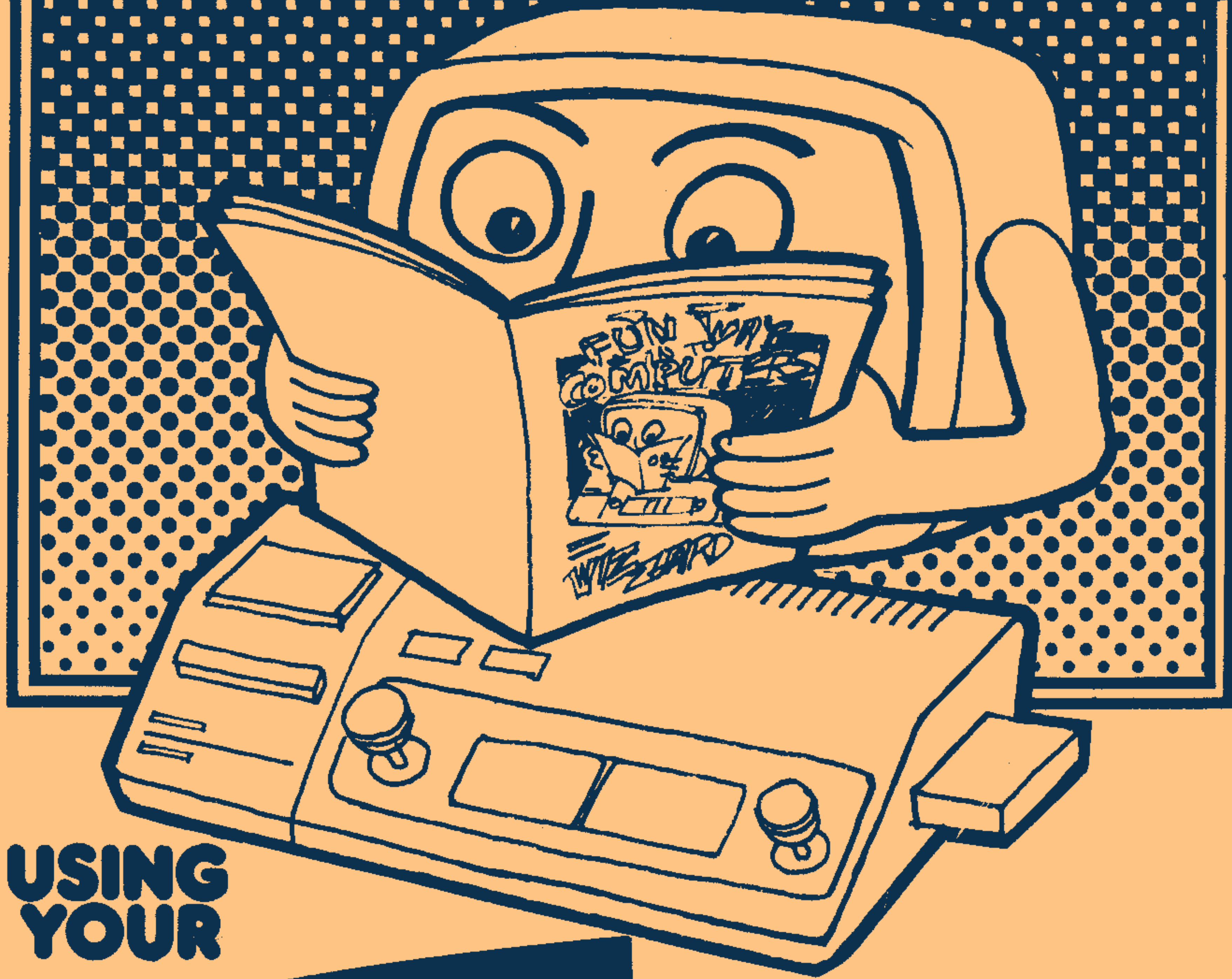


**DICK SMITH'S**

# **FUN WAY INTO COMPUTERS**



**USING  
YOUR**

# **WIZARD**

**Cat. B-6195**



## FOREWORD BY DICK SMITH

Hi! I guess if you're reading this book, it's because you've either just bought a Wizzard computer, or are thinking of buying one. So I'd like to welcome you to the fascinating world of computers.

It's hard to describe my excitement when we were presented with the chance to sell the exciting new Wizzard computer in Australia and New Zealand. Because of the Wizzard's incredibly low price, I knew that it would give many more people than ever before the opportunity to get familiar with, and confident with, computers. Particularly kids -- and I know from my own daughters that all of our kids are going to HAVE to get along with computers. Their world is going to be full of the things, so I believe it is tremendously important for us to help them become the masters rather than the slaves!

To ensure that the Wizzard really would help everyone learn about computers, though, we needed a REALLY GOOD introductory book to go with it. One that didn't assume that you already knew quite a bit about them, that didn't bamboozle you with a lot of technical jargon, that gave clear, easy to read explanations without "talking down". And above all, one that avoided the terribly boring, serious tone that always seems to creep in around computers.

In short, we needed a book that really would help people use the Wizzard to understand computers, and would make this both easy and FUN!

That then was the challenge I set Jim Rowe and Sue Robinson -- to come up with a better introduction to computers than I've ever seen before. Not an easy job, but I think they've managed to do it in this "Fun Way into Computers" book.

So read it for fun and enjoyment -- and learn quite a lot about computers at the same time!

*Dick Smith*



## ERRATUM:

Please note that initial shipments of the Y-1605 Wizzard BASIC cartridge exhibit a minor peculiarity concerning the RND function, described on page 41.

With these BASIC cartridges, the number guessing program on page 39 will not run properly when the new line 10 is typed in as shown on page 41. However normal operation of this and similar programs using the RND function can be achieved quite simply, by adding this extra line:

```
15 A = INT(A)
```

In other words, whenever you use the RND function to generate a random number, turn the number into an integer using the INT function before doing anything else.

National Library of Australia Card No.  
and ISBN 0 949772 12 7

COPYRIGHT (C) 1982, DICK SMITH ELECTRONICS

The material in this book is protected by copyright. It may not be legally reproduced, stored in a retrieval system, transmitted or copied by any means -- whether electrical, magnetic, photographic or other technology -- without specific written permission from the publisher. All such rights are reserved by Dick Smith Electronics Pty Ltd, Sydney, Australia.

**DICK SMITH'S**  
**FUN WAY**  
**INTO**  
**COMPUTERS**  
**USING YOUR WIZZARD**

Written by Jim Rowe and Sue Robinson.

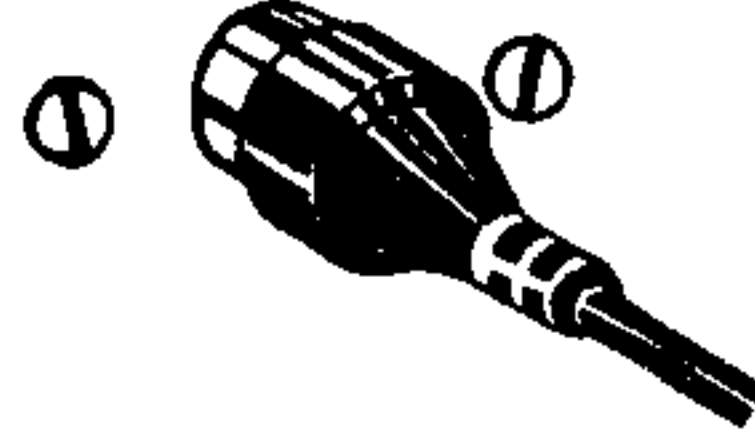
Cartoons by Alistair Barnard

PUBLISHED BY DICK SMITH ELECTRONICS PTY LTD,  
SYDNEY, AUSTRALIA

### READ THIS FIRST!

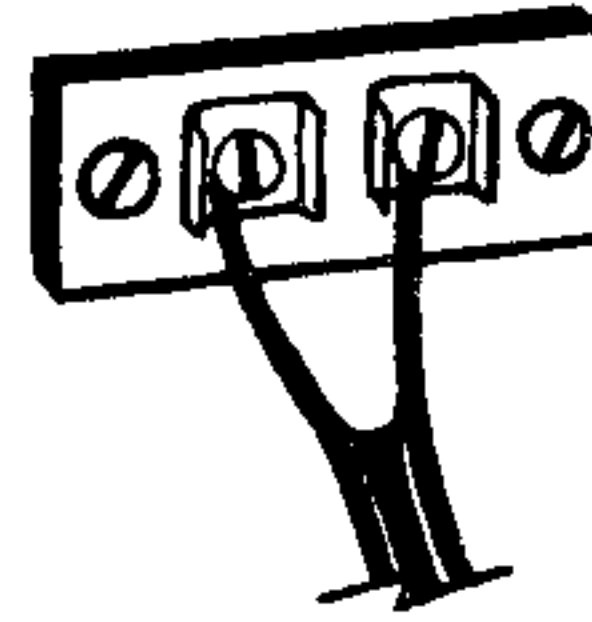
Before you unpack your Wizzard and start putting it together, take a quick look at your TV set. Particularly the aerial socket around the back.

If you are lucky,  
it will look like this:



...and you won't need to read any more of this page.

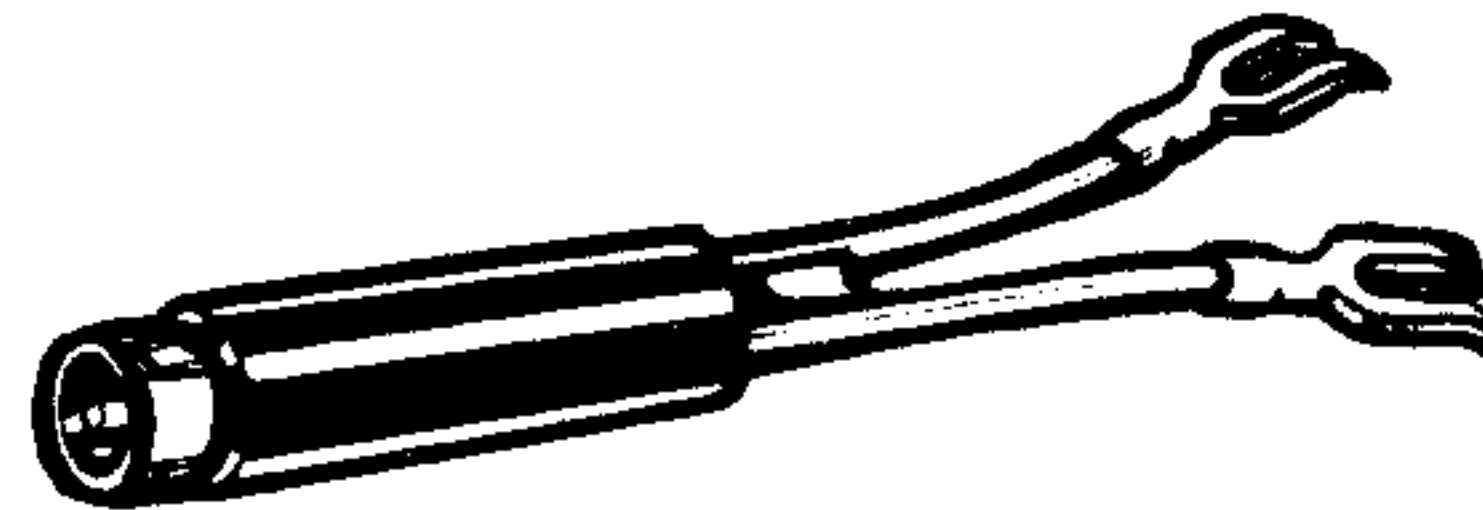
But if it doesn't, and  
looks like this instead:



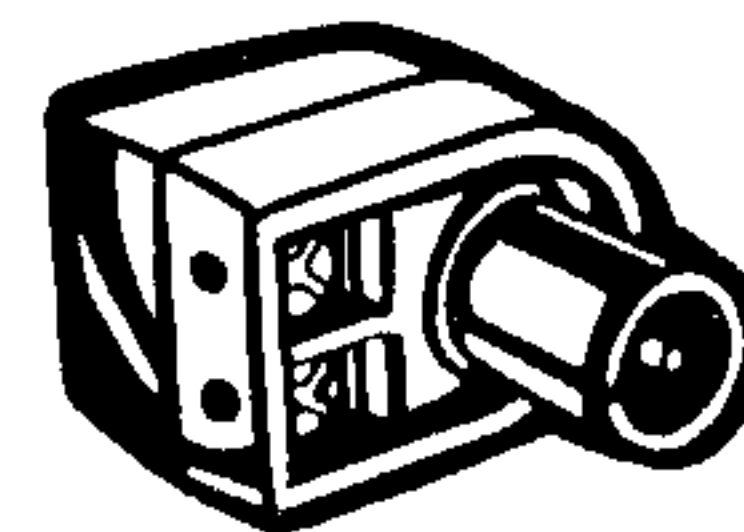
don't worry, you can still use the Wizzard, but you will need at least one extra gadget called a BALUN. In fact, you'll probably also need a second one, but slightly different.

Both types of balun are available from any of the Dick Smith stores and many of our resellers. They cost around \$2.00 each.

You will need a "75 ohm to 300 ohm"  
TV-game type balun. Our L-4454 balun  
is suitable and looks like this:



The other balun you are likely to  
need is a "300 ohm to 75 ohm" balun.  
Our L-4456 balun is one of these;  
it looks like this:



You'll find out how to use these baluns to connect the Wizzard to your TV on page 8, but we wanted to warn you about the possible need for them, to save you being disappointed later.

There's nothing worse than starting something, only to find that you can't go on because you haven't got a special part. Especially as you generally find this on a Saturday afternoon, when all the shops have shut!

