```

You can see that having the numbers go up by ten makes it easier to put in something else in your list of things to do.
Notice how you tell yourself to finish doing the things on your list. You put the instruction in Line 6Ø.
Now, you can command the Amstrad to do things at once:

```
PRINT"AMSTRAD"
```

(Don't forget to press the ENTER key after the command.)
You can, instead, make a list of commands for the Amstrad to obey:

```
1Ø MODE Ø
2Ø PRINT"AMSTRAD"
3Ø END
```

Don't forget to press the ENTER key after typing line 1Ø and after typing lines 2Ø and 3Ø.
When you make a list of instructions for the computer to follow, the computer will remember them. But it won't act on them immediately. It will follow the instructions only when you order it to.
You give it the order by typing RUN.
The list of instructions is called a PROGRAM.

If you want to tell the computer that you are going to give it a new program, type NEW.

So, now type NEW.

Now type in this program:

```
10 MODE 1
20 PRINT"AMSTRAD"
30 END
```

Notice that Line 10 is different to the Line 10 you typed in before.

Now type RUN.

FAT LETTERS AND THIN LETTERS

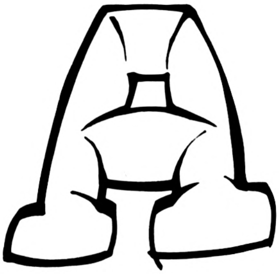
Notice that the word AMSTRAD appears on the screen in a different size for each program that you typed in.

Why is that?

It's because of the word MODE.

MODE tells the Amstrad to divide the screen up into little blocks. Of course, you can't actually see these little blocks, but the Amstrad divides the screen up in its memory.

When you tell the Amstrad to divide the screen up into MODE Ø blocks, the letters it shows on the screen are fatter than the letters it shows when it is in MODE 1.



Let's look at the way the letter A is shown in MODE Ø.

First, type NEW.

Now, enter this new program:(By the way, don't confuse the number Ø with the letter O. The number Ø has a little slanting line through it and is near the top right-hand corner of the keyboard.)

```
10 MODE Ø
20 PRINT"A"
30 END
```

Now type RUN. Notice how the A shape is made up of little dots.

Now let's look at the way the letter A is shown in MODE 1.

First, we have to command the Amstrad to go to MODE 1.

Does this mean we have to type in a whole new program?

Not really. First, type LIST. This will show you all the lines of your program.

The only line you want to change is Line 1Ø.

So, all you have to do is type in:

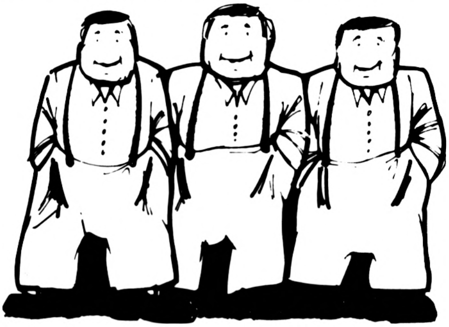
```
10 MODE 1
```

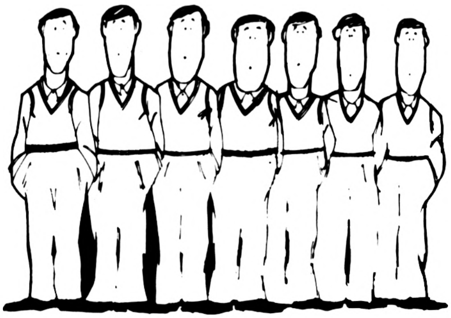
The screen is looking a bit messy now. So clean it up by typing CLS. You haven't lost the program. Type LIST and you'll see it again.

Now type RUN.

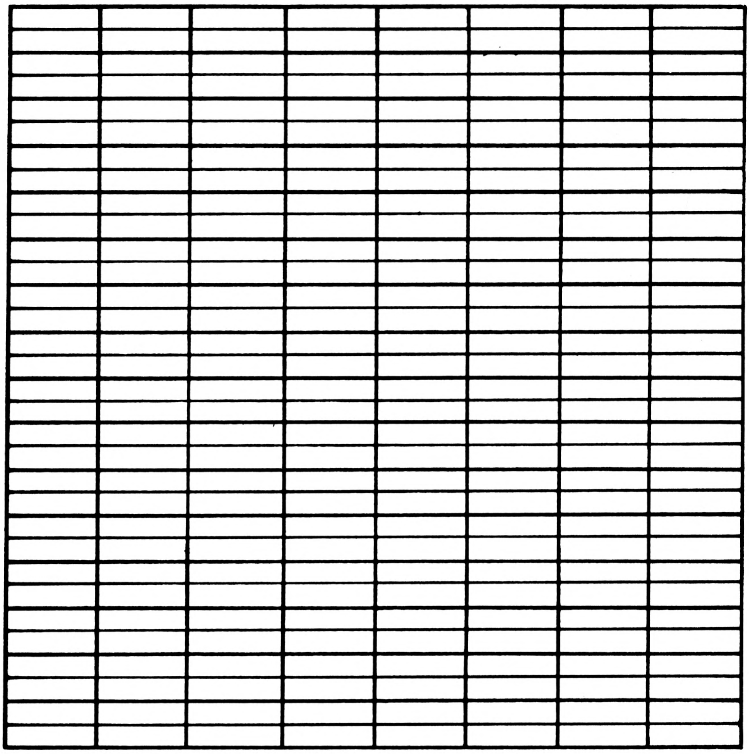
Notice how the letter A is not as fat as the letter A you saw for MODE Ø.

That's because the little dots making up the letter A in MODE 1 are thinner than the dots in Mode Ø, so it looks thinner.





Suppose some fat men try to squeeze into a telephone box. Then suppose some thin men do the same. You can see that more thin men will fit in. In the same way, more letters will fit on the screen in MODE 1 than in MODE Ø. In Mode 1 the Amstrad can fit 4Ø characters across one line of the screen. It can only fit 2Ø characters in MODE Ø.



By the way, the little dots that make up a letter or a picture on the screen are called PIXELS. The fatter the pixels a letter is made up of, the fatter the letter is. The fatter the letters are, the fewer of them will fit onto the screen. It's just like the fat and thin men and the telephone box.

You can not only put letters on the screen.

The pixels are dots. So you can order the Amstrad to join the dots together to draw lines, and triangles, and circles; and even pictures. Of course, you have to give the Amstrad careful instructions to do these things. As you practise more and more, you will learn how to write bigger and bigger programs.

The Amstrad can draw letters and pictures on the screen in three different MODES. You've looked at MODE Ø and MODE 1.

You can easily see what the letter A will look like in the remaining MODE.

First, type LIST to look at Line 1Ø.

```
1Ø MODE 1
```

You just want to change the 1.

So, press the SHIFT key and hold down.

Use the upward-arrow key that's near the COPY key to shift the ■ sign (it's called the COPY CURSOR) to go on the 1 of Line 1Ø. Now, press the COPY key carefully until you see the copy cursor positioned over the 1. Now type in 2. Then press ENTER.

Notice that so long as you press COPY, you copy what's in Line 1Ø. If, instead, you type something without pressing COPY, you add something new to Line 1Ø.

Now you can RUN your program and see what the letter A looks like in MODE 2.

NUMBERS AND CHARACTERS

Let's go back to the program we started with. Type in:

```
10 MODE 1
20 PRINT"AMSTRAD"
30 END
```

Look again at line 20:

```
20 PRINT"AMSTRAD"
```

Why can't we just say AMSTRAD, without the " marks? If you want, try it and you'll see that the Amstrad thinks it is a number and prints a zero.

That's because the Amstrad has to know whether you are talking about a *character* or a *number*.

Numbers can be added, or subtracted or multiplied or divided.

1+2=3
4−2=2
4 * 2=8 (* means *multiply* for the Amstrad. Look for it on your keyboard.)

4 / 2=2 (/ means *divide* for the Amstrad. Look for it on your keyboard.)

Characters are all the things that are not treated as if they are numbers.

"AMSTRAD" is a collection of characters. You can't treat it the way you can a number. You can't do things like addition and subtraction to it.

The " marks tell the Electron that it is dealing with characters.

But let's look at 1984.

1984 looks like a number but it can also be treated as if it is a collection of characters. It depends on what you mean when you say 1984.

If you mean the year 1984, then it's a collection of characters. If it is how many pounds you have in the bank, then it's a number.

Your name is a set of characters. You can't multiply your first name by your surname to give a new name. But you can multiply ten by two to give twenty.

WHAT'S HAPPENING INSIDE?

Here, once again, is the list of instructions for what you might do on a Saturday:

```
10 PLAY FOOTBALL
15 HAVE A SHOWER
20 READ A BOOK
30 WATCH TELEVISION
40 HAVE DINNER
50 PLAY WITH THE AMSTRAD
60 END
```

You write these instructions on a piece of paper. You can't write them on air! You need somewhere — a piece of paper — to put the instructions. If you write down even more instructions, you will need more paper.

In the same way, when you enter a program into the Amstrad, it is kept inside the space the Amstrad uses to store the instructions you give it in your program. The larger your program is, the more space it will need.

This space is in the computer's memory. You can think of it as being made up of tiny little brain-boxes.

Does that mean that if you could look into the tiny brain-boxes in the computer's memory, you would see the same sort of letters and numbers as the ones you write on a piece of paper?

No, not at all.

Take two little bits of blank paper. On the first piece, write a 1 on one side and a Ø on the other side. Do exactly the same thing on the other piece of paper.

Now put the two pieces side-by-side. If you do this and then turn one piece over, and then the other, you will get the following patterns:

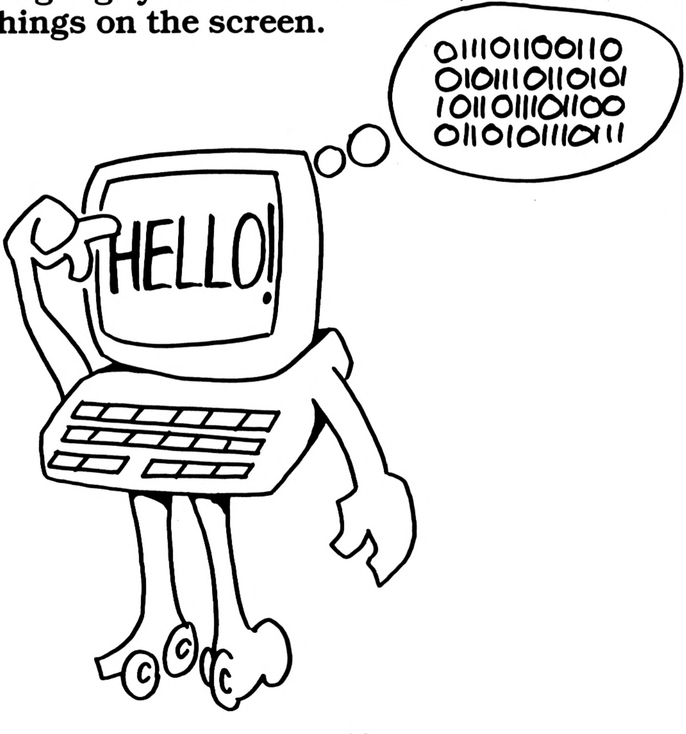
```
11
1Ø
Ø1
ØØ
```

If you tell yourself that you are an Amstrad, and that each of the patterns means something to you, you will get an idea of how the Amstrad stores information in its memory.

For example, you can tell yourself that the pattern 11 means the letter P.

Every time you press a key, the Amstrad changes the letters into patterns like the ones above. Of course, you've only got four patterns above. That's because you used two pieces of paper. But if you took *eight* pieces of blank paper and then did exactly what you did above, you would get 256 different patterns! That's more than enough for the Amstrad to change what you type in into the patterns of Ø and 1 that it can understand.

Of course, you can't understand the language that the Amstrad uses inside itself. So the Amstrad very kindly changes it to language you can understand, when it shows things on the screen.



SOME THINGS REMAIN THE SAME AND SOME THINGS CHANGE

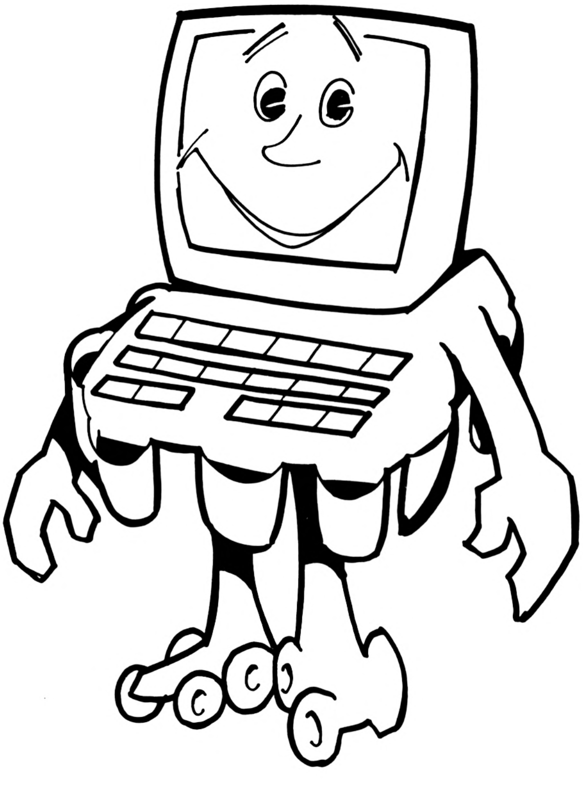
You were born with one nose. You will have one nose all your life (I hope!) When something doesn't change, it is called a CONSTANT. Can you think of other things that you could call constants?

See what's in one of your pockets right now. Maybe your pocket's empty! Or maybe it's got something in it — money perhaps, or a telephone number or the name of your favourite pop star. The contents of your pocket can change. Things that can change are called VARIABLES.

If your pocket contains things that can be added, subtracted, multiplied or divided, it contains NUMBER variables. For example, if your pocket contains five pence, you can add two pence to it to make seven pence. Or you can put 15 pence — three times five — into it.

If your pocket contains changeable things like telephone numbers or names or addresses, it contains what the Amstrad calls STRING variables. Remember that you can't treat these as if they are numbers. You can't take away your address from somebody else's address to get a new address!

You can imagine that the Amstrad has many, many pockets in its memory. It keeps number variables or string variables in its memory pockets. Of course, it can also keep things that don't change in its memory pockets.



A BIGGER PROGRAM

Now let's type in the same program that we used before:

```
10 MODEEE
```

I did that wrong, but I realised it before I pressed ENTER. I had to correct it, of course, but all I had to do was keep pressing the DELETE (the key marked DEL, top right of the keyboard) key to get rid of the last two letters E.

Next I typed:

```
20 PRINT AN "AMSTRAD"
```

Oh no! I didn't want the AN there. If I left it, the Amstrad would think I had typed in rubbish. So I had to correct it. I pressed ENTER. Then I held down the SHIFT key. I used the up-arrow to get on the 20. Then I pressed COPY until I got to the A. Then I used the right-arrow key to skip over the A and the N. Then I pressed the COPY key till I got to the end of Line 20, just after the " marks. Then I pressed ENTER.

Now I typed CLS and then LIST. That was tiring. But we all make mistakes, that's why it's so useful to be able to edit program lines, instead of having to type them out completely again. That would be even more tiring.

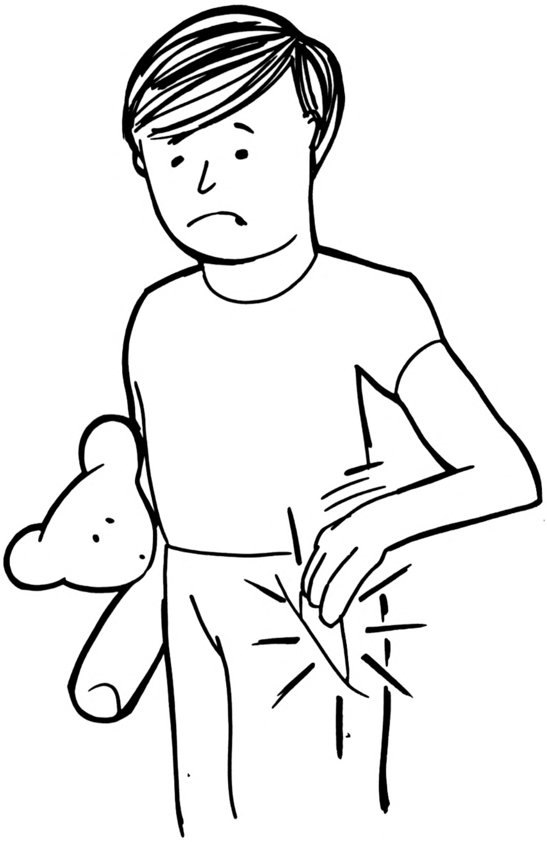
```
10 MODE 1
20 PRINT"AMSTRAD"
30 END
```

We are now going to change this little program a lot, and make it bigger.
First, we want to get rid of line 2Ø. All we have to do is type:

```
2Ø
```

Now press ENTER. That's how we can get rid of a whole line in a program.
Type CLS and then use LIST.
We want to tell the Amstrad to reserve one of its memory-pockets for a NUMBER variable. Now, one pocket looks just like another. So how can we tell if the pocket should contain a number or a string variable in it?
Suppose you wanted to reserve one of your pockets for money only. One way of doing it (but don't really!) would be to get a sticky label, write CASH on it, and stick it on a pocket.
If you want to reserve a pocket for putting names only into it, you could get a sticky label and write NAME$ on the sticky label, then put this on the pocket. The $ sign tells you that you are not dealing with numbers but with characters. You are dealing with STRING variables when you store names. You can't do arithmetic with names!
What would you do if you wanted to reserve a pocket for putting telephone numbers in? Is a telephone number a number that people add to other telephone numbers? Would PHONE$ be a nice sticky label to put over a pocket?

NUMBER VARIABLES



Let's pretend that your CASH pocket starts off with no money in it. Your CASH is Ø.
We can say:

```
CASH = Ø
```

Now suppose that you add 5 pence to what's inside your cash pocket.

We can now say:

```
CASH = CASH + 5
```

How much CASH is in your pocket now? If CASH started off by being 2, and you added 7 to it, how much CASH would there then be in the pocket?

(What could you do if you wanted to reserve a pocket for MARBLES?) Type in a new Line 2Ø:

```
2Ø CASH = Ø
```

You've now ordered the Amstrad to reserve one of its memory pockets for a NUMBER variable called CASH. Inside this variable memory pocket, it puts Ø.

How can we know what the Amstrad has in its variable memory pocket called CASH? After all, because it is a number variable, we could change it by adding, subtracting, multiplying or dividing. We don't want to do the hard work of remembering what's inside the variable pocket, especially if what's inside it keeps changing. We would prefer the Amstrad to do the donkey work and tell us what we want.

It's easy. All we have to do is to command the Amstrad to PRINT what CASH is on the screen.

So let's type at Line 23:

```
23 PRINT CASH
```

LIST then RUN the program. It won't surprise you to see Ø on the screen. After all, that's what CASH starts off as.

Suppose your friend came along just now and saw the Ø, and asked you what it meant. You could say, "That's CASH". But why should you waste your breath when the Amstrad can give a message for you?

So type in Line 22:

```
22 PRINT "CASH = "
```

LIST the program. Remember that if the screen gets messy you can always clean it up with CLS.

See the difference between Line 22 and Line 23? The " marks in Line 22 tell the Amstrad that it is dealing with characters not numbers.

Notice that you've got Lines 22 and 23. Suppose you wanted to type a line in between these two lines. Of course, you have to give this new line a number. You can't have a line whose number is 22-and-a-bit! But the Amstrad is very friendly. Just type RENUM, then LIST your program again.

Magic! The program lines are renumbered and go up in tens again!

So here's the very same program, but with new numbers going up in tens:

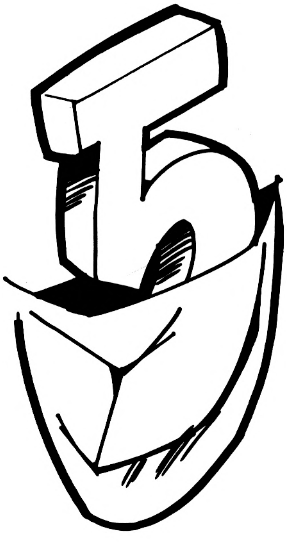
```
10 MODE 1
20 CASH = 0
30 PRINT "CASH = "
40 PRINT CASH
50 END
```

Now RUN the program. You can see how the Amstrad prints a message on the screen.

Now, say we want to increase the contents of the CASH memory pocket by 5.

Let's type in the fresh line, Line 45:

```
45 CASH = CASH + 5
```



Let's tell the Amstrad to print a message after this. Type Line 47:

```
47 PRINT "CASH = "
```

Let's order the Amstrad to also tell us what the CASH variable has changed to after we added 5 to it. Type:

```
49 PRINT CASH
```

Type RENUM to renumber the program. Now LIST it.

```
10 MODE 1
20 CASH=0
30 PRINT "CASH = "
40 PRINT CASH
50 CASH = CASH + 5
60 PRINT "CASH = "
70 PRINT CASH
80 END
```

What does the Amstrad put into the CASH number variable pocket when it comes to Line 20?

What does the Amstrad put into the CASH number variable pocket when it comes to Line 50?

Now RUN the program. Don't forget that whenever you want to see the lines of your program after you run it, you can use LIST again.

Let's go back to your own pocket. Let's suppose that you start off, once again, with no cash in it. Now, suppose you keep adding two pence to it, three times.

CASH = Ø Nothing at the beginning

CASH = CASH+2 2 pence inside now
FIRST TIME

CASH = CASH+2 4 pence inside now
SECOND TIME

CASH = CASH+2 6 pence inside now
THIRD TIME.

Now, suppose you wanted to be able to tell yourself to add to what's in your pocket, without having to remember *how many* times you want to do this.

Suppose you took a piece of paper and wrote these instructions on it:

```
FOR K = 1 TO 3
CASH = CASH + 2
NEXT K
```

Now suppose you took a sticky label and wrote **K** on it and stuck it on another pocket. Now start reading the instructions you wrote.

The instruction **FOR K = 1 TO 3** means that, the FIRST TIME , there will be 1 in this K pocket. So you add 2 pence to the CASH pocket. When you see the instruction **NEXT K**, you know that the K pocket now has got 2 in it, for the SECOND time that you have to add 2 pence to the CASH pocket. Now you come to **NEXT K** again, and now the K pocket contains 3, for the THIRD TIME that you have to add 2 pence to the CASH pocket.

As soon as K becomes more than 3, you stop adding 2 pence to the CASH pocket. That's because the instruction **FOR K = 1 TO 3** tells you to do the action only three times.

Let's try this again, with different numbers.

Suppose you wanted to add 3 pence each time to your CASH pocket. Suppose you wanted to do this 4 times.

Here are the instructions you could write for yourself to follow:

```
CASH = Ø
FOR K = 1 TO 4
CASH = CASH + 3
NEXT K
```

How many times are you adding 3 pence to the CASH pocket? How many pence will there be in the CASH pocket after you've added 3 pence to it for 4 times?

Remember our program so far:

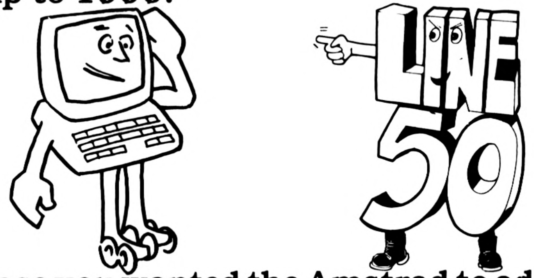
```
1Ø MODE 1
2Ø CASH = Ø
3Ø PRINT "CASH = "
4Ø PRINT CASH
5Ø CASH = CASH + 5
6Ø PRINT "CASH = "
7Ø PRINT CASH
8Ø END
```

At Line 5Ø, the Amstrad is adding 5 to its CASH number variable pocket. Let's order it to do this 2Ø times!

Type in Line 45:

```
45 FOR K = 1 TO 2Ø
```

The Amstrad will do everything between Line 5Ø and Line 7Ø as many times as Line 5Ø tells it to. Line 5Ø tells the Amstrad that K will go up to 2Ø. You can change K to go up to 1Ø or 3Ø or any number you like. Try K going up to 1ØØØ!



Suppose you wanted the Amstrad to add 7 to its CASH number variable pocket 4Ø times. What would you change Line 5Ø to? Try it now, and remember that at Line 6Ø you are ordering the Amstrad to add 7 to its CASH pocket.

Now type in line 55:

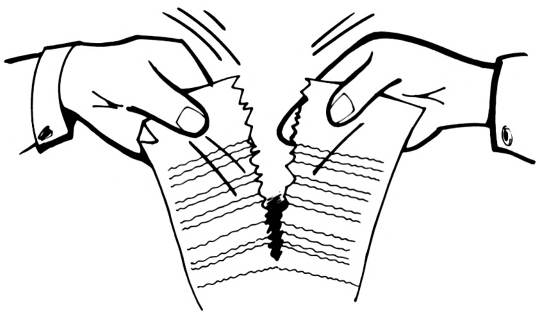
```
55 NEXT K
```

LIST your program. Now RENUM it and LIST it again. Use CLS if you want to, before doing a LIST. Now we have:

```
 1Ø MODE 1
 2Ø CASH = Ø
 3Ø PRINT "CASH = "
 4Ø PRINT CASH
 5Ø FOR K = 1 TO 2Ø
 6Ø CASH = CASH + 5
 7Ø NEXT K
 8Ø PRINT "CASH = "
 9Ø PRINT CASH
1ØØ END
```

SAVE IT!

Remember our list of things we might do at the weekend? You know, we might decide to scrap this list.



We could tear or break it up and so destroy it.

Suppose we wanted to use the list again some other weekend! We would want to save it and store it in a safe place. It would be pretty boring if we had to write the instructions in the list again and again. So we save the list.

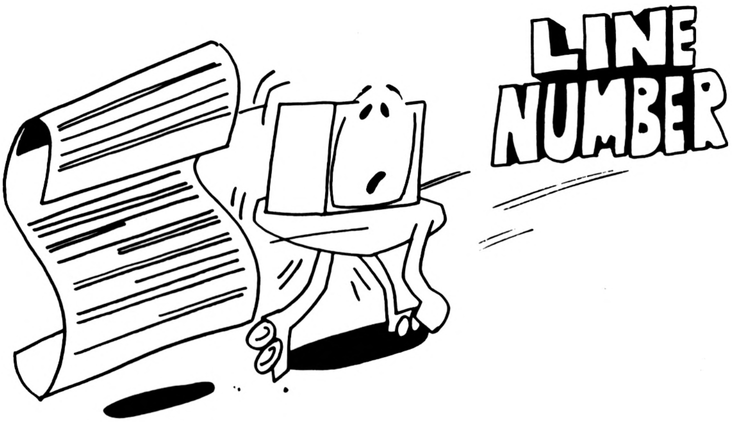
Our Money Program for the Amstrad is something we should save before we switch off the Amstrad. Remember, although we sometimes treat this pretty little machine as if it's human, that's only because we have imagination. The Amstrad is still just a machine, and as soon as we switch it off it goes to sleep and forgets everything we've taught it. So we should SAVE our Money Program before sending the little Amstrad to sleep.

But not yet! Just once more, look at the Money Program again. If it isn't already in the Amstrad, type it in. Change Line 50 to:

```
50 FOR K=1 TO 1000
```

Now LIST it.

But this time, after you type RUN, press the ESCAPE key (The red key marked ESC situated top left of keyboard) quickly twice. The Amstrad escapes from doing the Money Program instructions!



Notice that the Amstrad is really trying to look after you. It tells you at what line number the program escaped. Not bad for something that's really a collection of chips!

Before we let ourselves get carried away, let's remember that our little Amstrad can't think for itself. It tells you where it escaped because there's a program inside it that comes ready-made when you get the Amstrad. This ready-made program also does things like allowing you to save your programs on tape.

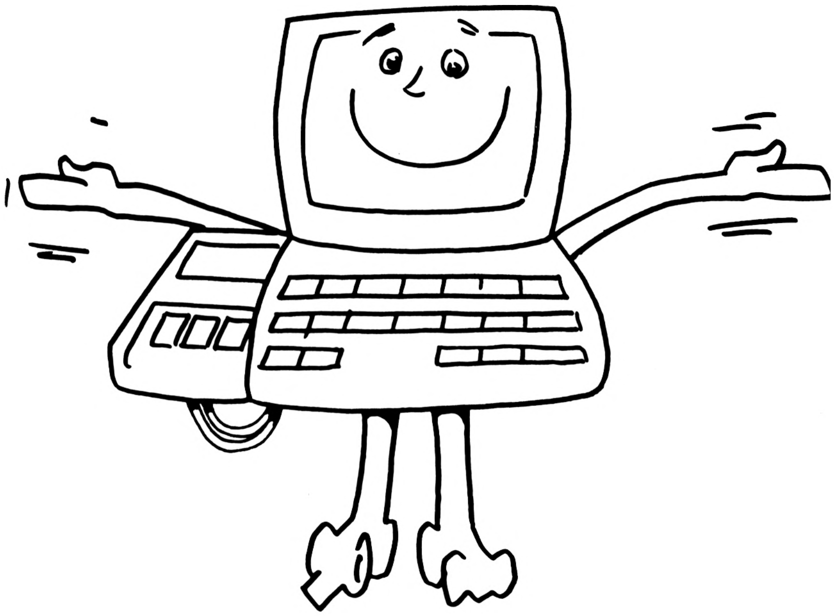
The ESCAPE key allows you to stop a program that's running. But the program itself comes to no harm. You can RUN it again or LIST it and even change it. Changing a program is called EDITING a program. We've already done some editing. ESCAPE gives you a chance to change your mind when you are in the middle of running a program.

Just a minute! Sometimes, people who write programs don't want you to be able to use the ESCAPE key in the way we just talked about. So you'll find that they have stopped the ESCAPE key from being able to do what they usually do. They have disabled these keys. Sorry to have kept you waiting to find out how to SAVE your MONEY Program. Here's how:

Put a tape — one of the 15-minute ones — into the built-in recorder. Make sure that the tape is at its beginning, then wind it just a little way forward so that there really is magnetic tape to record your program and not a little bit of dead plastic!

Now type:

```
SAVE "MONEY"
```
Remember, our little Amstrad doesn't know you've finished giving it a command until you press the ENTER key. So press ENTER.



Look at the message the Amstrad gives you on the screen and do exactly what the message says. If you don't already know, ask someone to explain what buttons to press to record something on a cassette recorder. After that, press any key.

While your Amstrad is recording your MONEY program, it will show you that it is recording it. If it records it successfully, you'll see the ready sign coming onto the screen.

Sometimes, your program may not be recorded successfully. When that happens, ask someone older for advice.

Let's look at the instruction SAVE"MONEY" again.

MONEY is the name we gave to our program. But we could give it any name we like, so long as the name isn't more than 16 characters (including spaces) long. We could have typed SAVE "SOULS" instead. But it's better to give programs names that remind us what they are about.

Now that you've learnt how to save a program, you will want to know how you can bring it back into the Amstrad from where it is on the tape. Of course, you should have an idea as to where on the tape you saved your program. If you check the counter on the tape recorder, you can note down at what numbers your programs begin and end.

Suppose a program begins at 15Ø on the tape counter. You can wind the tape forward or backward to just before 15Ø. Then you can start loading in your program.

1. Get as close as possible to just before the beginning of your program. You may have to wind your tape to do this.

2. Type:

```
LOAD"MONEY"
```

3. Now press ENTER.

4. When you see the message

```
Press PLAY then any key
```

on the screen, press the PLAY button then any key.

You'll see

```
LOADING MONEY BLOCK 1
```

appear on the screen. Then when you see the ready sign, you know the MONEY program is safely back inside the memory of the Amstrad.

If you wanted to, you could now LIST the program and change it around — in other words, EDIT it. Instead, you could order the Amstrad to obey the instructions of the program, by typing RUN.

Suppose you didn't want to go to all the trouble of first loading in the program and then running it. Suppose you just wanted to bring it in so that the Amstrad obeyed the program instuctions at once.

What you can do is, do action 1 above, but then type in

```
RUN "MONEY"
```

instead of

```
LOAD "MONEY"
```

when you come to action 2 above. Then do action 3 above. Try it and see.

At last. We've saved the MONEY program. We can load it in later any time we like. Let's have a bit of a break.

SHOWING OFF

I hope you saved your Money Program on tape. If you did, load it in and LIST it.

Anyway, here it is again:

```
 10 MODE 1
 20 CASH = 0
 30 PRINT "CASH = "
 40 PRINT CASH
 50 FOR K= 1 TO 20
 60 CASH = CASH + 5
 70 NEXT K
 80 PRINT "CASH = "
 90 PRINT CASH
100 END
```

RUN the program.

Let's look carefully at what the Amstrad displays, or shows, on the screen. Notice that the message CASH appears on one line of the screen, and the number on the next line below.

Maybe we prefer to have the number on the *same* line of the screen as CASH. How can we do this?

Of course, we'll have to change or EDIT the program, because we now want it to do something different.

Press the SHIFT key and hold down.

Use the up-arrow key to move the EDITING CURSOR to just on the 3 in Line 3Ø.

Press the COPY key gently and see how Line 3Ø is being copied freshly.

When the editing cursor gets just past the end of Line 3Ø, press the SPACE BAR (the long bar at the bottom of the keyboard) once. This gives a little space to separate one thing from another.

Now, what we want to do is have the Amstrad print what's inside the CASH pocket, just after the message, on the same line. So, if Line 3Ø now became

```
3Ø PRINT "CASH = " ; CASH
```

it would do the trick. Now type a ; then press the SPACE BAR to get a separating space.

Now type CASH. Press ENTER to tell the Amstrad that you've finished typing in line 3Ø.

What about Line 4Ø? We don't need it anymore, because Line 3Ø is doing its job. To get rid of it, just type 4Ø and then press ENTER. If you like, clean the screen with CLS, then LIST.

Now RUN the Money Program and notice the display.

Just for fun, LIST Line 30 and EDIT it again, but this time use a , instead of a ;

RUN this version of the program and notice what the display looks like this time. Now LIST line 30 and EDIT it back to what it was before.

Can't we do the same things to Line 80 and Line 90? Of course we can. The ; or the , tells the Amstrad to print what comes behind it, on the same line of the display. Why don't you now change Line 80 and get rid of line 90?

By the way, don't you think it's a good idea to now renumber the program? Type RENUM.

Your new Money Program should look like this:

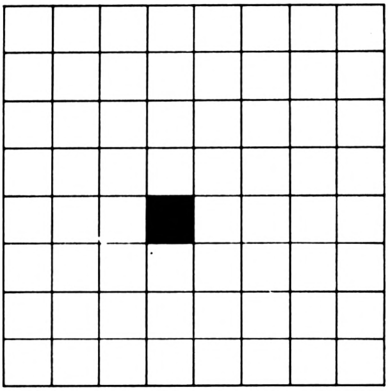
```
10 MODE 1
20 CASH = 0
30 PRINT "CASH = " ; CASH
40 FOR K = 1 TO 20
50 CASH = CASH + 5
60 NEXT K
70 PRINT "CASH = " ; CASH
80 END
```

LOCATE DANCING

I wonder if you've seen any old film musicals where someone danced across a floor that was divided up into squares just like a chess-board is.

Imagine that the screen is like that chess-board dance floor.

In MODE 1 the screen "dance floor" is divided up into 4Ø squares when you look from left to right. When you look from top to bottom, the "dance floor" is divided into 25 squares. If you want to, you can divide a big piece of paper into squares. If you do, make it 4Ø squares broad and 25 squares deep. This may help you to imagine what I'm saying much better. Each square can fit a single thing into it. This can be a single character like A or M, or a single number like 5 or 9.



Each square is itself divided up into 8 PIXELS across and 8 PIXELS down.

Remember that we said that PIXELS are little blocks. Remember that we also said that PIXELS can be fat or thin, depending on what MODE the Amstrad is in.

The fatter the PIXELS are, the fatter the squares of the screen "dance floor" are. Just like fewer fat men could fit into the telephone box, this means that the "dance floor" will have fewer squares in it in MODE 1 than it will have in MODE 2.

In MODE 1, there are 4Ø squares across the screen. In MODE 2, there are 8Ø squares across the screen. The PIXELS of MODE 1 are four times fatter than the PIXELS of MODE 2.

Let's go back to the Money Program and imagine that we want to move the word CASH over the squares of the screen.

The instruction to move things over the screen is LOCATE.

Suppose we want to move CASH five squares across the screen. LOCATE 5,1 will do this for us.

You can imagine that there is a dancer called TEXT CURSOR who is carrying CASH across the screen dance floor. The number 5 and 1 tells TEXT CURSOR to move CASH by five squares across on the first line of print.

EDIT Line 3Ø of the Money Program:

```
3Ø LOCATE 5,1: PRINT "CASH  =  " ; CASH
```

Before we carry on, let's think about how we should EDIT Line 3Ø.

First, press the SHIFT key and hold down.

Now use the up-arrow key to get the EDIT CURSOR on the 3 in 3Ø.

Now use the COPY key to COPY Line 3Ø to just before the P of PRINT.

We now have to put something new in. So press the SPACE BAR once, then type LOCATE 5,1: Then press the SPACE BAR again.

Now use COPY to copy the rest of line 3Ø to just after the H in CASH. Now press ENTER.

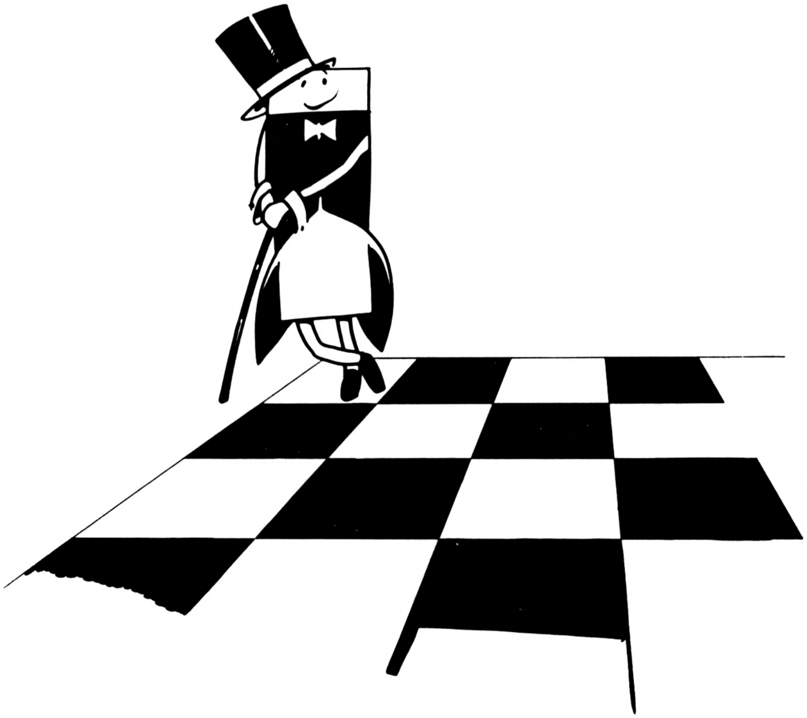
Now LIST and RUN the program.

Suppose we want TEXT CURSOR to dance across the screen, but not just five squares across but also ten squares down?

LOCATE 5,1Ø will do that for us. The first number is 5. It tells TEXT CURSOR how many squares to tab dance across the screen. The second number is 1Ø. It tells the TEXT CURSOR how many squares to dance down the screen.

Where does TEXT CURSOR begin its dance?

Look at the top left-hand corner of the



screen. Imagine TEXT CURSOR standing there, dressed in top-hat and tails and shiny shoes!

If you tell it to LOCATE 5,1 it will dance five places across the screen on the first line of print. It will then put down whatever it's "carrying".

If, instead, you tell it to LOCATE 5,1Ø it will dance 5 squares across and ten squares down the screen. It will then put down, on the screen, whatever it is "carrying".

There is one thing you have to be careful of when using LOCATE. You must enter two co-ordinates separated by a comma, otherwise the Amstrad won't understand you.

Now let's EDIT Line 3Ø to be:

```
3Ø LOCATE 5,1Ø: PRINT "CASH  =  " ; CASH
```

Now LIST and RUN the program.
Let's now EDIT Line 7Ø:

```
7Ø LOCATE 5,12: PRINT "CASH  =  " ; CASH
```

Where do you think LOCATE 5,12 makes TEXT CURSOR dance to?
LIST and RUN the program.

THE COLOURS OF THE RAINBOW

So far, we've been writing on the screen in bright yellow pen on blue paper. As the bright yellow pen is on top of the blue paper, we call the colour on top the CHARACTER COLOUR and the colour underneath the SCREEN COLOUR.

Remember, whatever is on top is called the CHARACTER. Whatever is underneath is called the SCREEN.

In MODE 1, if we want to write in red ink, we can tell the Amstrad what we want by saying PEN 3.

In MODE 1, up to 4 of the available colours can be put on to the screen at any time.

There is a choice of 27 colours. You are able to change the colour of the BORDER (area surrounding the PAPER), the PAPER (the area where the characters can appear) or the PEN (the character itself), all independently of each other. When the computer is first switched on, the BORDER and PAPER are both blue.

Now, what do we need to type to get colours for the paper [the SCREEN]? It's very easy. We use the PEN and PAPER commands.

Suppose we want to have CASH appear in RED ink (character) on YELLOW paper (screen). Look at Table 1 for the colour numbers.

Tables 1 & 2

MASTER COLOUR CHART

Ink Number	Colour/Ink	Ink Number	Colour/Ink
Ø	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	2Ø	Bright Cyan
7	Purple	21	Lime Green
8	Bright Magenta	22	Pastel Green
9	Green	23	Pastel Cyan
1Ø	Cyan	24	Bright Yellow
11	Sky Blue	25	Pastel Yellow
12	Yellow	26	Bright White
13	White		

Paper/Pen No.	Mode Ø	Mode 1	Mode 2
Ø	1	1	1
1	24	24	24
2	2Ø	2Ø	1
3	6	6	24
4	26	1	1
5	Ø	24	24
6	2	2Ø	1
7	8	6	24
8	1Ø	1	1
9	12	24	24
1Ø	14	2Ø	1
11	16	6	24
12	18	1	1
13	22	24	24
14	Flashing 1,24	2Ø	1
15	Flashing 16,11	6	24

Note that in Tables 1 & 2, when the computer is first switched on, the PAPER used is number Ø. If you look at Table 2, in the first column, you will see PEN number Ø. Now look along the same line in the Mode 1 column, you will see colour number 1. If you now refer to the master colour chart (Table 1), you will see that number 1 is equal to blue, which as already explained, is the colour of the PAPER when the computer is first switched on.

Type in a new Line, Line 15:

```
15 PEN 3
```

This gives us RED pen (CHARACTER).

Type in another new line, Line 17:

```
17 PAPER 1: BORDER 24
```

This gives us YELLOW paper (SCREEN)and BORDER.

Type RENUM.

LIST the program, then run it.

Wait a minute! It's true that we can see red letters on yellow, but the whole screen itself isn't yellow. Although what we can see is very pretty, and we should remember it, what we now have to do is to clear the screen completely to the paper (screen) colour. The command CLS does this for us.

LIST the program, and now type this:

```
35 CLS
```

Now RUN the program, and see what happens.

Suppose you change Line 3Ø to PAPER 3: BORDER 6 to give a RED paper screen. What happens if you write in red ink on red paper? Why not try it and see what happens when you RUN the program.

If you do try it, try also listing the program!

If you want to change the background colour directly, without writing it in a program line, all you have to do is type the command INK and the current screen colour number followed by the new colour number you want.

The INK command has two numbers, the first the number of the PEN or PAPER to be changed, the second is the colour that the PEN or PAPER is to be changed to.

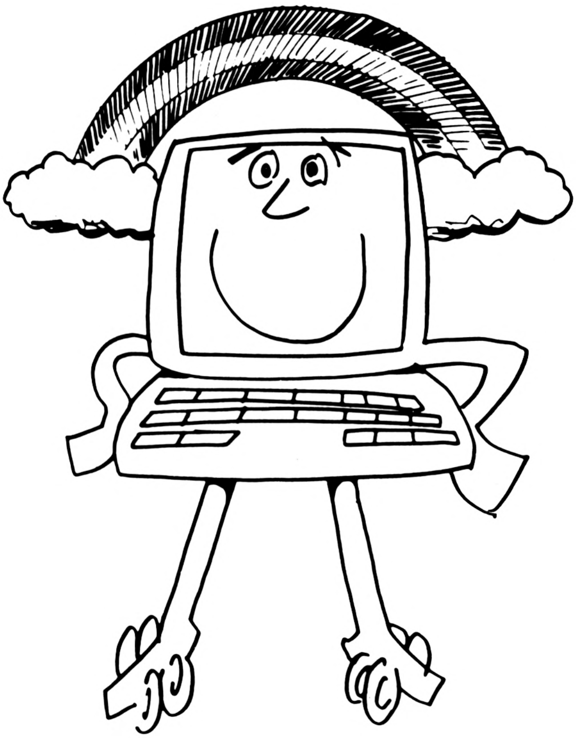
If you find that you can't see the listing because both the PEN and PAPER are the same colour, just change the PAPER colour. You can do this directly by typing PAPER and a number that's different to the PEN foreground number. Try PAPER 2. Now you can LIST the program and see it again.

But, if you want to have different PEN and SCREEN or PAPER colours while the program is RUNNING, you will have to make sure that Lines 2Ø and 3Ø give different colours.

Try different PAPER, PEN and BORDER colours.

DIFFERENT MODES

Of course, the Amstrad has other MODES.
In MODE Ø, up to 16 of the 27 available colours can be put on to the screen at any time.
In Mode 2, up to 2 of the 27 colours can be put on to the screen at any time.



SWEET SIXTEEN

MODE Ø is very interesting, because you can use up to 16 colours of the 27 colours. Remember, to change a paper colour use the INK command.
To change a PEN colour use the INK command and a BORDER colour the BORDER command.
You can also make the colour of the characters flash between one colour and another. This can be achieved by adding an extra colour number to the INK command of the PEN. eg. ink __,__,__. You can also make the colour of the PAPER behind the characters flash between one colour and another. This can be achieved by adding an extra colour number to the INK command for the PAPER. eg. ink __,__,__.

Of course, you'll have to EDIT your program if you want to try MODE Ø and new paper, pen and border colours.

If you do want to EDIT your program, first LIST it. It should be:

```
10 MODE 1
20 PEN 3
30 INK 0,24: BORDER 24
35 CLS
40 CASH = 0
50 LOCATE 5,10 : PRINT "CASH = ";CASH
60 FOR K = 1 TO 20
70 CASH = CASH + 5
```

```
80 NEXT K
90 LOCATE 5,12: :PRINT CASH = ";CASH
100 END
```

Now put a tape into the cassette player and SAVE this program under a new name.
Now change Line 10 to read:

```
10 MODE 0
```

Change Line 20 to:

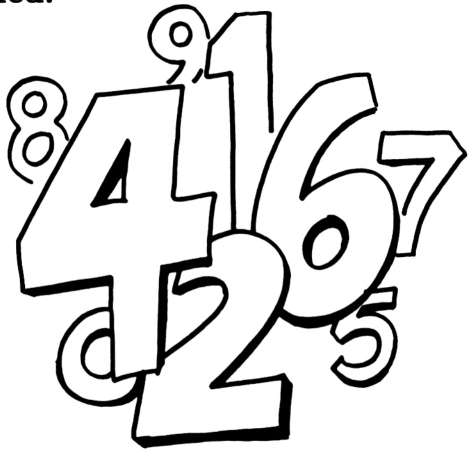
```
20 PEN 7
```

Change Line 30 to:

```
30 INK 0,10: BORDER 6
```

Now RUN the program.
You can have a lot of fun by using different MODEs, and different PAPER, PEN and BORDER colours in each MODE.

Remember to fully reset the computer's colours back to normal by pressing together the CTRL SHIFT and ESC keys when you have finished.



BACK INSIDE THE AMSTRAD'S POCKETS

If you remember, the pockets of the Amstrad's memory can have STRINGs inside them. To show that what's inside the pocket is a string, we can imagine that the sticky label on the pocket has the name of the pocket written on it, with a $ at the end of the name.

So, suppose we have a string pocket that we label A$. Suppose we want to put the string "LOCO" inside the A$ pocket.

Now, we are writing something new, so don't use the Money Program. If the Money Program is already in the Amstrad, type in NEW.

Type:

```
10 MODE 1
20 A$ = "LOCO"
```

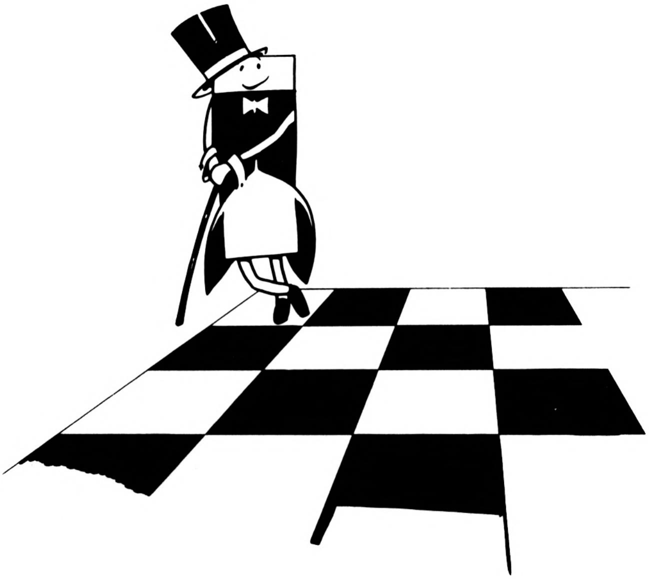
This puts the "LOCO" character string inside the variable string pocket that is labelled A$.

Type:

```
30 PRINT A$
```

RUN this little program.

Let's not forget our friend TEXT CURSOR, the LOCATE dancer.



LIST the program and change Line 3Ø to:

```
30 LOCATE 5,10: PRINT A$
```

Line 3Ø shows on the screen what the variable string pocket A$ contains. What does A$ contain?

Why not try and add Lines to change the PAPER, PEN and BORDER colours. Don't forget to RENUM after you add new lines.

Try getting red PEN (the characters) on yellow PAPER (the screen).

Did you manage? Here's what I got:

```
10 MODE 1
20 PEN 3
30 PAPER 1: BORDER 7
40 A$ = "LOCO"
50 LOCATE 5,10: PRINT A$
60 END
```

Why not type in this program and try it?

When I tried it, it looked very pretty. But it wasn't exactly what I wanted. I wanted the PAPER (or screen) to be completely yellow with a different colour border, but I forgot to clear the screen to the PAPER (or screen) colour. So I added a line after Line 3Ø:

```
35 CLS
```

Now let's add:

```
55 LOCATE 5,12:PRINT "MOTION"
```

Let's RUN the program. It looks like an advertisement for a train.

Well, so far the contents of the A$ variable pocket haven't changed. It's still got "LOCO" in it. But try this:

```
57 A$ = A$ + "MOTION"
58 LOCATE 5,14: PRINT A$
```

Quick, before we forget, let's RENUM this program. Now let's LIST it and then RUN it.

What does A$ contain now?

SAVE the program in the usual way. Let's save it under the name "TRAIN".

What does the + sign in the line A$ = A$ + "MOTION" mean? All it means is that "LOCO" and "MOTION" are put together to make up "LOCOMOTION". Then "LOCOMOTION" is put inside the A$ string variable pocket. The A$ pocket used to have "LOCO" in it. But now

what's inside has changed to "LOCOMOTION".

Putting two strings together to make a new string is called CONCATENATION.

Try this by typing it in directly and not as a program line:

```
PRINT "ELECT"+"ION"
```

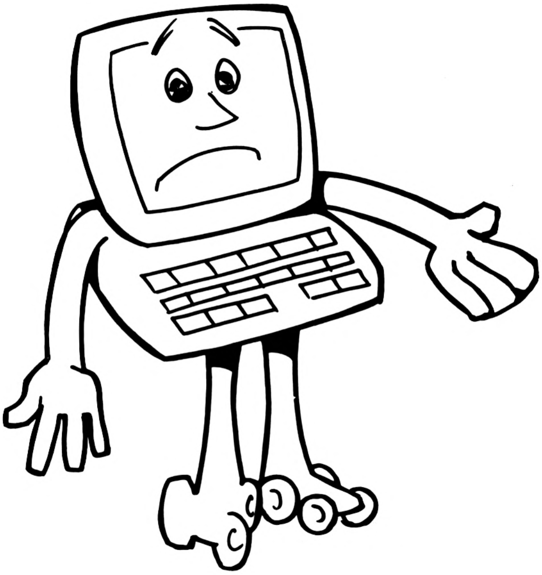
Now type NEW and try this as a little program:

```
10 A$ = "ELECT"
20 B$ = "ION"
30 C$ = A$ + B$
40 LOCATE 10,12: PRINT C$
50 END
```

If you look at Lines 10, 20 and 30 you will see that this time we have three string variable pockets. These pockets are called A$, B$ and C$.
Line 10 puts "ELECT" into pocket A$.
Line 20 puts "ION" into pocket B$.
Line 30 looks and sees what's inside both the pockets A$ and B$. It copies "ELECT" and "ION" and puts them together to make "ELECTION". It puts "ELECTION" into pocket C$.

Then, at last, Line 40 shows us, on the screen, what pocket C$ has got inside it.

IF YOU DON'T TELL IT WHAT YOU MEAN, THE AMSTRAD WON'T KNOW WHAT YOU MEAN

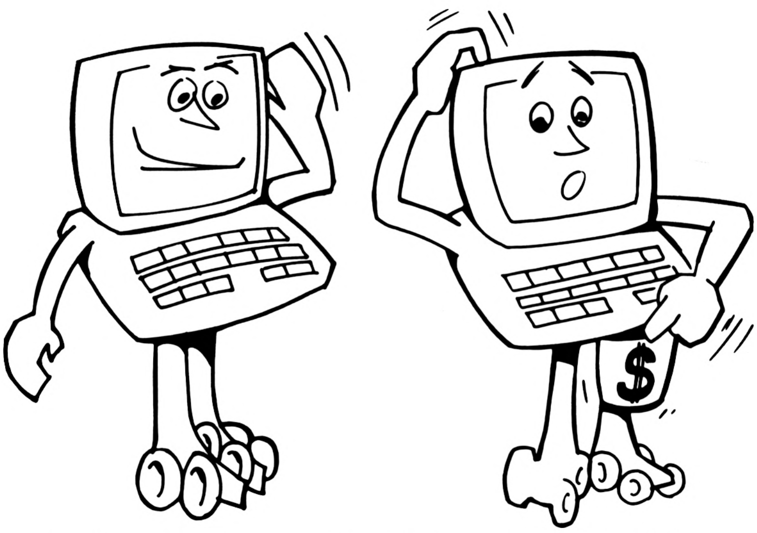


Just to see what happens, first type NEW and then type directly:

```
PRINT CASH$
```

The Amstrad doesn't print anything when you press ENTER because you haven't first told it to create the string variable pocket CASH$. You have to first type in something like Line 1Ø in the program above. So you would be alright if you typed, for example:

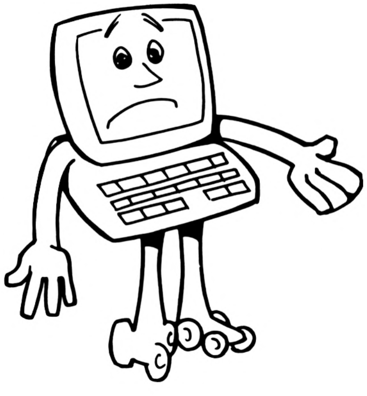
```
CASH$ = "CASH"
PRINT CASH$
```



Again, just to see what happens, type this in:

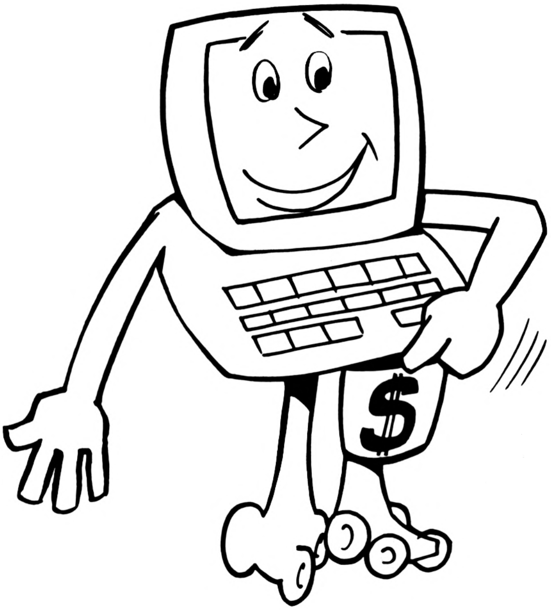
```
PRINT CASH
```

The Amstrad doesn't understand you! It treats CASH as a number and prints a Ø.
Remember, to be a character string, CASH must have " marks around it. "CASH" is a character string.



To be a string variable pocket that can contain a string, CASH must have the $ sign at the end of it. CASH$ is a string variable.
So if you only type PRINT CASH the Amstrad thinks it should make for a NUMBER variable pocket called CASH. So the Amstrad prints Ø. But if we first tell the Amstrad that there is a CASH number variable pocket, then it will be quite happy. So type:

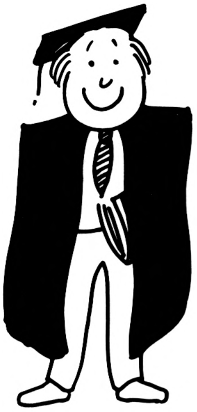
```
CASH = 3Ø
PRINT CASH
```

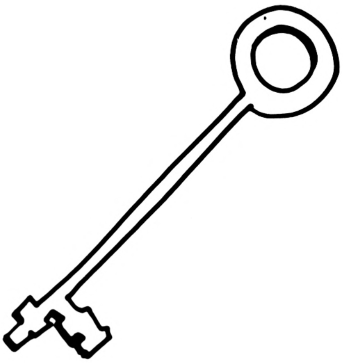


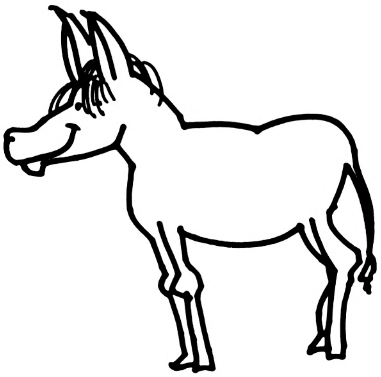
What do you see on the screen?
By the way, the messages the Amstrad gives you when it can't understand something you've typed in, are called ERROR MESSAGES.

CONCATENATING STRINGS AND ADDING NUMBERS

CONCATENATION is a very, very big word. It's a big mouthful, but its meaning is very simple. It just means putting characters side-by-side. So, if you put "DON" next to "KEY", you'll get "DONKEY".







Remember that our little Amstrad has to be told exactly what to do. To put strings side-by-side, you have to put the + sign between them. Otherwise, the Amstrad won't understand what you mean.

So, to tell the Amstrad to put "DON" and "KEY" side by side, you would have to use something like D$ = "DON" + "KEY". Then the D$ pocket would contain the character string "DONKEY" inside it.

Of course, if you wanted the Amstrad to show what it had inside the D$ pocket, you would have to tell it to PRINT D$.

Now, here's something interesting. Type in:

```
PRINT 3 + 4
```

The answer you see is 7. Here, the + sign doesn't put the 3 and the 4 side-by-side to give 34. It gives 7 instead.

Why is that? Well, because there are no " marks around the 3 and the 7, the Amstrad knows that it is looking at NUMBERS and not at strings. It ADDS the numbers 3 and 4 to give 7.

Ahah! Can you guess what would happen if you put the " marks around the 3 and the 4 ? Why not try it and see. Type:

```
PRINT "3" + "4"
```

Don't forget to press ENTER!

CUTTING THE STRING

I sincerely hope you SAVED "TRAIN" on tape earlier. If you did, you can LOAD "TRAIN" now in the usual way.

Here's TRAIN:

```
10 MODE 1
20 PEN 3
30 PAPER 1: BORDER 7
40 CLS
50 A$ = "LOCO"
60 LOCATE 5,10: PRINT A$
70 LOCATE 5,12: PRINT "MOTION"
80 A$ = A$ +"MOTION"
90 LOCATE 5,14: PRINT A$
100 END
```

Take a piece of paper and write on it: LOCOMOTION.

Now, start from the left, from the letter L. Count one-two-three-four until you come to the letter O. What have you got? That's right, LOCO.

If you cut the paper just after T, you'll get two pieces of paper. The left piece of paper will have LOCO on it.

(By the way, be careful if you use scissors. Ask someone older before you use scissors. Sharp things are dangerous.)

We can tell the Amstrad to do the same thing with character strings. We can tell it to cut up a string in the way we cut-up

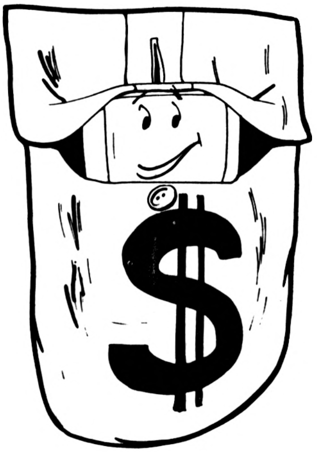
LOCOMOTION from the left.

The special word that does this is LEFT$(.

You can pretend that the (sign is like a pair of scissors that's ready to cut a string.

Add these lines to the program TRAIN:

```
95 L$ = LEFT$(A$,4)
96 LOCATE 5,16: PRINT L$
```



Let's look at line 95. The Amstrad looks inside the A$ pocket and sees
"LOCOMOTION"
inside it. Then it counts 4 starting from the leftmost character, L. It gets LOCO after counting 4 characters forwards, starting from L. It then puts the string "LOCO" into the string variable pocket L$.

Then line 96 shows us, on the screen, what's inside L$.

If there's a LEFT$(, shouldn't there be a RIGHT$(? Of course there is. What do you think RIGHT$(does?

Just for fun, try counting from the right with the word DONKEY.

Write DONKEY on a piece of paper. Start from the right, from the letter Y. Go backwards, counting one-two-three. You come to the letter K. What have you got? That's right, KEY.

If you cut the paper just before K, you'll get two pieces. The right piece will have KEY on it.

Let's add Lines 97 and 98 to TRAIN:

```
97 R$ = RIGHT$(A$,6)
98 LOCATE 5,2Ø: PRINT R$
```

Let's look at line 97. The Amstrad looks inside the A$ pocket and sees the character string "LOCOMOTION" there. Then it counts six characters backwards, starting from N. It gets the string "MOTION". It then puts the string "MOTION" inside the string variable pocket R$.

Line 98 shows us, on the screen, what's inside R$.

We should now RENUM the program TRAIN. It's changed quite a lot, so why not SAVE it again? Call it TRAIN1 if you like.

Now you can reset the computer by pressing the CTRL, SHIFT and ESC keys in order—but holding each key down until the ESC key is finally pressed. The screen will clear and the original message will reappear as if you had just switched on.

WHAT MUST I DO, IF....

Remember our list of things to do at the weekend?

```
10 PLAY FOOTBALL
20 READ A BOOK
30 WATCH TELEVISION
40 HAVE DINNER
50 PLAY WITH THE AMSTRAD
60 END
```

Let's add Line 15:

```
15 HAVE A SOFT DRINK
```

Now, when you read your list of things to do, you'll have a soft drink after playing football. But, maybe sometimes you're not thirsty! Then you'll only really want a soft drink if you're thirsty.

Let's change Line 15 to:

```
15 IF I'M THIRSTY, THEN
   I'LL HAVE A SOFT DRINK
```

Before, you would always have a soft drink. Now, because you've used IF and THEN, you'll have a soft drink only if you're thirsty.

IF and THEN, you won't be surprised to learn, are words the Amstrad can understand.

RICH MAN, POOR MAN

Imagine you start off with 2 pounds in your CASH pocket. Suppose some very generous person keeps giving you 5 pounds to put inside your CASH pocket. Suppose he does this 1Ø times. As soon as your CASH pocket has more than 2Ø pounds inside it, you shout, "I'M RICH! I'M RICH!"

Let's write this out carefully:
MY CASH POCKET HAS 2 POUNDS IN IT.
Another pocket has 1Ø in it, to show how many times the generous person will give me money to add to CASH. I'll call this pocket KOUNT.
I'll keep adding 5 to CASH. I'll do this 1Ø times, because KOUNT contains 1Ø.
IF CASH becomes more than 2Ø, I'll shout, "I'M RICH! I'M RICH!"
But anyway, whether I've got more than 2Ø pounds or not, I'll tell you what's inside my CASH pocket.
Of course, I'll keep checking to see whether I've added money 1Ø times. Now let's try to type in a program the Amstrad will understand:

```
1Ø MODE 1
2Ø CASH = 2
3Ø FOR KOUNT = 1 TO 1Ø
4Ø CASH = CASH + 5
5Ø IF CASH > 2Ø THEN PRINT "I'M RICH!
   I'M RICH!"
6Ø PRINT "CASH = " ; CASH
7Ø NEXT KOUNT
8Ø END
```

Let's look at line 5Ø. Can you guess what the > sign in IF CASH > 2Ø means? That's right, the > sign simply means *more than*.
(There's a < sign as well, just next to the *more than* sign. It means *less than*.)
RUN the program.
How much money does the CASH pocket end up containing?

Well, CASH ends up with 52 in it.

What's the use of IF? It's very useful. It allows you to do something only if there's a reason. It allows you to make a decision. IF you're thirsty, then you decide to have a soft drink. You'll only follow Line 15 of your weekend list if you have the right reason to do so.

Similarly, because of the IF in Line 5Ø, you'll only shout that you're rich when you have more than 2Ø pounds in your pocket.

To see this, change Line 4Ø to:

```
4Ø CASH = CASH + 1
```

Now, the CASH pocket is being increased by 1 each time, instead of 5.

RUN the program now and see what happens.

BUT ONLY THEN....

Now suppose the generous person didn't want to be *too* generous.

Suppose he only wanted to repeat giving you 5 pounds until your CASH pocket had more than 2Ø pounds in it. Then he would stop.

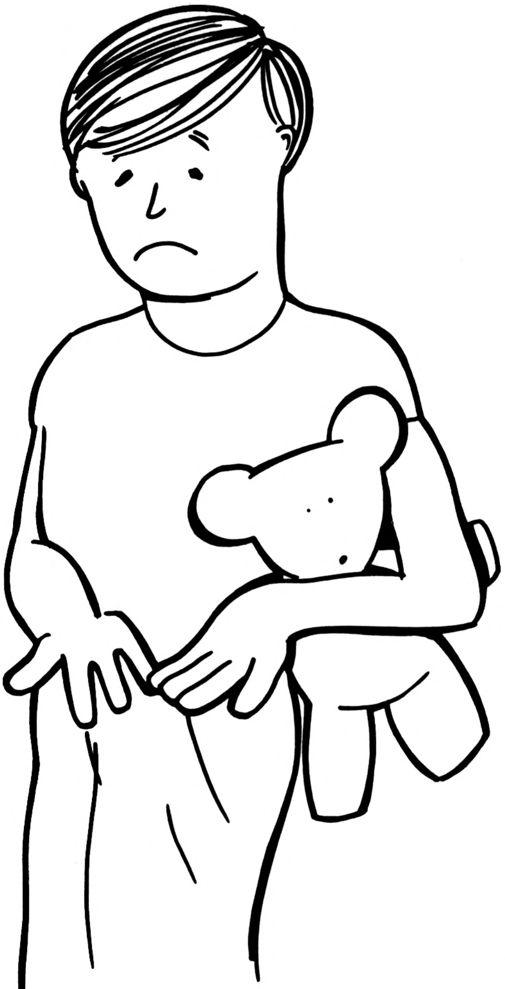
Just imagine it for yourself. You've got 2 pounds in your CASH pocket. He gives you 5 pounds. You add this to the 2 in your CASH pocket. Have you got more than 2Ø pounds now? No. So he gives you another 5 pounds. And so on. He will repeat what he's doing. But only until you have more than 2Ø pounds.

IF and THEN are two words the Amstrad understands.

Let's write a little program using IF and THEN. Remember to use NEW to show the Amstrad that you're writing a new program.

Type in the following program. You'll see big gaps in it, but don't try to copy the gaps. Just type the lines in as usual, one after the other. Of course, when you finish typing a line, you have to use ENTER to let the Amstrad know.

```
1Ø MODE Ø

2Ø PEN 4
3Ø INK Ø,14: BORDER 18
4Ø CLS
5Ø PRINT "I'M POOR!"
```



```
6Ø CASH = 2
7Ø CASH = CASH + 5
8Ø PRINT "CASH = "; CASH
9Ø IF CASH< 2Ø THEN GOTO 7Ø
```

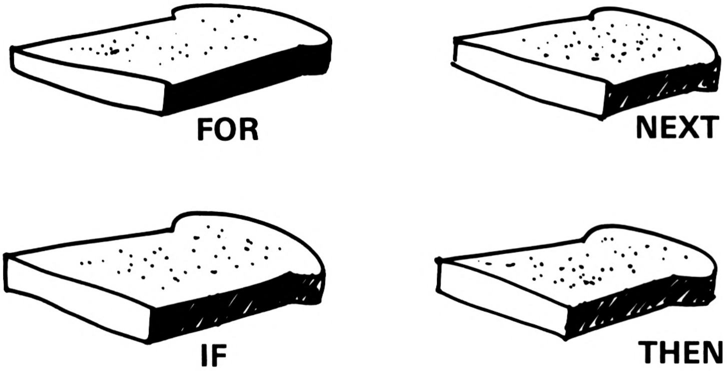
```
100 PRINT
110 PRINT
120 PRINT "YOU'RE RICH."
130 PRINT "SO NO MORE MONEY
    FOR YOU!"

140 END
```

Look at Lines 100 and 110 of the program.
These program lines just print out blank lines. The blank lines separate the "YOU'RE RICH" message from the other messages on the screen. RUN the program and see. Then, if you like, leave out Lines 110 and 120 and see what happens when you now RUN the program.

Remember how to get rid of a complete line? Just type its number again and then press ENTER.

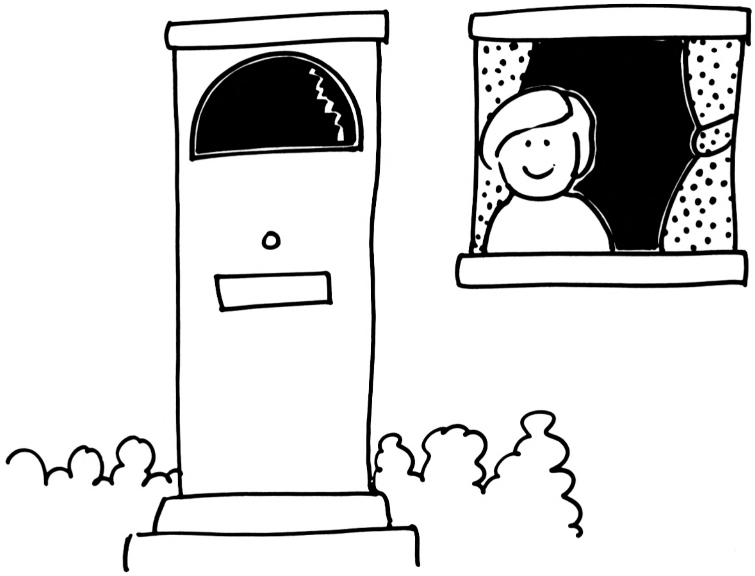
By the way, I'm sure you've noticed that:
FOR always goes together with NEXT.
IF always goes together with THEN.
They go together like the two slices of a sandwich! And, just as a sandwich usually has something in between its two slices, they usually have something between them!

If you are now going straight onto the next chapter don't forget to re-set the computer in the way already shown.



When you typed the last program you might have noticed the funny word GOTO. Your Amstrad usually does things in the order of its line numbers, but sometimes you might want it to jump around in a different order. This word does that and says "when you have done that go to somewhere else." So, GOTO 7Ø in line 9Ø means if you have less than 2Ø pounds go to line 7Ø again.

THE AMSTRAD EXPECTS....

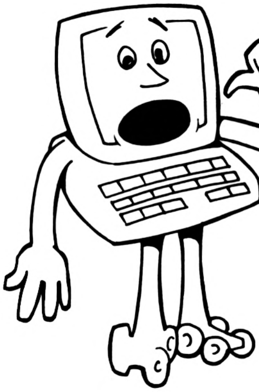


If you're expecting a letter on your birthday, maybe you'll get up early. Then maybe you'll stand by the door and wait for the postman to put a letter through the letter box.

You can also make the Amstrad wait and expect something to be put into one of its memory pockets.

What word will the Amstrad need to INPUT something into a pocket?

The word INPUT, of course.



The Amstrad can wait and expect you to give it a NUMBER to put into a number variable pocket. Or the Amstrad can wait and expect you to put a character STRING variable into a string variable pocket.

How will it know what sort of variable to input into a pocket?

Well, once again, it all has to do with whether there is a $ sign to tell the Amstrad to expect a character string variable.

For example, if you tell the Amstrad to INPUT N$, it will think that what you type in is a character string. It will put this string into the string variable pocket N$.

But if you tell the Amstrad to INPUT N, it will think that what you type in is a number. It will put this number into the number variable pocket N.

THINK OF A NUMBER!

Now type in NEW, then type this little new program:

```
10 MODE 0
20 PEN 7
30 LOCATE 3,2: PRINT "TYPE IN"
40 LOCATE 3,3: PRINT "YOUR FAVOURITE"
50 LOCATE 3,4: PRINT "NUMBER"
60 LOCATE 3,5: PRINT "FROM 1 TO 9"
70 INPUT NUM
80 LOCATE 1,7: PRINT "YOUR FAVOURITE
   IS "
90 LOCATE 8,10: PRINT  NUM
100 NUM = NUM * 9
110 ANSWER = NUM * 12345679
120 LOCATE 3,12: PRINT "HERE'S A
    LOT OF"
130 PEN 3
140 LOCATE 6,14: PRINT ANSWER/10
150 PEN 15
160 LOCATE 8,16: PRINT "!!!"
165 PEN 7
170 END
```

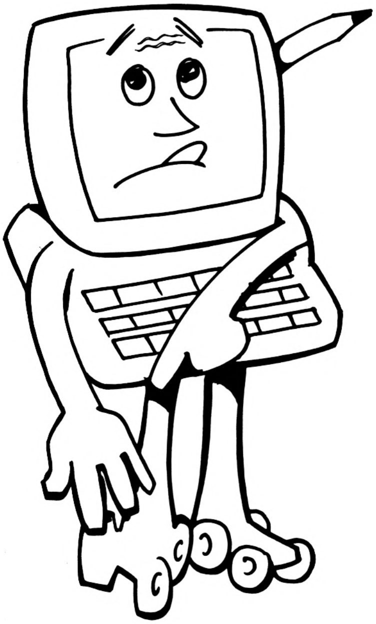
Run the program. Now LIST it.

Let's look at Line 70. If you had typed NUM = 8 at Line 70, the number 8 would already be in the NUM number variable pocket. You'd be stuck with it, and you wouldn't be able to choose a number from 1 to 9.

But, because Line 70 has the word INPUT, the Amstrad waits for you to type in a number to put into its NUM pocket.

To show you that it's expecting an input from you, the Amstrad shows a ? on the screen.

Let's look at Line 1ØØ. Here, what's inside the NUM pocket will be multiplied by 9.



Suppose you type in 4 as your favourite number when you see the ? mark. The NUM pocket will then have 4 inside it. After the Amstrad does what Line 1ØØ says, the NUM pocket will have 36 inside it.

Let's look at Line 11Ø. Here, NUM is multiplied by a very large number. We don't have to worry, though, because the Amstrad will do this big job for us. It will then put the new number into the ANSWER pocket.

By now, you should understand what the other lines of the program do. But, let's just one more time go through the program line-by-line. It's a good idea to first type in MODE 1 directly then LIST the program. Then look at the listing on the screen as you read what follows.

Line 1Ø tells the Amstrad to go into Mode Ø. Remember that we can use 16 colours in MODE Ø.

Line 2Ø gives a PEN (CHARACTER) colour number of 7.

Line 3Ø uses LOCATE to print a part of a message. LOCATE dances 3 across and 2 down the screen.

Line 4Ø uses LOCATE for another part of the message. It dances 3 across and 3 lines down.

Line 5Ø uses LOCATE for another part of the message. It dances 3 across and 4 down.

Line 6Ø uses LOCATE for the last part of the message. It dances 3 across and 5 down.

Line 7Ø shows a ? mark and makes the Amstrad wait for a number to be typed in. It puts it into the number variable NUM.

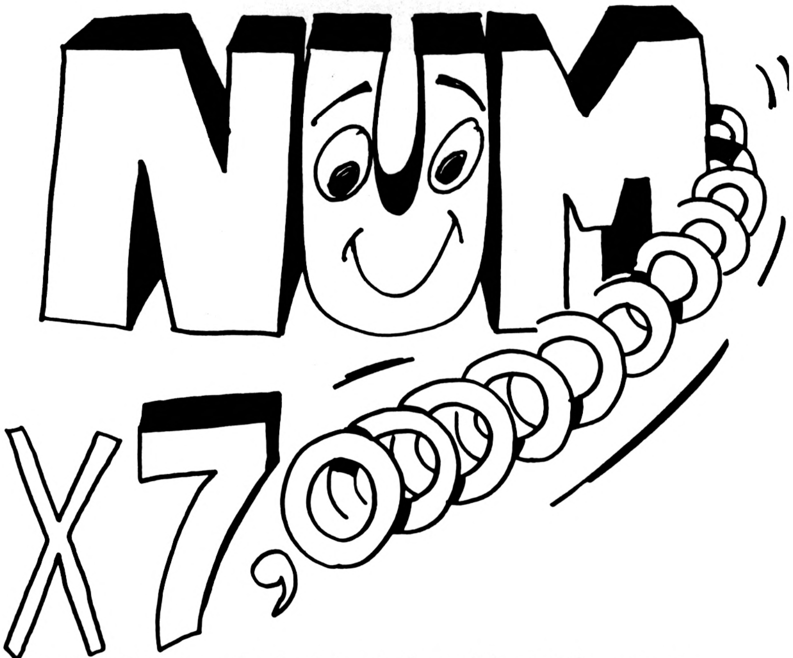
Line 8Ø uses LOCATE to show another message on the screen.

We use different numbers for LOCATE because we want to place things in different places on the screen.

Line 9Ø uses PRINT and LOCATE to show us, on a part of the screen, what NUM contains at this moment.

Line 1ØØ multiplies what NUM contains by 9. So now NUM has changed to containing 9 times what it did before.

Line 11Ø multiplies NUM by a large number.



We don't have to bother about how, because the Amstrad does that for us. It puts this number into the number variable ANSWER.

Line 12Ø places another message on the screen, using LOCATE.

Line 13Ø changes the PEN (CHARACTER) colour number to 3. Whatever is next shown will be in this new ink colour.

Line 14Ø uses PRINT and LOCATE to show what's inside ANSWER, on a part of the screen.

Line 15Ø changes the PEN colour number to 15. This pen colour in Mode Ø will flash.

Line 16Ø just prints three ! signs on a part of the screen, using LOCATE.

Line 165 changes the PEN colour back to purple.

HOW REMARKABLE!

We've gone through all the lines of this program but, of course, we will get to know things better and need explanations less.

If we want to put in little explanations inside a program itself, so that the program can be understood by someone else, we can use REM. As you can guess, REM simply tells the Amstrad that what follows it on a line is a remark. A remark just tells you something about a part of a program. The Amstrad does nothing when it comes to the REM while it is RUNning a program. The REM line is only shown when you LIST the program.

For example, we could add Line 145, just before Line 15Ø:

```
145 REM FLASHING CHARACTER
    COLOUR
```

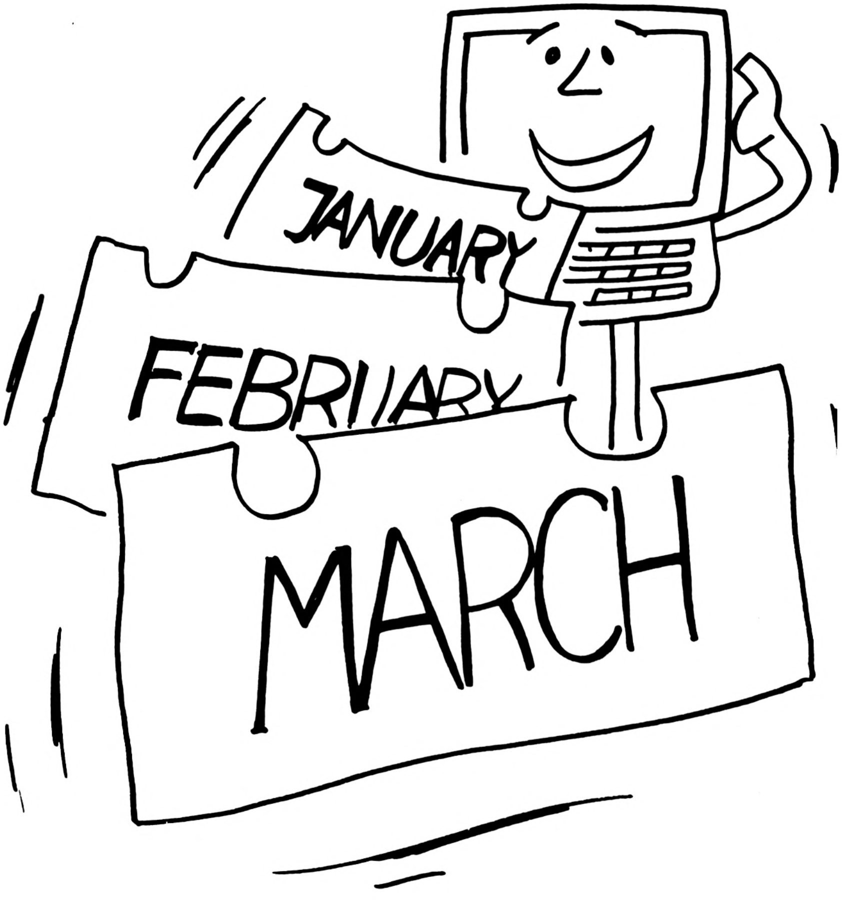
We can add other REM lines in the same way, if we want to explain some important part of a program. This is really useful in a long, long program, which could be hard to read and understand if it isn't explained a little at important points.

But, remember, too many REM lines can also make a program difficult to read. You have to use your common sense.

For example, one nice place to put a remark would be just before Line 1Ø. We could add:

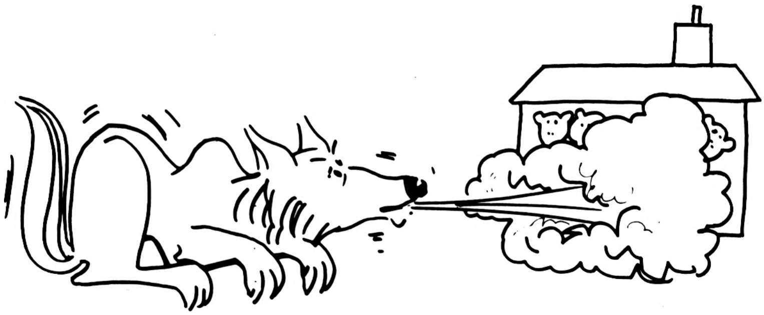
```
5 REM FAVOURITE NUMBER PROGRAM
```

Here we are just reminding ourselves what the program is about. Three months later we can look at Line 5 and easily know what the program is about.



HUFF AND PUFF

I have to admit it, it's a long time since I was at school. But I seem to remember a story about three little piggies who each built a house. I can't remember the story completely, but I'm certain one of the little pigs built his house of straw and the other one built his of bricks. And there was a big bad wolf who huffed and puffed and blew down the house of straw. But he completely ran out of huff and puff when he tried to blow down the house of bricks.



What's that got to do with programs?
A very great deal.
A house of straw is untidy. Bits of it keep falling off and you have to keep patching it. If part of a wall needs repair, you have to take out bits of good wall too, and the straw you put in is very difficult to put exactly where you want it to go.

If your house is built of bricks, then it's easy to take out a bad brick and put in a fresh one. It's easy to replace one part by another.

In the same way, you should write your programs so that they are neat, and built up of small parts. Each part should have a clear purpose. A brick has a solid, clear shape. It fits neatly with other bricks.

Is that easier to say than do? At the beginning, yes. But as you practise, it will become as easy to do as to say.

I wanted to write a game.

1. First, I wanted to PREPARE THE SCREEN.

2. Then I wanted to tell the Amstrad to create a string variable pocket and a number variable pocket.

3. I wanted to give some messages to the player, to TELL HER OR HIM ABOUT THE GAME.

4. I wanted to INPUT whether the player wanted to play the game or not. As soon as he wanted to stop, the game would END. Otherwise I would carry on with the game.

5. Then I wanted to give a problem to the player. I wanted him to guess whether a wolf was "huffing" or "puffing". This is really what the game is about.

6. While the player was trying to guess what the wolf was doing, I also wanted to check whether he had guessed right. Had he WON? If he won, I would do this:

I would tell him he had won and then ask him if he wanted to play again.

7. I wanted to tell him how many tries he had so far. I also wanted to check whether he had more than 3 guesses. If he had, then the wolf had huffed and puffed enough times to blow his house down. So he had LOST. I would give him a message and ask if he wanted to play again.

So long as the player didn't have more than 3 goes, he could keep playing until he wanted to stop.

If you think about writing any kind of program, you'll find that it uses the same sort of program bricks:

A brick to show things on the screen at

the beginning.

A brick to create certain variables.

A brick to give messages about the program.

A brick to input information for the program to look at or to use.

A brick to create something out of information the program is given. This can be a problem for the user, as in a game. But this can instead be a solution, as when the computer adds a lot of numbers together and tells you the result on the screen.

A brick to see if the program has got a result that the user wants. In this case, has the player won?

A brick to see if the program has a result the user doesn't want. In this case, has the user lost because he's used up all his tries?

A brick to keep the user informed about what's happening. In this case, how many guesses.

Each PROGRAMMING brick is called a SUBROUTINE. Notice that we haven't actually done any programmming. But we've done something more important than writing program lines. We have planned a program to be built out of SUBROUTINE bricks.

THE HUFF-PUFF GAME

We will play the game in MODE 1.

1. We'll describe or DEFINE the PREPARE SCREEN PROCEDURE brick. Let's give it a name to remember it by — let's call it REM PREPARE. So our description can be called REM PREPARE. Here it is:

```
REM PREPARE
```

The PEN colour is red.
The PAPER colour is bright cyan with a bright blue border.

```
RETURN
```

Notice the RETURN. What do you think it means? That's right, it means that we've finished the defintion of a SUBROUTINE.

2. Let's DEFINE the SUBROUTINE to create the variables we need for Huff Puff. Let's call it VARIABLES.

```
REM VARIABLES
```

Ask for the player's name. That is, INPUT the name as a character string. Put it into a a string variable pocket called NAME$.

Tell the Amstrad to create a variable number pocket called TRIES with Ø inside it. (TRIES will be increased by 1 each time the player has a go.) TRIES will tell the player how many goes he or she has had so far.

```
RETURN
```

3. DEFINE the SUBROUTINE to TELL the player about the game.

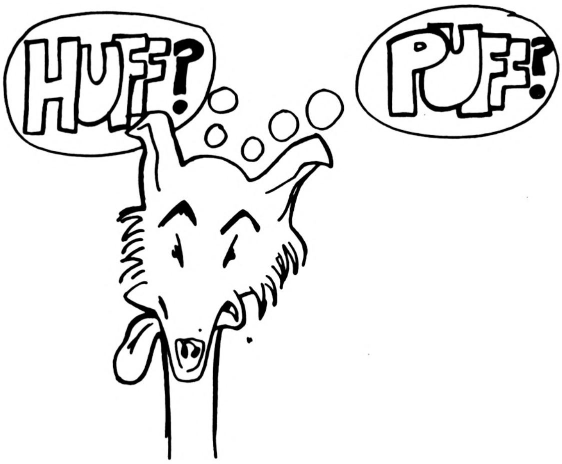
```
REM TELL

Print "This is the Huff and Puff game"
Print" Type huff if you think I'm huffing"
Print" Type puff if you think I'm puffing"
Print"then press ENTER"
Print"Yours sincerely, Wolf!"

RETURN
```

Now we have to keep repeating the problem (which is the procedure brick number 5 below) until the player types "S" for stopping the game. Number 8 below does that. We'll keep looking at A$. So long as it is not equal to "S" when the IF line is reached, the game will be repeated. We'll also tell the player to press S to stop the game if that's what the player wants. If he wants to keep playing, we'll tell the player to press P. As soon as S is pressed, we'll end the game.

4. This is the PROBLEM subroutine. Let's call it REMPROBLEM. As soon as the player tries this problem, he's having a go, so we'll increase TRIES by 1.

The problem is, is the Wolf doing a huff or a puff? How do we tell the Amstrad to choose whether the Wolf is doing a huff or a puff?

There is a very useful word called RND. If you have the numbers 1, 2, 3, and so on, written on separate pieces of paper and close your eyes and pick one, you are choosing a number at RANDOM. You can't be sure what number you will choose. RND tells the



Amstrad to close its eyes and choose a number.

If you say INT(RND(1)*12) + 1 it will choose a number from 1 to 12.

If you say INT(RND(1)*2) + 1 it will choose 1 or 2. There we have it. We will say INT(RND(1)*2) + 1 If the number that comes up is 1, then we'll put "HUFF" into a string variable pocket R$.

If the RND function chooses 2, then we'll put "PUFF" into R$.

REM PROBLEM

First, add 1 to the number variable TRIES. So, TRIES = TRIES + 1.

Make a number variable equal to INT(RND(1)*2) + 1. Call this number variable R. So R = INT(RND(1)*2) + 1.

Now if R = 1, then we say R$ = "HUFF". If R = 2 then R$ = "PUFF".

Now we'll ask the player: "Do I huff or do I puff?"

What has the player typed in? We tell the

Amstrad to expect HUFF or PUFF. We'll say something like, INPUT B$. If it is " HUFF" , we'll check R$ to see if it's equal to B$. If it is, we'll tell the player he's won, using the WON subroutine. If R$ is not equal to B$, we'll just carry on with telling the player how many goes he or she has had. We'll do the same kind of thing if the player has typed in " PUFF" . If the player types in something totally different, he just loses another go.

RETURN

5. The WON subroutine prints the player's name, says he has won, and gives a message like " I can't blow your house down. Yours disappointedly, Wolf. " Then, we ask the player if he wishes to continue, using the PLAY OR STOP subroutine.

6. We tell the player how many goes he has. If it's more than 3, we do a subroutine that we can call the HUFFPUFF subroutine. Otherwise, we repeat the problem.

7. REM HUFFPUFF can print out messages like " it's your try number four, your house is made of straw. I've huffed and puffed and blown your house down. Yours in anticipation, Wolf!" Then we ask the player if he wishes to continue playing, using the PLAY OR STOP subroutine.

8.REM PLAY OR STOP asks the player if he wishes to continue playing.

If the player hasn't said he wants to finish, or hasn't won, or hasn't used up all his goes, this means that A$ is certainly not " S" . So the game is repeated from number 4 above.

I've left it up to you to make items 5, 6, 7 and 8 more like subroutines.

PROCEED WITH YOUR PROGRAMS!

Let's make a list of the procedures, together with other bits of the Huff and Puff game:

```
 10 MODE 1
 20 GOSUB 140
 30 GOSUB 190
 40 GOSUB 240
100 GOSUB 340
110 GOSUB 550
130 GOTO 100
```

That's the whole of the Huff Puff game, written out as a list of Subroutines. Each time the Amstrad comes to a GOSUB it goes to that line and returns when it finds a RETURN Command.

So, when the Amstrad COMES TO LINE 20 and sees GOSUB 140, it looks for this:

```
140 REM PREPARE
150 PEN 3
160 PAPER 2: BORDER 2
170 CLS
180 RETURN
```

So now let's carry on with all the definitions of the procedures.

```
190 REM VARIABLES
200 PRINT "WHAT IS YOUR NAME?"
210 INPUT NAME$
220 TRIES = 0
230 RETURN
240 REM TELL
250 PRINT "THIS IS THE HUFF AND PUFF
    GAME"
```

```
260 PRINT "TYPE HUFF IF YOU THINK I'M
    HUFFING"
270 PRINT"TYPE PUFF IF YOU THINK I'M
    PUFFING"
280 PRINT "THEN PRESS ENTER"
290 PRINT "YOURS SINCERELY, WOLF"
330 RETURN
340 REM PROBLEM
350 TRIES = TRIES +1
360 R= INT(RND(1)*2)+1
370 IF R = 1 THEN R$ = "HUFF"
380 IF R = 2 THEN R$ = "PUFF"
390 PRINT "DO I HUFF?"
400 PRINT "DO I PUFF?"
410 INPUT B$
420 IF B$ = R$ THEN GOTO 460
430 RETURN
450 REM WON
460 PRINT NAME$
470 PRINT "YOU WON!"
480 PRINT "I CAN'T BLOW YOUR HOUSE
    DOWN!"
490 PRINT "YOURS IN DISAPPOINTMENT"
500 PEN 0
510 PRINT "WOLF"
515 PEN 3
520 GOTO 850
550 REM GOES
560 PRINT "YOUR NUMBER OF TRIES
    SO FAR IS"
570 PRINT TRIES
580 IF TRIES > 3 THEN GOTO 600
590 RETURN
600 REM HUFFPUFF
610 PEN 3
```

```
620 CLS
630 PRINT "YOUR TRY NUMBER FOUR"
640 PRINT "MEANS YOUR HOUSE IS OF
    STRAW"
650 PRINT "I'VE HUFFED AND I'VE
    PUFFED AND BLOWN YOUR HOUSE
    DOWN"
660 PRINT "YOURS IN ANTICIPATION"
670 PEN Ø
680 PRINT "WOLF!"
690 PEN 3
840 REM PLAY OR STOP
850 PRINT "TYPE S TO STOP"
860 PRINT "TYPE P TO PLAY"
870 INPUT A$
880 IF A$="P" THEN GOTO 1Ø
890 IF A$ = "S" THEN END
```

Type the program in. If you want to list it, it's a useful trick to type LIST 10-18Ø, press ENTER. You'll see part of the listing on the screen. To get more, type in further blocks of line numbers.

If you run the program, you'll notice that the screen doesn't look very neat. It's important that people can read what's on the screen easily. So work on this program to make it look better. Use everything you have learnt.

If you get stuck, just ask somebody who knows a bit of programming to help you. There are many things I don't understand, and I often ask other people for help. But first I try to do the things myself. So try improving this program yourself first. If you practise whenever you can, you will soon be able to write your own programs!

INDEX

Duckworth Home Computing

MY AMSTRAD CPC 464 AND ME
Jack Walker

Playing with a computer should be fun as well as instructive. The purpose of this book is to help both children and parents to understand how the Amstrad computer works and what it can do.

The book adopts a simple, friendly style with emphasis on learning as fun. A major feature is the use of short, self-contained sections which encourage children and parents to work at a steady, unhurried pace. An hour a day with book and computer should enable a child to grasp the simple but powerful ideas behind programming. This is not a text for learning BASIC but rather a way to get to know the Amstrad using simple BASIC commands. Children as young as 6 or 7 should be able to understand the text and operate the computer with or without parental help.

Jack Walker has written and edited numerous computer books. He runs his own editorial and marketing company, J.S.W. Publishing Services.

ISBN 0-7156-1964-0

9 780715 619643

Duckworth
The Old Piano Factory
43 Gloucester Crescent, London NW1

ISBN 0 7156 1964 0
IN UK ONLY £2.95 NET

MY AMSTRAD CPC464 AND ME

JACK WALKER