## LIST OF CONTENTS

CHAPTER	PAGE
1 -- What a computer isn't .....	1
2 -- Getting your Wizzard going .....	4
3 -- A bit more about computers .....	11
4 -- Teaching your Wizzard some tricks .....	17
5 -- Becoming a programmer .....	23
6 -- More programming .....	26
7 -- More on tables .....	33
8 -- Guessing games .....	37
9 -- Making Wizzard do more of the work .....	41
10 -- Saving programs on cassette .....	45
11 -- Subroutines .....	48
12 -- Storing data in your programs .....	52
13 -- Getting Wizzard to play a tune .....	56
14 -- Which colour would you like? .....	61
15 -- Drawing on the screen (graphics) .....	65
16 -- A bit more about Wizzard's maths .....	70
17 -- Some final comments .....	74
Appendix A: What the unfamiliar words mean .....	75
Appendix B: A summary of Wizzard's BASIC .....	77
Appendix C: Wizzard's error messages .....	82
Appendix D: Wizzard's ASCII character codes .....	84
Appendix E: Answers to quick quiz questions .....	85

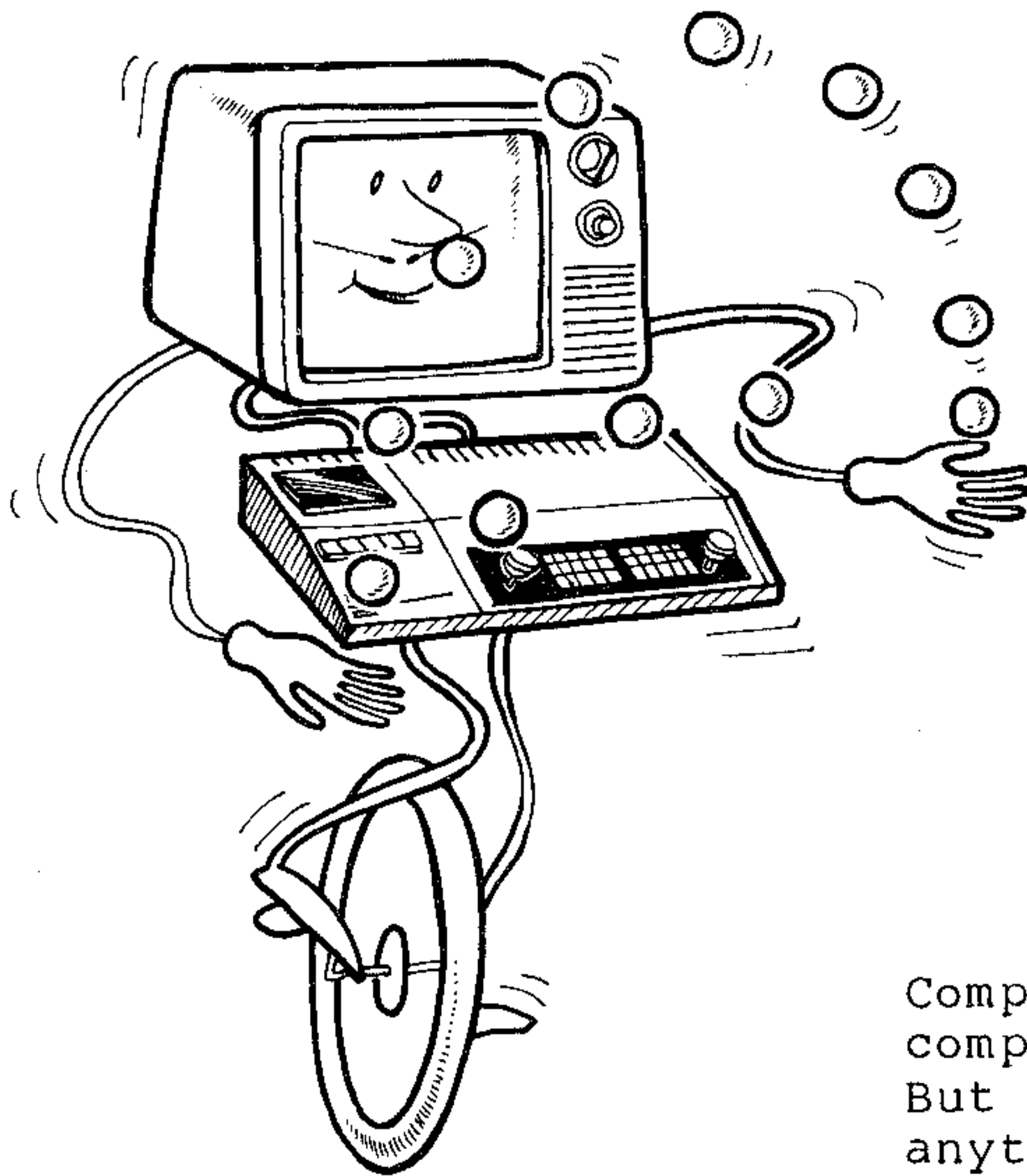
## CHAPTER 1 -- What a computer isn't

First of all, forget everything you've heard about computers being electronic brains -- they're not.

Computers can't think like you and I. The only thing they can really do is obey commands, like a trained dog (which may have a brain -- but a genius it's not).

Just as you train a dog to sit or fetch, you can train a computer to add numbers together or store information for you.

Only it can do these things FAST.



Computers do seem to be able to do complicated and impressive tricks. But remember, they can't do anything without instructions -- step by step -- from a human being.

Is a computer really just a super fast dum-dum then -- all "muscles" and no brain?

In a word -- yes!

A computer can't think for itself. It does exactly what it has been told to do.



Let's imagine that you move to a new town -- a town that you haven't visited before. And you have to work out how to get to the local shops, the school, the railway station etc.

It won't be easy. But after a lot of mistakes, you'll finally work out the shortest way of getting to each place.

Now that you have worked it all out, you can save other people the same trouble. Just give them a set of simple instructions. For example, to get to the railway station you might write the following:

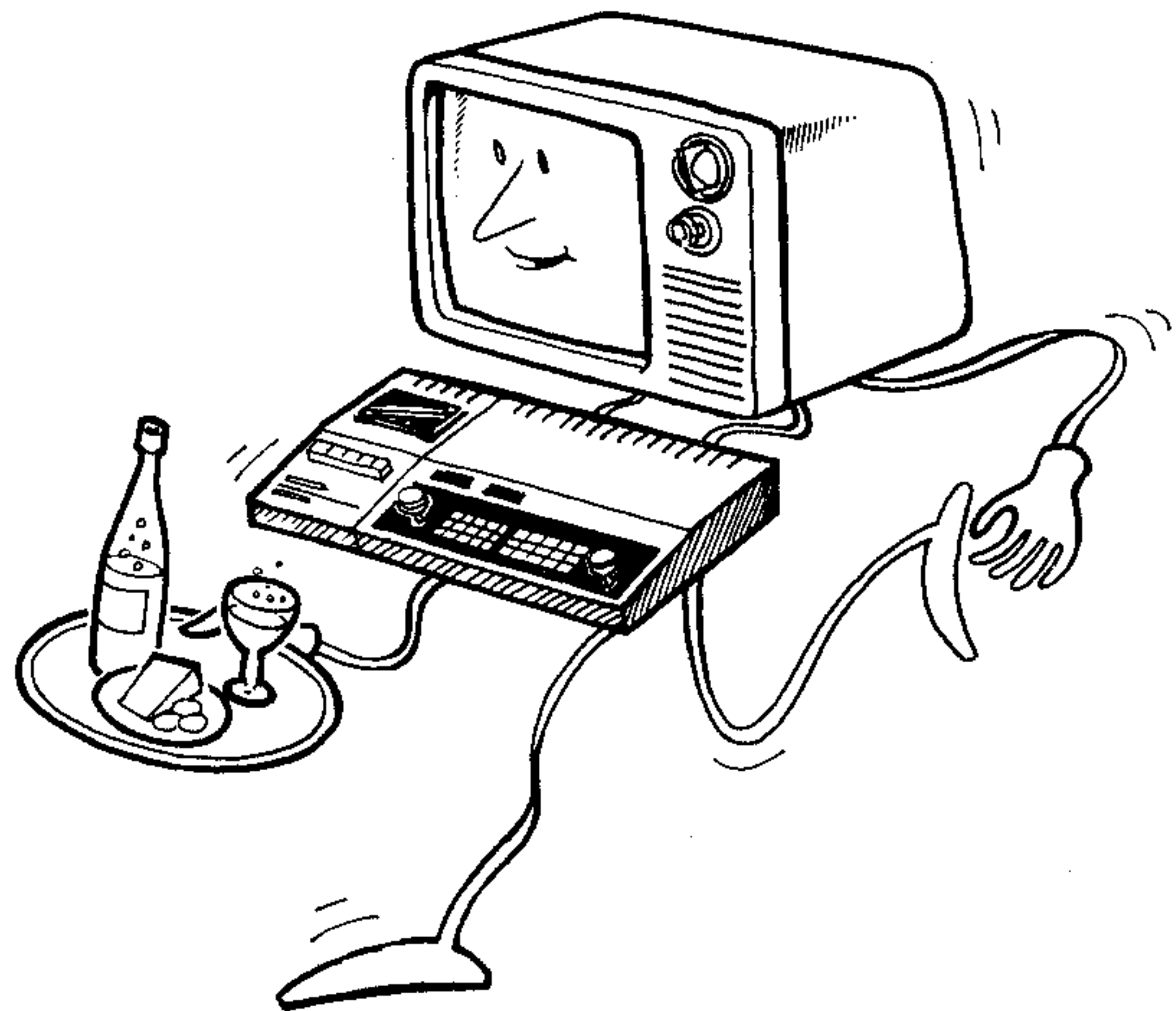
1. Turn left outside the front gate
2. Walk along the street until you reach the fire station.
3. Turn left and you'll find the station 100 metres along, on the right.

To get there, they won't really have to think, just follow your instructions.

In a similar way, when a computer does anything, no matter how complicated, it is simply following a set of step-by-step instructions. This is its "PROGRAM".

When you give the computer the step-by-step instructions it must follow to do a job, you are "PROGRAMMING" it.

So think of a computer as a sort of super fast, electronic slave. A slave which can do all sorts of useful things for you, providing you first tell it exactly how to do them.



Just before you unpack your Wizzard computer and get going with it -- try testing yourself with the simple questions on page 3.

Chapter 2 will tell you how to put the Wizzard together.

## QUICK QUIZ 1

1. What can a computer work out for itself?
2. What does a computer actually do? (Tick one)
  - a. Tell us things we don't know
  - b. Figure out the answers to our problems
  - c. Just follow our instructions, or somebody else's, and do it faster than we can
3. What do you call the set of instructions we give a computer?
4. Why did the computer cross the road?
5. What must you remember when telling a computer what to do? (Tick the item or items that are right)
  - a. Ask it nicely
  - b. Tell it in detail everything you want it to do
  - c. Don't swear at it
  - d. Get your instructions in the right order
  - e. Be sure to speak clearly to it

Now turn to page 85 to check your answers. If you have got anything wrong, go back through chapter 1 until you fully understand it. Then you will be ready to go on to chapter 2.



## CHAPTER 2 -- Getting your Wizzard going

This chapter will tell you how to get your Wizzard going, so you can try your hand at some programming.

The Wizzard is a very powerful little computer and it's ideal for learning about programming.

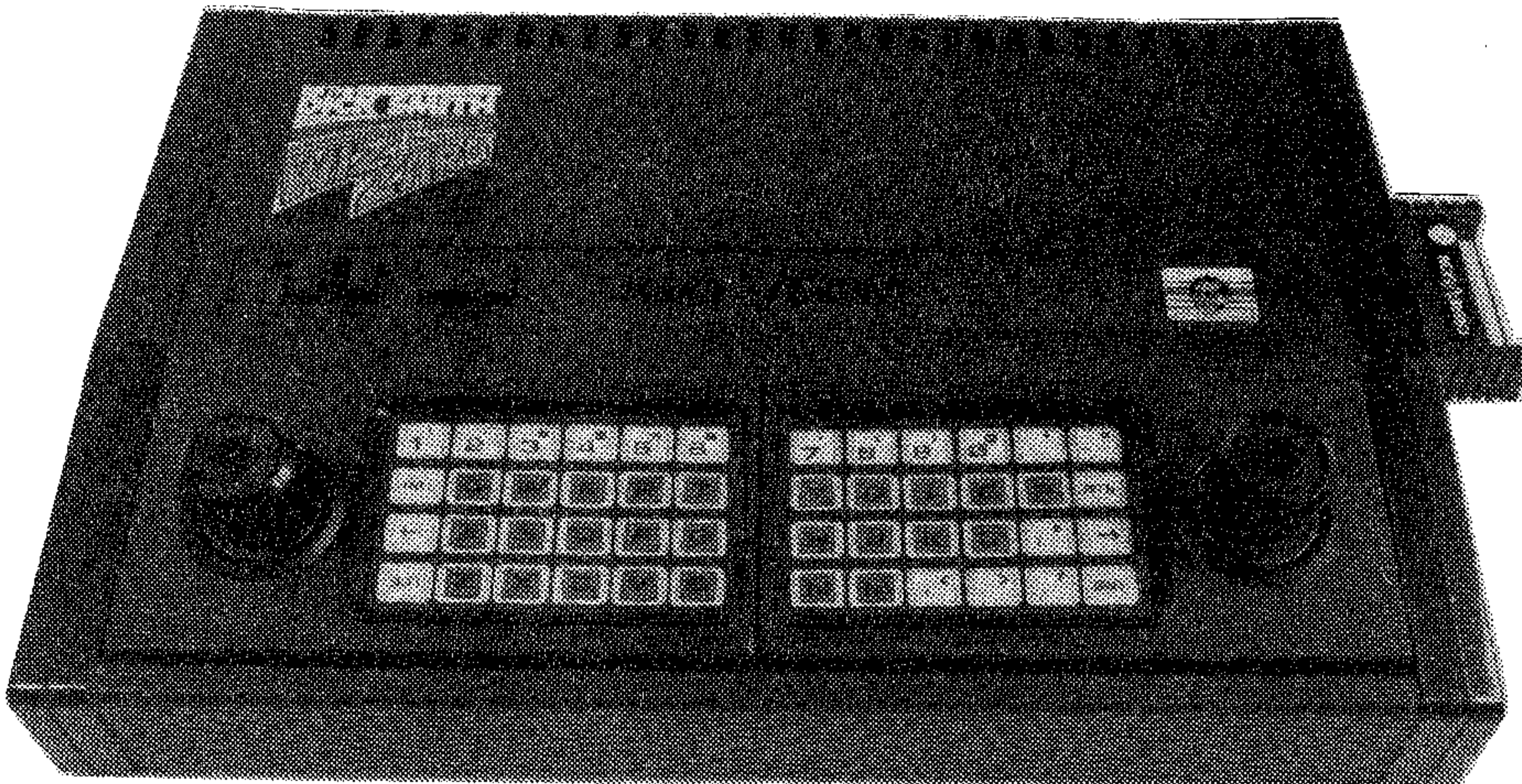
It costs much less than most computers, it's really easy to connect up, and just as easy to use.

The first thing to do is unpack your Wizzard from its box...

### What you get

Your Wizzard consists of 4 main pieces:

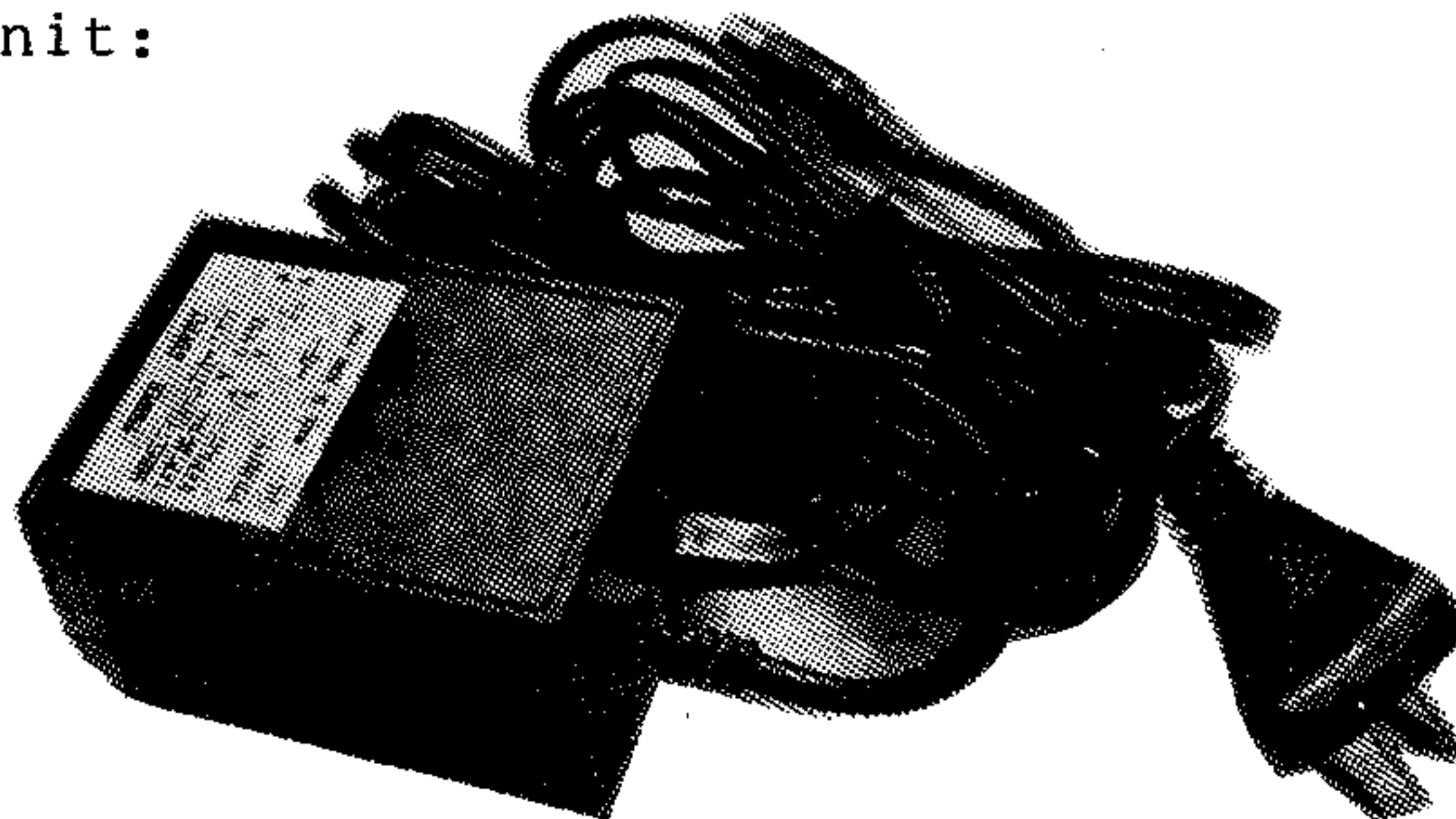
1. The Wizzard console itself. This has the two hand control units, making up the keyboard, a cable coming out of the back for connection to the aerial switch box and your TV set, and a socket on the back for the cable from the power adapter unit.



It has another socket on the right-hand side, to plug in your programming cartridge. This socket will also accept games cartridges, which have already been programmed.

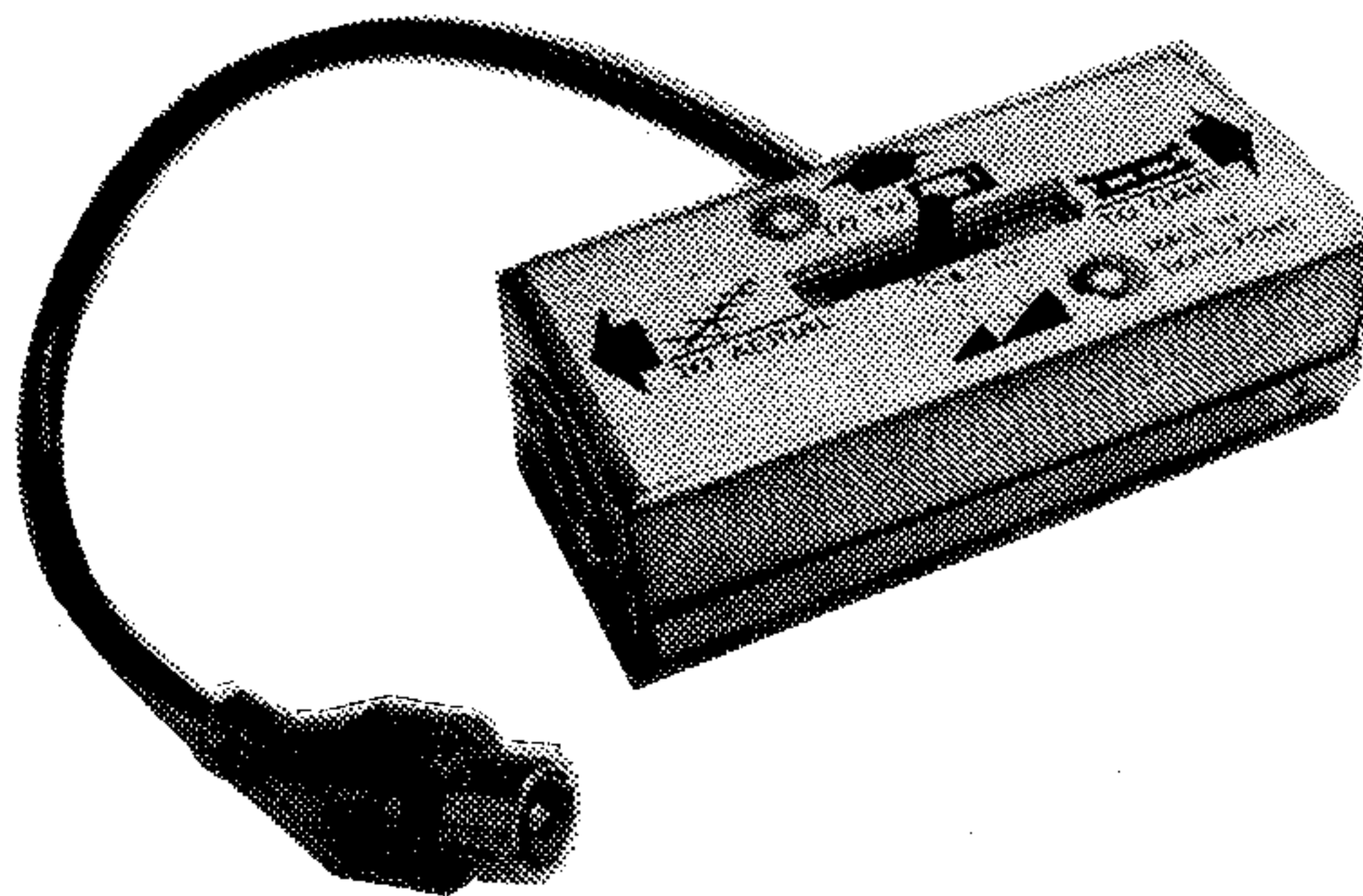
There's an on/off switch and a RESET button on top, but more about these later.

2. The power adaptor unit:



This has a cable on one end, which plugs into the power point, and another cable which plugs into the back of your Wizzard at the other.

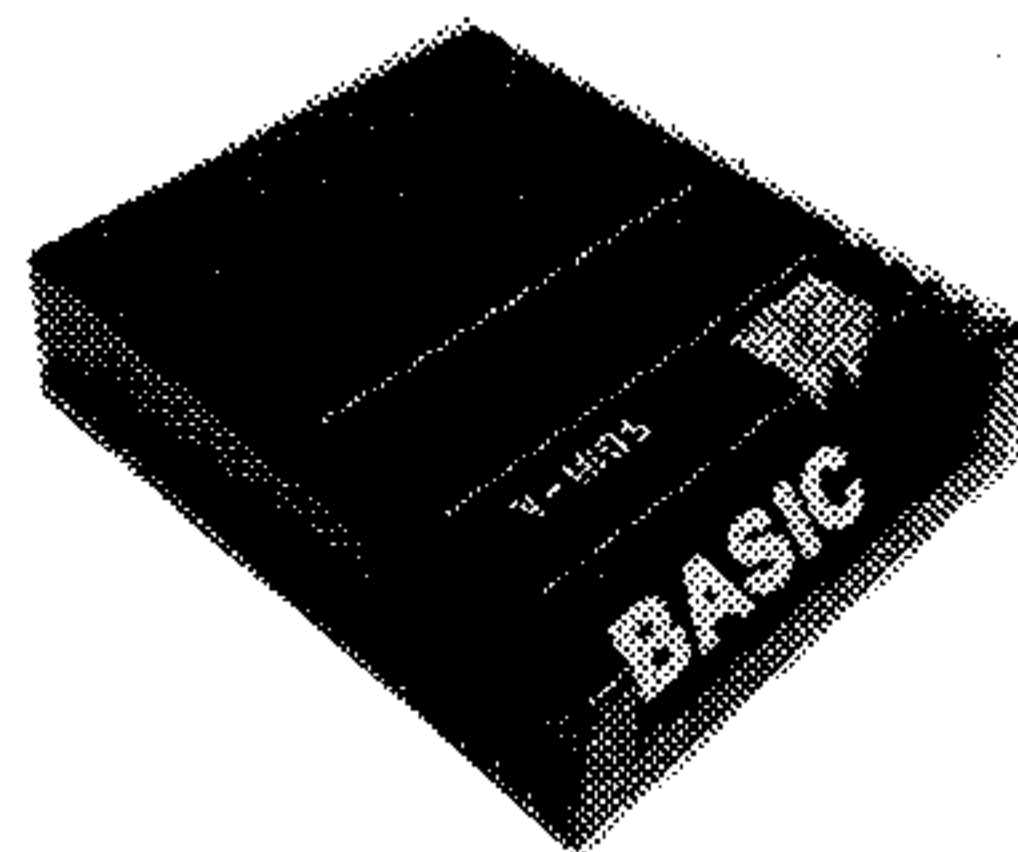
3. The aerial switch box:



This has a short cable and plug, which plugs into the aerial socket on your TV set. It also has two sockets: one to take the plug on your TV aerial cable, and the other to take the plug on the Wizzard's output cable. There is a sliding switch on the top, to let you connect the TV to either the aerial (for normal reception) or to the Wizzard, without fiddling with plugs.



#### 4. The BASIC cartridge:



This plugs into the socket on the side of your Wizzard and tells the computer how to understand your programs, which are written in simple BASIC language.

#### And for keeping records...

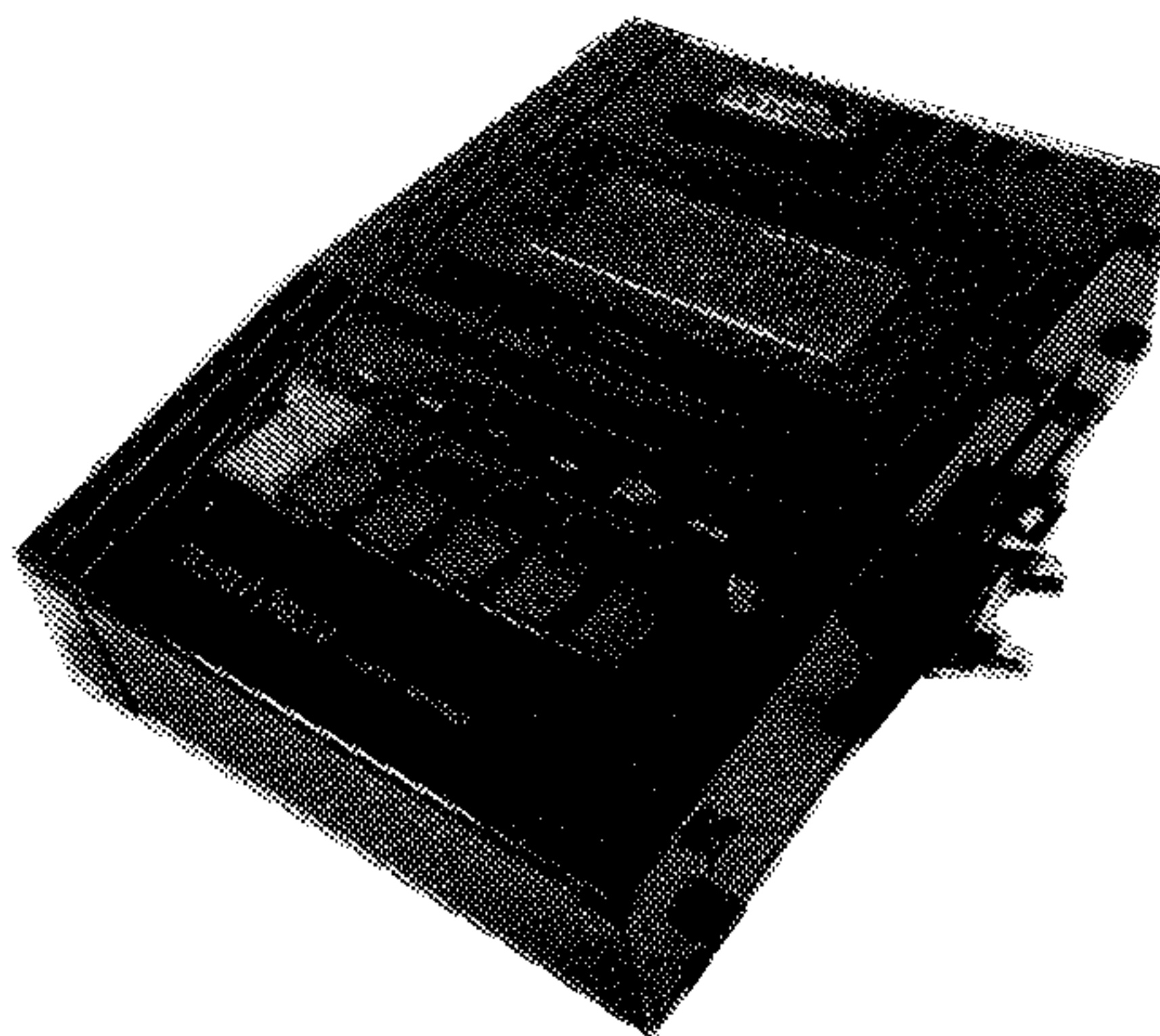
Wizzard does not have a very big memory and it forgets the instructions you have given it whenever you turn it off.

Sometimes you will want to use the same instructions again and again. And it will be a nuisance to have to keep telling them to Wizzard every time you want it to do that job.

Wizzard's Cassette attachment is an optional extra but it is well worth having, so that you can keep a record of the programs you have written for Wizzard. (We explain how you set it up and use it in chapter 10).

If you get the cassette attachment, it is a good idea to also get some Dick Smith C-10 cassette tapes to go with it.

These look like ordinary cassette tapes, but they are specially designed for recording your lists of instructions (PROGRAMS) in the same way as an ordinary cassette records music.



## Putting it all together

First of all, place your Wizzard in a convenient place, somewhere in front of your TV set. It should be within a couple of metres of a power point.

Make sure it's also where you can sit down in front of it comfortably, for hours on end. Computers are like good books -- once you start, you don't want to stop!

In this book,  
this sign means  
"do this yourself"



Now take the power adaptor unit and plug the cable with the small round 5-pin plug into the socket on the back of Wizzard.

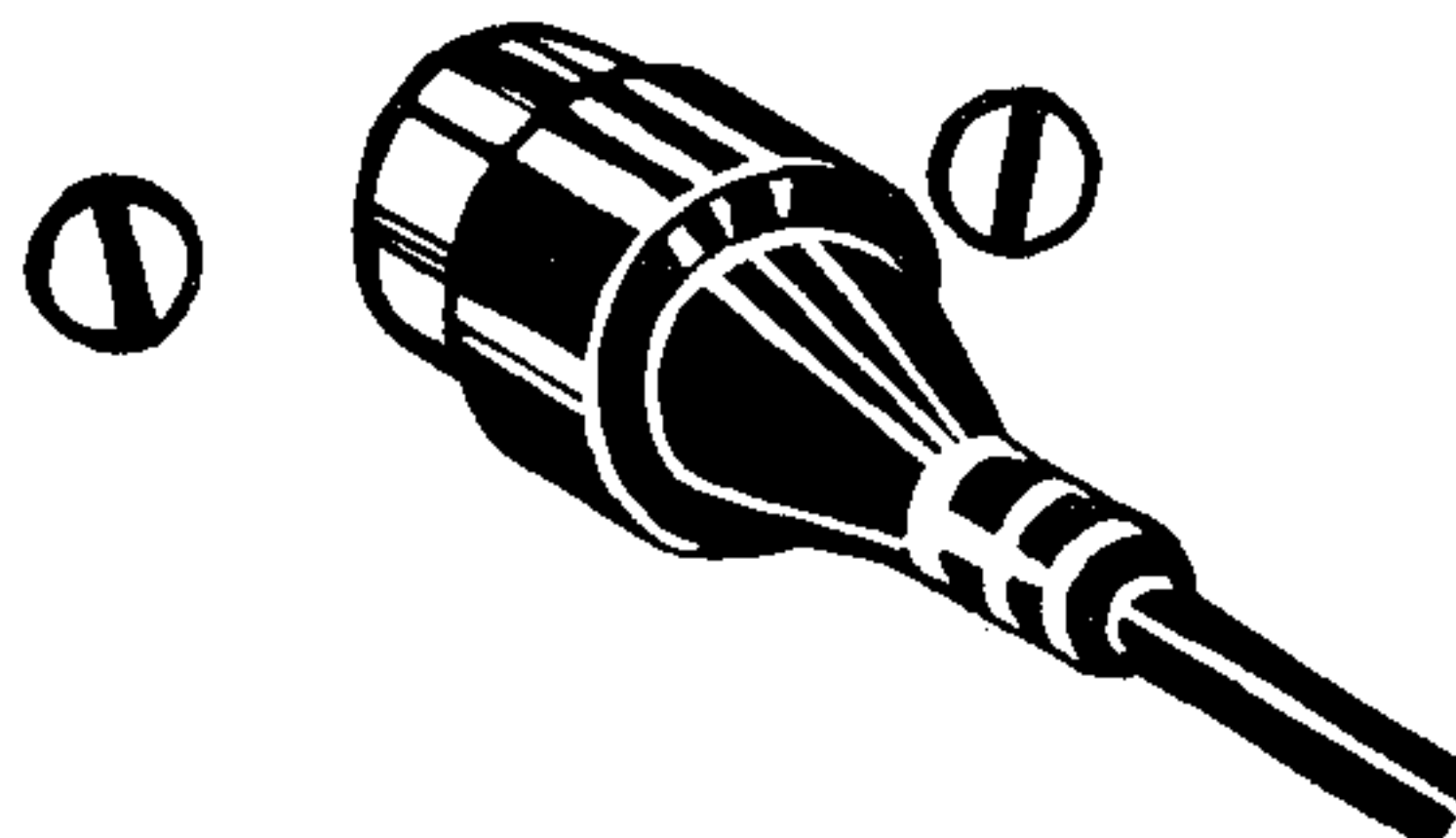
Be careful, don't force it. It will only plug in properly with the pins towards the bottom, and when you have them right it slips in easily.

Put the power adaptor itself behind the Wizzard, or under the table on the floor -- somewhere out of the way, but where the cables won't be strained or pulled out accidentally.



Then plug the adaptor's second cord, with the 2-pin power plug, into the power point.

You can turn on the power point at this stage, it won't do any harm. Now take a look at the back of your TV set, where the cable from your aerial connects to it.



If the back of your TV set has a round aerial socket (see the "READ THIS FIRST" section at the front of the book), just:

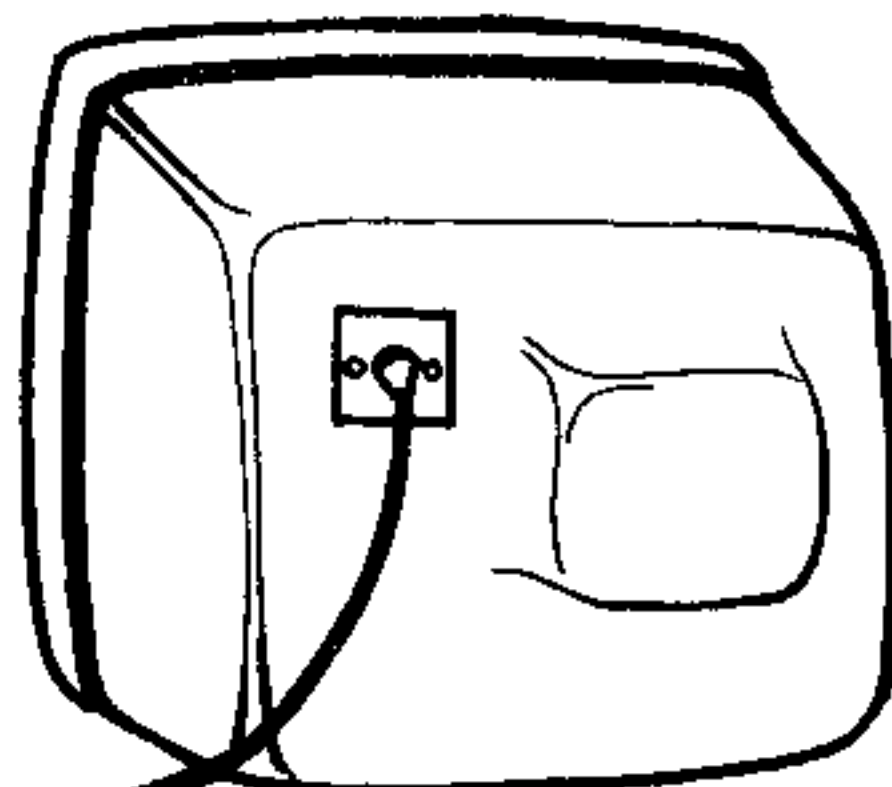


Plug the cable from Wizzard's aerial switch box into it.

from TV  
aerial



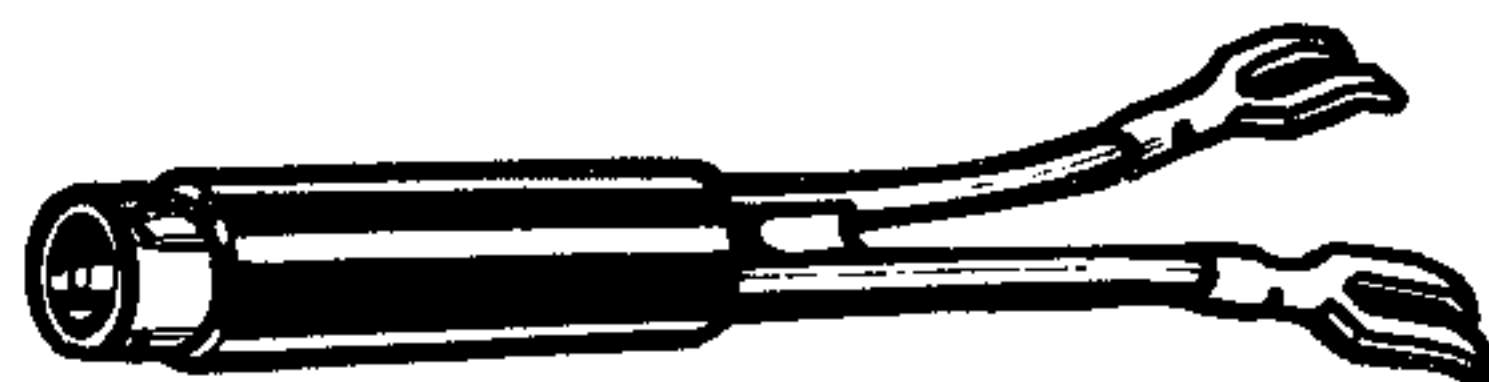
Then plug the TV aerial lead  
into the "aerial" socket on  
the switch box, and the lead  
from Wizzard into the "game"  
socket.



from Wizzard

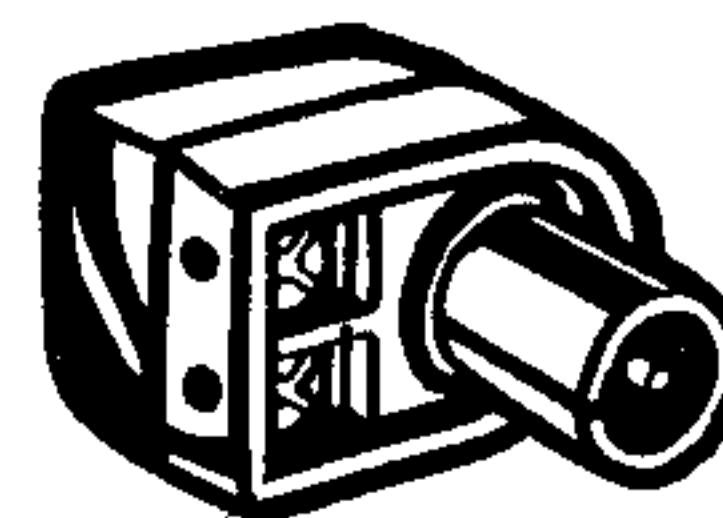
If your TV has a couple of screw terminals instead of the round socket, you will need to use the baluns mentioned in the READ THIS FIRST section. But don't despair. There's no real problem.

Use the "75 ohm to 300 ohm" TV-game type balun (L-4454) to adapt your TV set's aerial terminals so that they take the plug from the Wizzard's aerial switch box. Here's another picture of it:



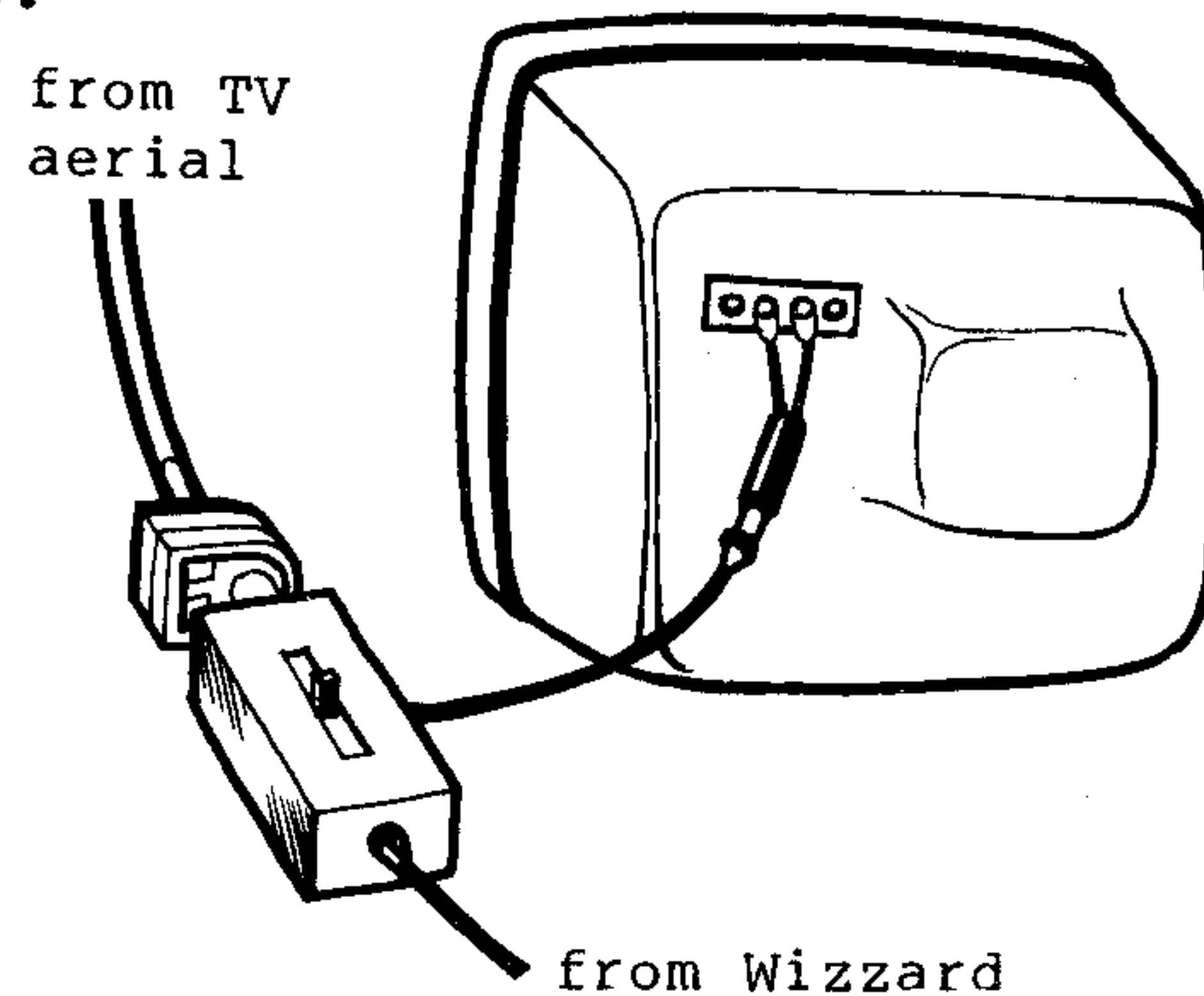
The wires with the "spade" lugs connect to the terminals on your TV, while the socket at the other end takes the plug from the Wizzard's aerial switch box.

You'll probably need the other balun, the "300 ohm to 75 ohm" type (L-4456), to adapt your twin-lead aerial cable so that it can still connect to the switch box. Here's another picture of the balun we mean:



The round plug on one side goes into the aerial socket on the switch box, while the screw terminals on the other end take the wires from your TV aerial cable.

So if you've had to use the baluns, your connections should look like this:



Turn on your TV and set the channel selector to channel 1 in the normal (VHF) channel range.

Also set the Wizzard's aerial switch box to the "game" position.

You're now ready for the last big step: plugging the BASIC cartridge into the Wizzard. But before you do this, make sure the Wizzard's on/off switch is in the OFF position.

In fact you should always make sure this switch is in the OFF position before you plug in or remove a cartridge from your Wizzard. Otherwise either the Wizzard itself, or the cartridge, may be damaged and you won't be able to use it again until it is fixed!



Switch off the Wizzard then plug in the BASIC cartridge with its label facing up.

Again do this carefully, so you don't damage anything. Push it all the way in, so that the groove is in line with the edge of the Wizzard.



If you've got the cartridge around the right way, and properly lined up, it slips in easily without too much effort.

Once the cartridge is in place, you can turn on the Wizzard again. This time you should be greeted by a green picture on the TV screen, showing this message in black:

CREATIVISION BASIC  
BY VTC  
COPYRIGHT 1982

>H

Now see if you can do this quick quiz before going on to Chapter 3, which tells you more about how your Wizzard works.

#### QUICK QUIZ 2

1. Why might you need baluns to hook up your Wizzard? (tick one)
  - a. Because your power point is the wrong colour
  - b. Because your telephone is too far away
  - c. Because your mother likes the look of them
  - d. Because your TV set has the wrong sort of aerial socket
2. What must you do to your Wizzard before you plug in the BASIC cartridge? (tick one)
  - a. Say "Hold still, this will tickle!"
  - b. Turn Wizzard off
  - c. Switch off your TV
3. What channel should your TV be on?
4. Why couldn't the Wizzard talk to his TV set?

### CHAPTER 3 -- A bit more about computers

In Chapter 1 we explained that a computer was like a high-speed electronic slave. Well it is, in the sense that it does exactly what you tell it to do -- and it does it very quickly.

But even though it is so quick, it is really far more stupid than any human slave could ever be.

It will do exactly as it is told but it won't do anything unless you tell it to!

Perhaps the main reason why people find computers a little hard to get used to is that they usually obey the instructions in a program in much less time than it takes to work out or give them those instructions.

And that can seem pretty nifty!

The main thing to remember about the way a computer works is that it can do only two of the many things that we can do with our brains.

We can learn, remember, have ideas, work out answers to questions, think of jokes and all sorts of things.

The computer can only remember (using its MEMORY) and obey orders (using its PROCESSOR). But it does those two things very quickly.

Now because a computer has absolutely no intelligence, it can't decide anything from what it has done before.

It can't guess either, or work something out for itself when you don't explain it properly.

It can't even remember anything unless you tell it to.

So the instructions in its program must be very clear and complete, right down to the smallest detail.

It relies on YOU to do the thinking...

Sounds easy enough, doesn't it? But it isn't quite as easy as it seems. Explaining something to a dum-dum can be tricky!

Clever as we humans are, we are just not used to explaining things clearly or completely. And of course we don't have to be all that careful when we're talking to another person, because we can rely on them "knowing what we mean".

But this isn't good enough when you write a computer program. So what usually happens when you try it out is that you find the computer "gets it wrong".

Or it doesn't do that job at all -- it does something entirely different!

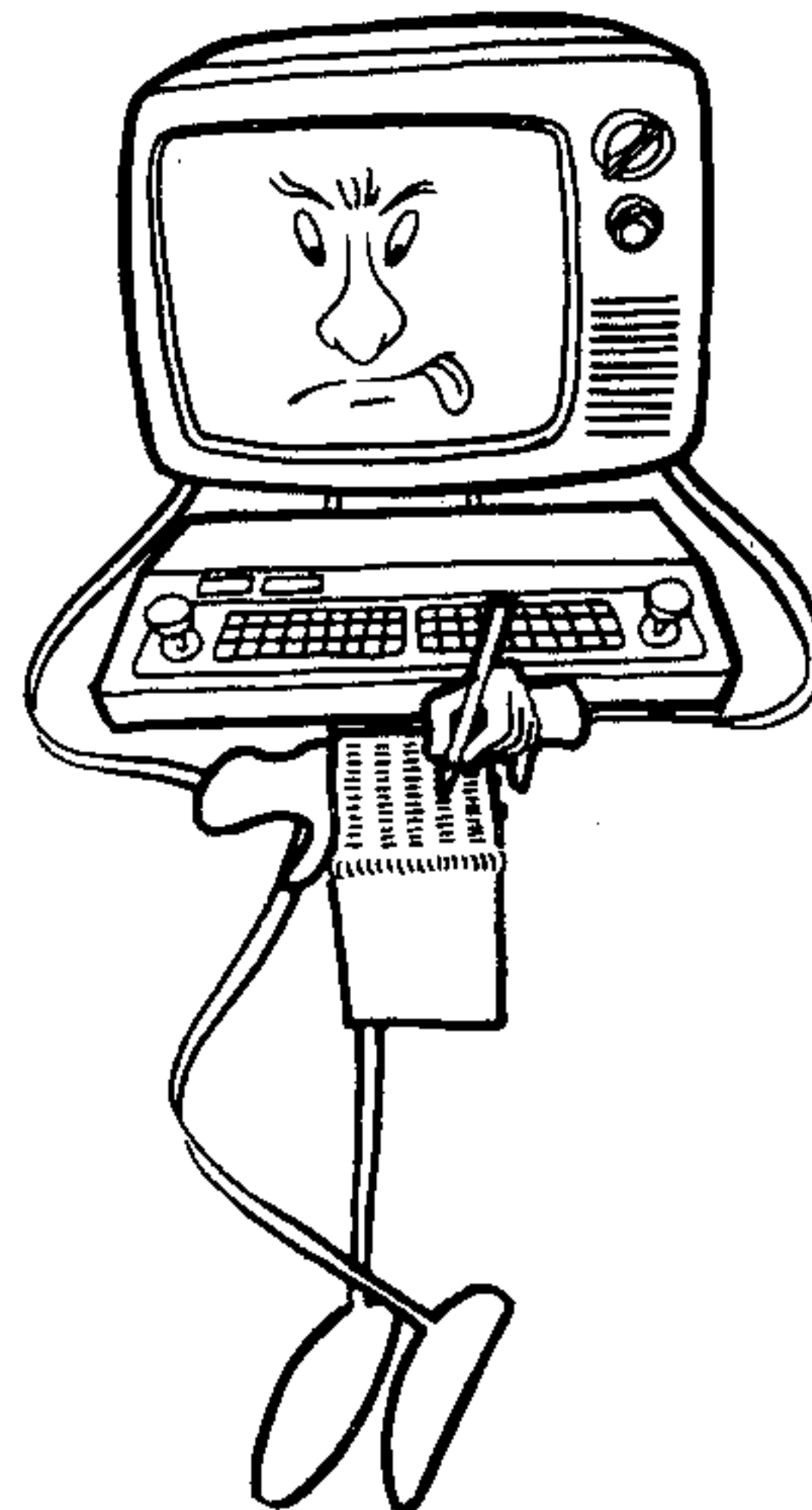
When this happens (and it does all the time, even with experienced computer programmers), it is almost NEVER because the computer has made a mistake.

At least 999,999 times out of 1,000,000 it is because YOU, the programmer, have blundered.

You'll have left out an important step, or told it to do one thing when you should have told it to do the opposite, or something like that. And the computer, being a completely obedient dum-dum, will do EXACTLY what you tell it. No more, and no less.

Here's a very simple example. Suppose you want the computer to add two numbers together, say 2 and 2. Like most of us, you'd simply tell the computer to take the first number and add the other one to it. But if you write a program to do just that, you'll find nothing seems to happen. Why?

Because you also have to tell it to show you the answer...



With another person, you wouldn't need to tell them this, because if you simply said: "Add 2 and 2", they would assume that you also wanted them to tell you the answer. But a computer can't assume anything, so you have to work out and tell it every single thing you want it to do.

Funny, isn't it? We humans are supposed to be the bright ones, but a stupid computer can show us up every time, simply by doing exactly what we tell it!

So a machine is forcing you to think and explain yourself more clearly. If you don't, it won't do what you want!

### LOADING in, and RUNNING programs

OK, you may say, you get the message about computers being dumb, so that we have to tell them every little thing we want them to do. And you see why that forces US to be much more careful than if we were teaching another person how to do something. But what exactly does a computer program look like, and how do you get the computer to do what it says?

We'll start looking more closely at programs themselves shortly. For the moment, just think of it as a list of simple instructions. Like DO THIS, THEN DO THAT, and NOW DO THE OTHER.

#### IMPORTANT

You get the computer to obey your program by first loading the program of instructions into the part of the computer known as its MEMORY.

What it does is simply store your program, so that you don't have to keep feeding it into the computer over and over again.

#### IMPORTANT

When a program is loaded and in the computer's memory, you then get the other main part of the computer -- called a PROCESSOR or "CPU" (short for "Central Processing Unit") -- to follow the instructions in the program. This is called RUNNING the program.

It RUNS your program by doing two simple steps over and over again. First it FETCHES the first instruction from its MEMORY (which is just the same as you do when you remember something -- only much faster), then it looks at this instruction and does what it says. This is called EXECUTING the instruction. Then it fetches the next instruction, and executes it too. Then it fetches the third instruction, and executes it. And so on...



Not all that complicated, is it? Just a simple matter of looking at each instruction, and doing what it says. The main thing to remember is that it all happens very, very fast.

### How YOU "talk" to the Wizzard...

You can load the program into the computer's memory in a variety of ways. The first time round, you have to spell it out by touching the letters on the computer's special touch-sensitive keyboard. Once it's in, though, you can get the computer to SAVE it for you by recording it on a cassette tape.

This is the same thing as telling the computer to learn the program so that it doesn't forget it.

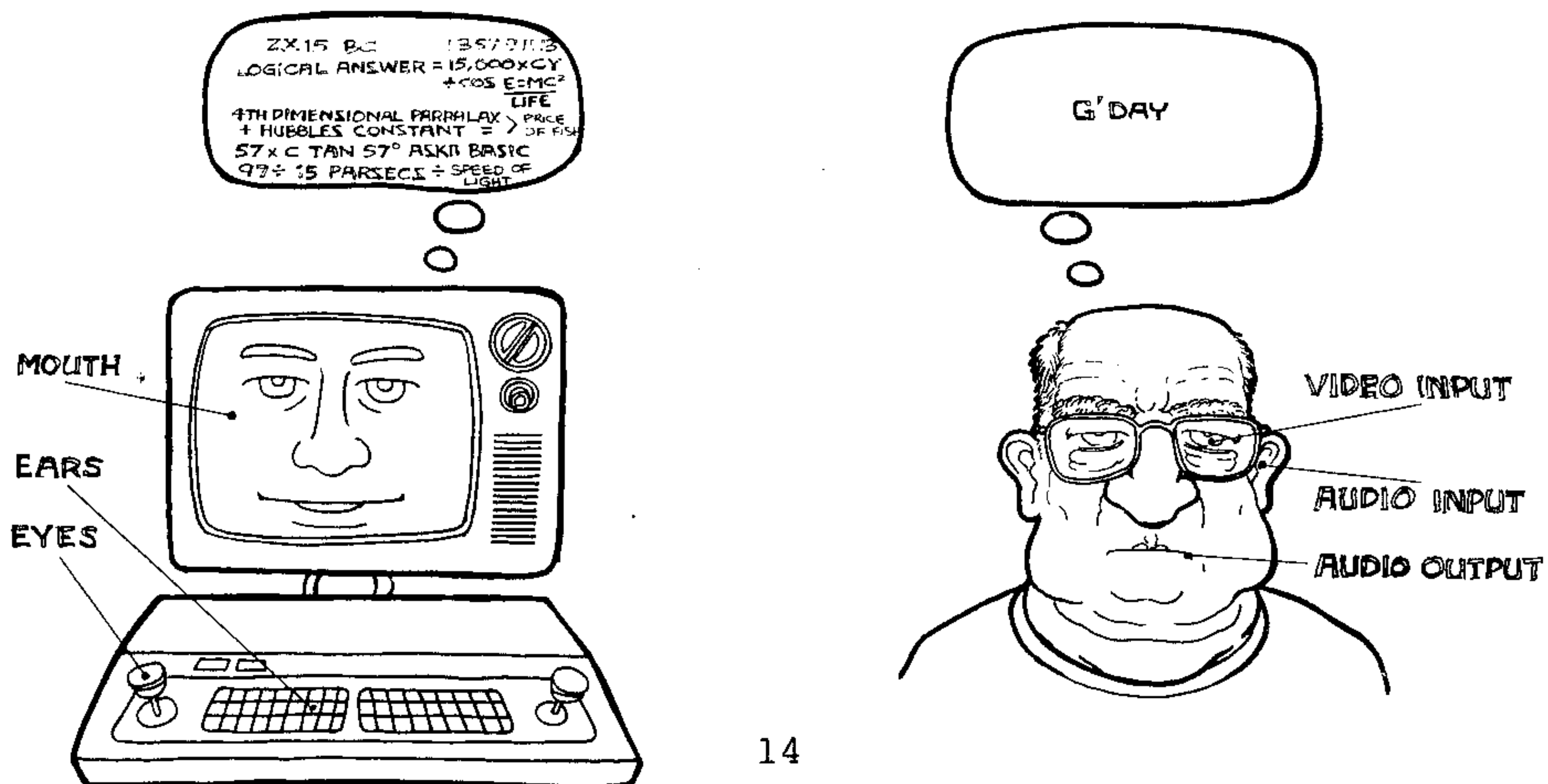
From then on, all you have to do is LOAD the program in again from the tape, each time you want the computer to do that job. The computer will remember how to do it.

Now a computer doesn't have ears to hear with or a mouth to speak with like we do. So we have to use a different way of COMMUNICATING with it: typing on its keyboard.

Computers are not even bright enough to understand English (which is FAR too complicated for them) so we have to learn a language which IS easy enough for them to follow.

Different computers understand different languages, but the Wizzard can follow programs written in a language called BASIC, which is very easy to learn because it is a lot like English.

These are some of the differences between people and computers:



You also use the keyboard to give the computer its special orders or COMMANDS.

For example you might type in the command "RUN" which tells it to start running the program you've just loaded into its memory.

Or you might type in "LIST" to get it to remember the whole program and show it back to you, so that you can check it for mistakes.

...and how it "talks" to you!

How does the computer show you things?

Generally it shows you things and lets you know what it's doing by printing messages on its VIDEO SCREEN.

NOTE

So the keyboard of a computer is rather like its eyes or ears, used for receiving information and its video screen is rather like its "mouth" -- which it uses for giving information.

In the case of your Wizzard computer, its video screen also happens to be the screen of your TV set.

One more thing. When you tell the computer to RUN the program stored in its memory, it doesn't forget it afterwards. So you don't have to load it in again every time you want to run it.

The program stays in the computer's memory until you tell it to forget it, or load in another program (for doing another job).

Until this happens, your computer will run the program as many times as you like.

The only other thing that can rub out the program is to turn off the computer's power switch, or pull out its plug. This will always give a computer instant amnesia.

In fact pulling out the power plug is the ultimate weapon -- a sure-fire way of showing any computer who's boss!

That's all for the moment, But here's another quick quiz.

### QUICK QUIZ 3

1. What does a processor (or CPU) do? (tick one)
  - a. Follow the orders in your program
  - b. Talk to you
  - c. Make cake mixes
  - d. Disobey orders
2. What happens when Wizzard executes an instruction? (tick one)
  - a. It kills it
  - b. It runs away with it
  - c. It obeys it
  - d. It remembers it
3. What is BASIC?
4. What does it mean when you load a program? (tick one)
  - a. You make it heavier
  - b. You put it into the computer's memory
  - c. You give it a long, cool drink
  - d. You see how long it is
5. What are you telling Wizzard to do when you type in the word "RUN"? (tick one)
  - a. You're telling it to go faster
  - b. You're telling it to go away
  - c. You're telling it to execute the instructions in the program
  - d. You're telling it to remember the program
6. What does Wizzard do when you tell it to "LIST"? (pick one)
  - a. It leans to one side
  - b. It recites the alphabet
  - c. It shows you the program you have loaded into its memory
  - d. It obeys the program's instructions
7. Why did Wizzard enter the sports carnival?



## CHAPTER 4 -- Teaching your Wizzard some tricks

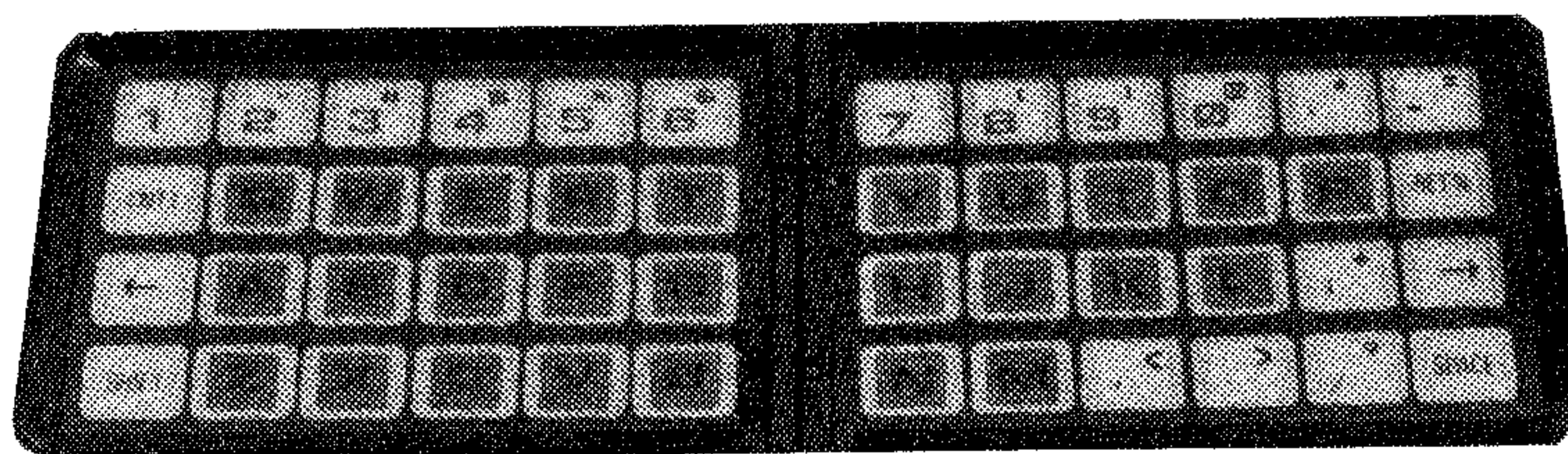
By now you should understand a bit about how a computer works.

You have also unpacked and connected Wizzard to the power point and the TV, so it is time to start getting some programs RUNNING.

Programming is a bit tricky so we will go slowly, one step at a time, with some explanation in between steps.

If everything is switched on, you should see the CREATIVISION message on your TV screen.

First, here is a picture of the keyboard:



### Getting Wizzard to say HELLO

This simple program will make Wizzard say hello:



Use your keyboard to type these lines exactly as they are written here (we will explain why they have to be typed this way in a minute).

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



At the end of each line hit the square marked RET'N on the keyboard.



### Typing mistakes are easy to fix

If you make a typing mistake, don't worry. It is easy to fix. You can correct mistakes in 2 ways:

Firstly if you notice your mistake before you hit RET'N to end up the line, you can simply go back to the place of the mistake by hitting the square marked with a backwards facing arrow (<--) and type in the right thing instead.

If you don't notice the mistake until after you have hit RET'N, you won't be able to move back to correct it using the arrow -- but DON'T WORRY. You can type the line in again, (correctly of course), by using the same number as the line with the mistake in it.

This is one of the reasons we use line numbers. If two lines have the same number at the beginning, the computer will ignore the first line you typed in and only notice the second one.

### Why we use line numbers

We explained earlier that a computer only does what you tell it to and that it is important to tell it everything it needs to know to run your program.

If someone tells you to put on your shoes and socks like this:

"Put on your shoes -- Oh I forgot, put on your socks first"

You would know to obey the second part of the command first and the first part -- putting on your shoes -- afterwards. But a computer can't work this out, so you have to have some way of telling it which commands to follow first, which to follow second and so on.

You can do this by giving a number to each instruction line in the program.

All Wizzard then has to do is follow them in order, starting at the line with the lowest number and going on to the one with the highest number.

It always does the instructions in the order of their line numbers, unless one of the instructions tells it to do something else.

Do you just number the lines 1, 2, 3 and so on? Well you can, but it's usually not a good idea. It's better to give them numbers with gaps between them, like 10, 20, 30 etc.

Perhaps you have already guessed why:

If you don't leave gaps, there won't be any room for adding in extra instructions later. And, quite often, you do have to add extra ones in, because if you're like most people, you'll find you have left some out!

Let's say, for example, that you discover you've left out an instruction between two lines. If they're numbered "7" and "8", you're going to be in trouble, but if they're numbered "70" and "80", all you have to do is give the extra line a number in between the two -- like "75".

Here are a few facts about instruction lines in a program:

**NOTE**

Every instruction line in a program must have a line number. We suggest the numbers go up in steps of 10 so you can add other instruction lines later. Instructions on the Wizzard must take no more than 25 characters (including spaces) after the number. However, you can squash up your instructions by leaving out the gaps between words if necessary.

#### Back to your program

Well the Wizzard is still waiting to do something!

So let's give it something to do with the program you've already typed in.

Firstly, try giving it a direct COMMAND.

**NOTE**

A direct command is like an instruction in a program, but it doesn't have a line number. This is because it is not part of a program. It tells the computer to do something with the program and to do it straight away.

Commands can be included in program instruction lines as well but they don't have to be in a line before the computer obeys them.

NOTE

One command is LIST -- this tells Wizzard to show your completed program on the TV screen.

If you've made no corrections, all Wizzard will do is show you your program as you typed it in. But if you had to retype a line because you made a mistake the first time, you'll see that the wrong line has been removed.

NOTE

Another command is RUN, which tells Wizzard to RUN your program by obeying the instructions you have given it.



Type in LIST and hit RET'N to get a list of your program on the TV screen.  
Then type RUN and hit RET'N to get your instructions EXECUTED.

If you typed the program in exactly as it appeared in the book, you should see lots of HELLOs on the screen.

What did this program really tell Wizzard?

Your first instruction was:

```
10 PRINT "HELLO"
```

Can you guess what that means? Right! It is telling Wizzard to print the word HELLO on the screen for you to see.

NOTE

Whenever you want Wizzard to print a message on the screen, you simply key in in instruction line which says PRINT and then gives the message you want printed, with quotation marks (") at each end.



These marks (") must always go round the message you want shown on the screen, or Wizzard won't print it.

The second line you typed was:

```
20 GOTO 10
```

You were telling Wizzard to go back to line 10 and start again.

That's why Wizzard printed HELLO not once, but over and over again. And it will keep printing HELLOs until you stop it.

This is known as a programming LOOP. We will explain more about loops in the next chapter.

For the moment, there are a couple of ways to stop the HELLOs.

One way is to turn off the power, which will make Wizzard forget the program altogether. The other way is to hold the CNT'L key down and at the same time press C. This stops the program running without making Wizzard forget it.



Use the second way now:  
hold down CNT'L and  
press C. What happened?

The screen stopped, with the last few HELLOs still on it, and a READY message to show that Wizzard is waiting for further orders.

Now do this quick quiz and you will be ready to go on to chapter 5 and learn some more programming.

#### QUICK QUIZ 4

1. Why do you give each line in your program a number? (pick one)
  - a. To show how important it is
  - b. To be tidy
  - c. To teach Wizzard how to count
  - d. To show Wizzard the right order for following your instructions
  
2. What is the difference between an instruction line and a command? (pick two)
  - a. One is said in a firmer tone of voice
  - b. One is part of a program and the other tells Wizzard what to do with the program
  - c. One is more informative
  - d. A command doesn't have a line number
  
3. Which is a better way to number instruction lines
  - 1, 2, 3, 4, 5, etc or
  - 10, 20, 30, 40, 50 etc
  
4. Why?
  
  
  
  
  
  
  
  
  
  
5. What is a programming loop? (Pick one)
  - a. The place where Wizzard hangs itself
  - b. A thing like a magnifying glass, to look at small programs
  - c. What happens when your program says GOTO an earlier line

## CHAPTER 5 -- Becoming a programmer

As you already know, a PROGRAM is a list of instructions in numbered lines telling Wizzard what you want it to do.

You also know that it's important to tell Wizzard everything you want it to do, and in the right order.

Let's look more into how you work out a program, using the BASIC language for your instructions.



Make Wizzard forget the last program by typing NEW and hitting RET'N

Now the memory is empty and you can start work.

We'll do some adding up first. If we wrote a PRINT instruction of the sort we looked at in chapter 4, like:

```
10 PRINT "2+2"
```

it would just print out the part inside the quote marks ("). We wouldn't get it to work out the answer. But we can get it to work out the answer by making a simple change...

### NOTE

To make Wizzard work like a calculator, give it the same sort of PRINT instruction, but WITHOUT the quote marks.



Type in these two lines EXACTLY as they are here (and hit RET'N at the end of each line).

```
10 PRINT 2+2  
20 END
```

Line number 10 tells Wizzard to print the answer to the sum 2 + 2. And line 20 means just what it says -- END. This line tells Wizzard that you have finished the program and it can stop following instructions until you give it some more.



Now get Wizzard to do this sum, by running the program:



Type in the command  
RUN and then press  
the RET'N key.

If you typed in the program exactly as shown, Wizzard will have obediently printed a 4.

Incredible, isn't it -- you've just got your Wizzard to tell you the sum of 2 and 2! Aren't computers fantastic?

### Adding another line to your program

Let's add another instruction line to this program. It's easy because we've left plenty of space between the line numbers.

Instead of just printing the answer to our sum, 4, let's get Wizzard to print out the sum as well, so it shows the whole thing: 2 + 2 = 4.

You already know how to do this from the last chapter.

All we need is a line at the start of the program (before line 10) which says: PRINT "2 + 2 =" and Wizzard will print what's between the quotes BEFORE it prints the 4.

The new line number must be lower than 10 if Wizzard is to follow that instruction first. We suggest number 5.



So key in the extra line  
with this number (ending  
it by hitting RET'N).  
Then LIST your program.

It should now look like this:

```
5 PRINT "2 + 2 ="  
10 PRINT 2+2  
20 END
```

Let's run this program and see what happens. The screen should say this:

```
2 + 2 =  
4
```

Not quite right yet, is it? It would look better all on one line. The way to do this is by re-typing line 5 like this:

```
5 PRINT "2 + 2 = ";
```

Did you spot the added semicolon at the end? It's important...

**NOTE**

A semicolon at the end of the PRINT instruction tells Wizzard NOT to shift its printing down to the next screen line when it has printed your message.



RUN the program now, and you'll see the semicolon has fixed our problem.

The screen should show this:

```
2 + 2 = 4
```

Try this quiz before moving on to chapter 6.

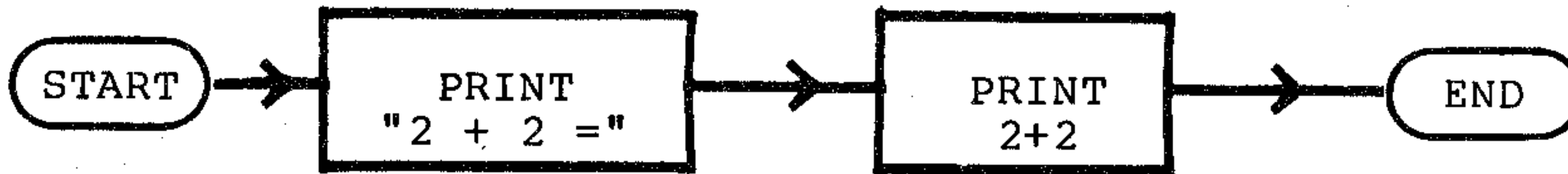
QUICK QUIZ 5

1. How would you change our program to give you the answer to the sum  $5 + 3$ ?
2. How would you change the program to work out the answer to the subtraction  $7 - 4$ ?
3. What does an END instruction do? (Pick one)
  - a. Tell the Wizzard to shoot itself
  - b. Tell the Wizzard you're shooting yourself
  - c. Just tell it that your program ends there
4. What does a semicolon do when you add it to the end of a PRINT instruction? (Pick one)
  - a. Make the Wizzard pause for breath
  - b. Tell it NOT to shift down to the next printing line on the screen
  - c. Tell Wizzard you don't quite know what you're doing

## CHAPTER 6 -- More programming

### Drawings of the program

Here is a drawing of the program you gave Wizzard at the end of chapter 5:



This drawing is called a FLOWCHART and it shows everything the Wizzard does in the right order. A flowchart can be very handy when you are working out programs.

There is something on the flowchart that you don't have written in your program. It's the START sign.

Always put this at the beginning of your flowchart. It may not seem much use with a simple program like this, but when your programs become more complicated, it comes in very handy to know where everything starts.

If you look at the chart, you should notice that the START and END are in oval-shaped boxes and the other 2 instructions are in rectangular boxes.

#### NOTE

It's useful to put rectangles round simple instructions and ovals round the START and END signs, because flowcharts can get complicated when you write longer programs.

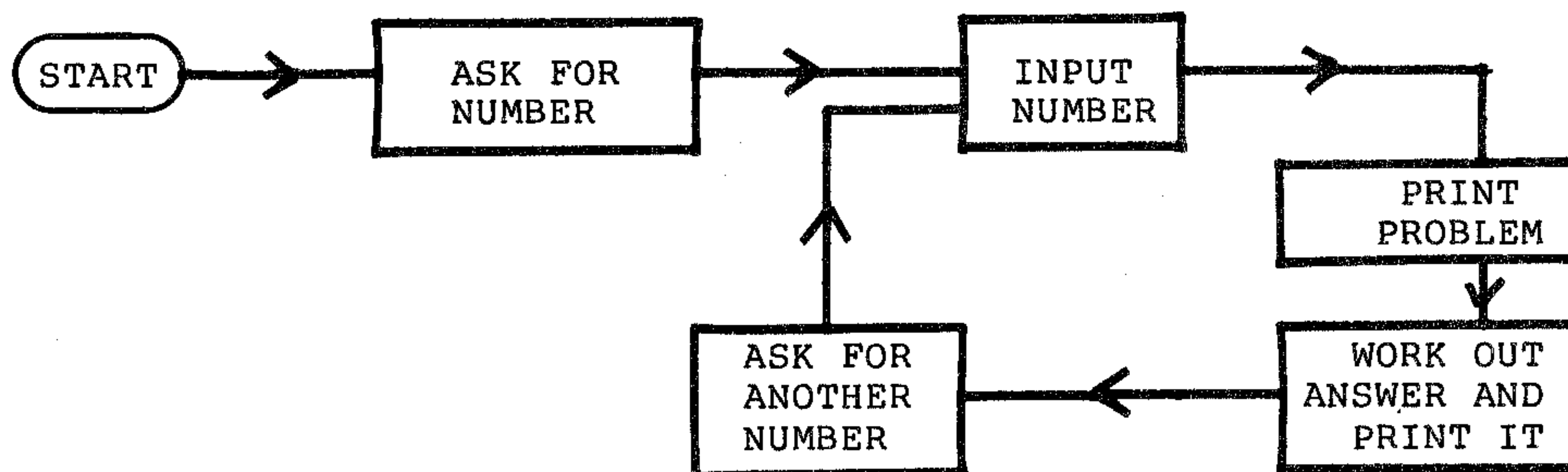
There are lots of different types of instructions, (you'll find some of them later in this book) and it is important to be able to tell quickly what type of instructions they are. You can do this if you have different shaped boxes around them.

We will show you the different boxes you should use for special instructions later. For the moment, all you have to remember is the oval and the rectangle.

Programmers usually draw up flowcharts then write out the program with line numbers before they type in the instructions. It is good practice to do this too, so from now on, you should have a pencil and paper to help you with your programming.



Got pencil and paper handy? Right! We thought you might like to tell Wizzard how to do some number tables. Here is a flowchart of the program you can use:



The best way to work out the program from the flow chart is to follow the arrows and give each box a line number, then work out what your instruction line should be so you can get Wizzard to do what's in the box.

Let's do this one together.

We don't have to give START a number but the first box should be number 10. In this box, Wizzard must ask for a number. So the instruction line could be something like this:

```
10 PRINT "GIVE ME A NUMBER"
```

The second box involves getting a number for Wizzard to use in the equation (this is the maths problem you want it to do).

**NOTE**  
There is a special instruction for this: an INPUT statement.

If you use it in the next line, it will look like this:

```
20 INPUT A
```

You can use any letter of the alphabet for the input -- not just A -- but don't put a number in. It must have a letter.

Line 20 is telling Wizzard to let you type in a number -- any one you like. When you have typed in your number, Wizzard keeps it in its memory in a place called A.

### A bit about variables

That last paragraph may seem pretty confusing so we thought you might like to learn a little about variables before we go on.

Wizzard has a very efficient memory. It learns things (or stores them) in various parts of its memory and gives each part an "address".

This is just like the addresses of different houses in a street. And when Wizzard remembers something for you, it is just like going to a particular place and fetching what it finds there.

When we tell the Wizzard to INPUT A, we are telling it to do two things. One is to let us type a number in. The other is to put the number, whatever it is, away in its memory -- marked A.

When you use letters instead of numbers like this, you are telling Wizzard to reserve a space in its memory for a number. You can then change the number (making A = something else) and Wizzard will do the same calculation for you with a different number. Because you can change the actual number in A, A is called a VARIABLE.

### Meanwhile, back to our program...

The lines you have worked out so far are:

```
10 PRINT "GIVE ME A NUMBER"  
20 INPUT A
```

The next box in the flowchart is where Wizzard shows you what it is going to do with the number. This means that you have to work out what you want Wizzard to do, then write an instruction line (number 30) telling Wizzard to print it.

Let us say you want Wizzard to multiply the number by 4. Try this line:

```
30 PRINT A;" X 4 =";
```

This tells Wizzard to print the number that it has stored in A and then print "X 4 =" after it (remember, the semicolon tells it NOT to move its printing down to the next line). If the number you enter is 8, Wizzard will show 8 X 4 =, when it executes the instruction.

Did you notice that there were 2 semicolons in line 30?

Good. The first one lets you use one PRINT instruction for 2 different types of printing.

Line 30 tells Wizzard to find the number in A, then print it, then add the bit in quotes straight after it, so both parts of the printing appear on the same line.

Clever - huh?

The next step is to get Wizzard to do the actual calculation and print the result.

### Arithmetic functions

Wizzard can add, subtract, multiply and divide, just like a calculator.

But if you look at the special keyboard, you will find no division sign there. And if you just use an X for multiply, how is Wizzard going to know when you mean it to be a multiplication sign, and when you mean it to be the letter X?

Yes, we know people could work it out, but Wizzard isn't people and it is too dumb. So, to make it absolutely clear for the poor, stupid computer, we have a series of special symbols for the different arithmetic functions.

#### NOTE

We use an asterisk (\*) for multiplying, a dash (-) for subtracting (just like the normal minus sign), a plus (+) for adding and a diagonal or "slash" (/) for dividing.

To tell Wizzard to work out 2 X 4, you would type in 2\*4. If you wanted it to work out 6 divided by 3 -- can you guess how to write it?

If you had 6/3 you were right. Now let's get back to this program.

The next step (line 40) must tell Wizzard to do 2 things. It must multiply the number in A by 4, and it must print the answer. Can you work out how the line should go?



It goes like this:

```
40 PRINT A*4
```

The next box gets Wizzard to ask for another number. I know you can already do this, so write line 50 as a PRINT instruction.

### Looping around

When Wizzard has asked for another number you want it to go back to step 20 (where you can type in another number to go into A).

This is called a LOOP, and the easiest way to make such a loop in your program is with a GOTO instruction.

#### NOTE

When you find a loop in your flowchart, an easy way to do this is with a GOTO instruction

The instruction line will be easy. Just put this:

```
60 GOTO 20
```

### The finished program

Your program should now look like this:

```
10 PRINT "GIVE ME A NUMBER"  
20 INPUT A  
30 PRINT A;" X 4 =";  
40 PRINT A*4  
50 PRINT "GIVE ME ANOTHER NUMBER"  
60 GOTO 20
```



Type this program in and LIST it so you can check it and correct any typing mistakes.

Now don't be shy -- try running it! (Type in RUN and hit RET'N)

Wizzard has asked you for a number, you'd better type one in.

Why not make it 1. Type in 1 and hit RET'N.

Wizzard should have told you what 1 X 4 is and asked you for another number.

Type in 2 this time, and again hit RET'N. Now it has told you 2 X 4, and asked you for another number -- why not try 3?

By typing in a different number each time, you will be able to get Wizzard to go right through the 4 times table for you.

What's that I hear? You would like it to do another table?

OK, you can do this, but you'll have to change two lines in your program: line 30 and line 40. Let's get it to do the 9 times table.

First, you'll have to stop the present program from running:



Hold down CNT'L and press C,  
to stop the program but keep  
it in Wizzard's memory.

Now type in a new line 30:

```
30 PRINT A;" X 9 =";
```

... and a new line 40:

```
40 PRINT A*9
```

Now type in LIST and press RET'N, and you will see your new program on the screen.

Run it, and you will find that your Wizzard can now multiply any number you give it by 9.

There is no quiz in this section. We think you've had quite enough hard work for now!

Just glance through the list on the next page, which shows all the things we've looked at so far. Then have a rest before going on to the next chapter -- you've earned it!

## WHAT WE HAVE LEARNED SO FAR

Here's a summary of the things we have covered so far.

LIST tells Wizzard to show you the program as you have written it.

RUN tells Wizzard to execute the program - by obeying the instructions in it.

PRINT tells Wizzard to print something on the screen. If you put quotes around the thing you want printed, Wizzard will print what's inside the quotes. To get it to work like a calculator and print the result of a calculation, don't use quotes.

INPUT is an instruction that gets Wizzard to stop and let you enter a number of your own choice.

VARIABLES are names for places where Wizzard can store different numbers. An example of a variable is the letter you use to tell Wizzard where to store a number you will INPUT.

GOTO is an instruction to get the Wizzard to jump to another part of the program. If you say GOTO 20, Wizzard will go to do the instruction on line 20. If the GOTO 20 instruction is on a later line, this makes what is called a LOOP in the program.

If you hold down CNT'L and hit C, you will stop a program running without making Wizzard forget it.

FLOWCHARTS are drawings of all the things you want Wizzard to do in a program. A flowchart has START and FINISH in ovals and all other instructions (that you know so far) in rectangles.

END tells Wizzard that there are no more instructions in the program.

A SEMICOLON in a PRINT instruction line tells Wizzard not to go to another printing line. It lets you use one PRINT instruction for printing two things, one straight after the other.

Wizzard can carry out calculations for you using these signs: + for addition, - for subtraction, / for division and \* for multiplication. It will carry out other calculations as well, but those are mentioned later in this book.

## CHAPTER 7 - More on tables

There is a way to get Wizzard to work out number tables automatically for you, without you even having to input any numbers.

```
> Remember we talked about variables <
> being addresses that Wizzard uses <
> to keep numbers in? <
```

We also said that you could change these numbers and that was why they were called variables.

Well - when you wrote the program in chapter 6, you changed the variable yourself each time you reached line 20 and had to input a new number. This time we will show you how to program Wizzard to change the variable for you, so that it will work out the full table all by itself.

Let's do the 4 times table again.

The first line of the program will look like this:

```
10 A=1
```

### NOTE

This is the way you tell Wizzard IN THE PROGRAM what is going to be the value of a variable like A, instead of having to enter the value using an INPUT line.

The next step is to get Wizzard show you what it is doing with A. You can do this in the next line.

This can be just like line 30 in the first program. It should look like this:

```
20 PRINT A;" X 4 =";
```

Because you have told Wizzard that A = 1, it will print 1 X 4 = when it executes this instruction.

The next line is simple:

```
30 PRINT A*4
```

Now we get a little more complicated...



We've now got to make Wizzard work out the next number for A, which will be 2. The next time round it should make it 3, then 4 and so on. To do this we give it an instruction like this:

```
40 A=A+1
```

This looks strange, and would be wrong if you tried to put it in an ordinary arithmetic problem. But remember here A doesn't just stand for a number - it really stands for the address of the number in Wizzard's memory.

<p>NOTE</p> <p>A=A+1 tells Wizzard that the next number for A must be one bigger than the last one. So if the last number in A was 1, the next one will be 2, and so on.</p>
--

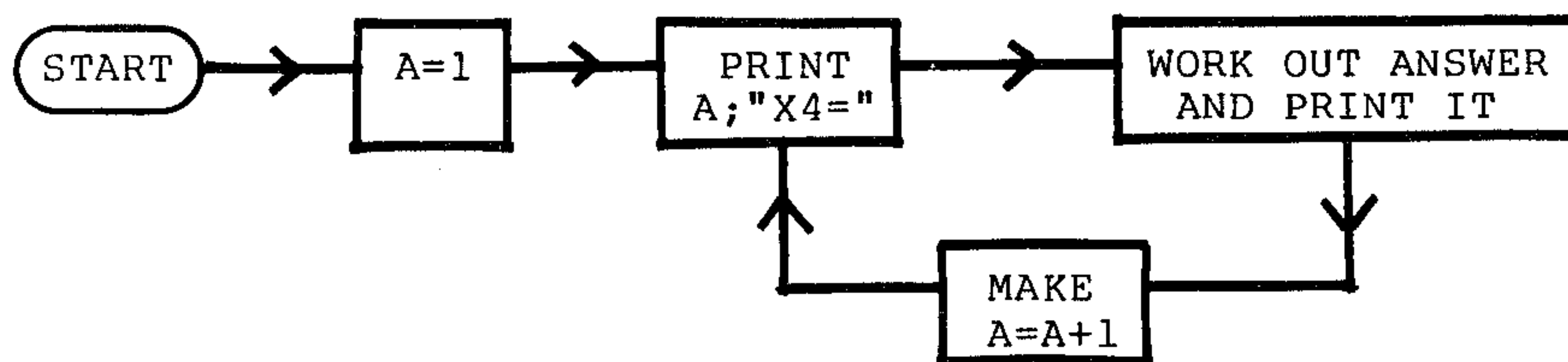
Now we need a loop in the program so that Wizzard will go back to multiply the next A by 4. It looks like this:

```
50 GOTO 20
```

So your program will look like this when it is written:

```
10 A=1
20 PRINT A;" X 4 =";
30 PRINT A*4
40 A=A+1
50 GOTO 20
```

Here is the flowchart for this program.



Why not test this program out?

Type in NEW and press RET'N to get Wizzard to forget the last program. Then type the new one in, just as it appears above.

Ready to run it? First check it by typing in LIST and pressing RET'N. Then if it's alright, type in RUN and press RET'N.

Isn't it cute the way Wizzard just keeps on and on multiplying numbers by 4? It will go on for ever unless you stop it -- doesn't that give you a feeling of power!

The flowchart shows this -- at present, the program has no END. But we can get it to stop the multiplications automatically at a given point (say after A=12), by adding a new instruction.

#### How to stop a loop going on for ever

Let's say you just want Wizzard to do the table up to 12 X 4 and then stop.

At the moment, the only way you can stop it is by holding down CNT'L and pressing C.



Do this now, to stop the program so that you can concentrate on what we are going to tell you.

Ready? Right. What we have to do is tell Wizzard to stop when the size of variable A has passed the value 12. To do this we use a new instruction: the IF ... THEN instruction.

It will look like this:

```
45 IF A=13 THEN END
```

#### NOTE

The IF ... THEN instruction gets the Wizzard to make a decision. It has to look at something (here the value of A) and decide IF it has reached a certain size (here 13). IF it has, THEN Wizzard follows the instruction on the rest of the line. But IF it hasn't reached the size yet, then Wizzard ignores the rest of the line and "drops through" to the next line.

We have made the IF ... THEN line 45 so that Wizzard will look at it before looping back to line 20.

Why not try this modification to the program by typing in the extra line now.



Type it in, then LIST the program to make sure it is there. If all is well, try running your improved program to see what happens.

You should have seen Wizzard do the "4 times" table up to 12 X 4, then stop by itself just as we wanted. (If it didn't, you must have made a mistake!)

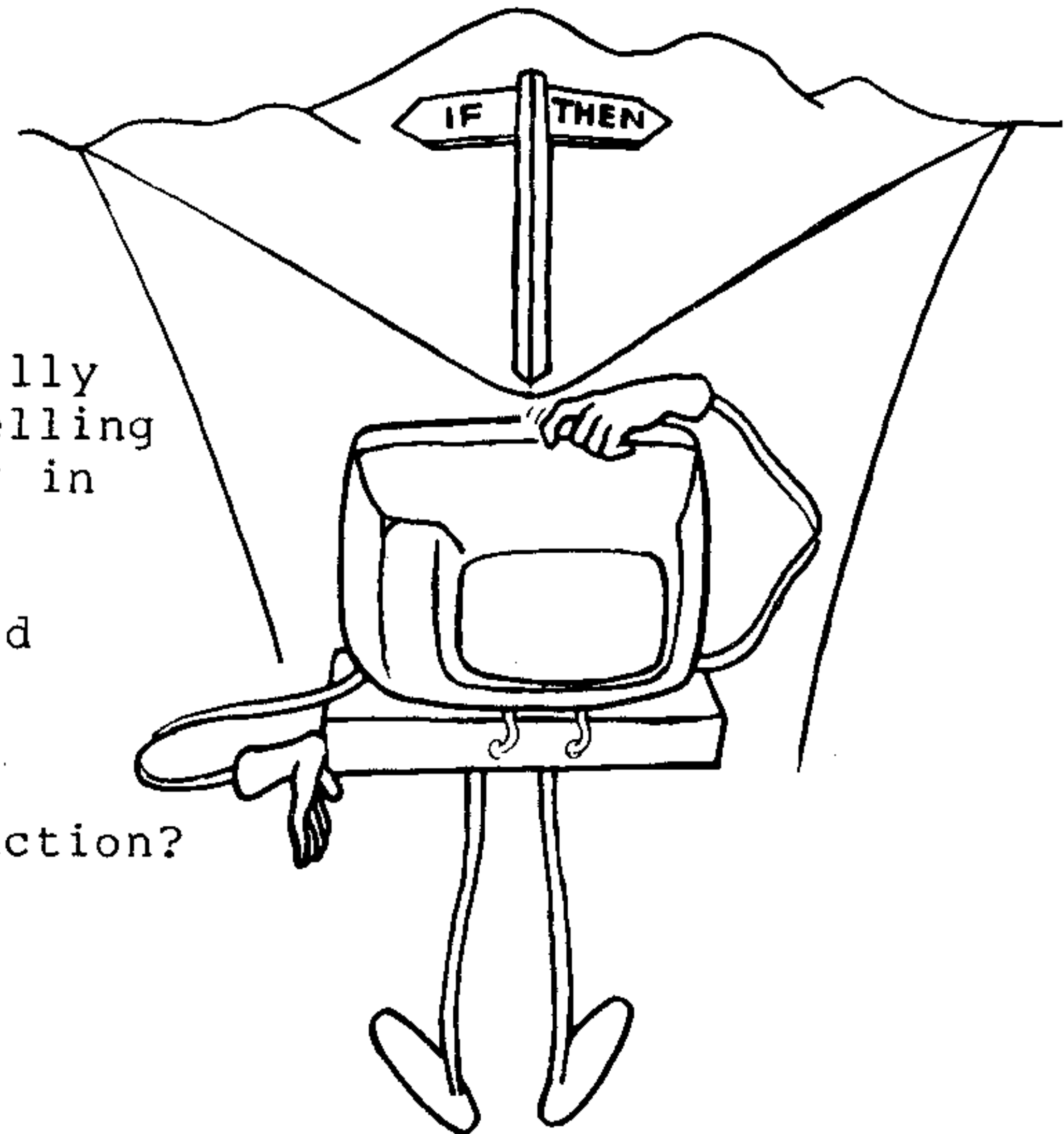
That was the effect of our new IF ... THEN instruction. While A was less than 13, it let Wizzard keep on looping back. But as soon as line 40 cranked the value of A up to 13, the second part of line 45 suddenly sprang into action and forced Wizzard to stop.

IF ... THEN is quite a powerful instruction. Because it lets us tell Wizzard how to make decisions, and do different things depending upon what it decides, it opens up all sorts of new possibilities. We'll see more of these shortly.

Here's a quick quiz to do before we go on to write our own guessing games in chapter 8.

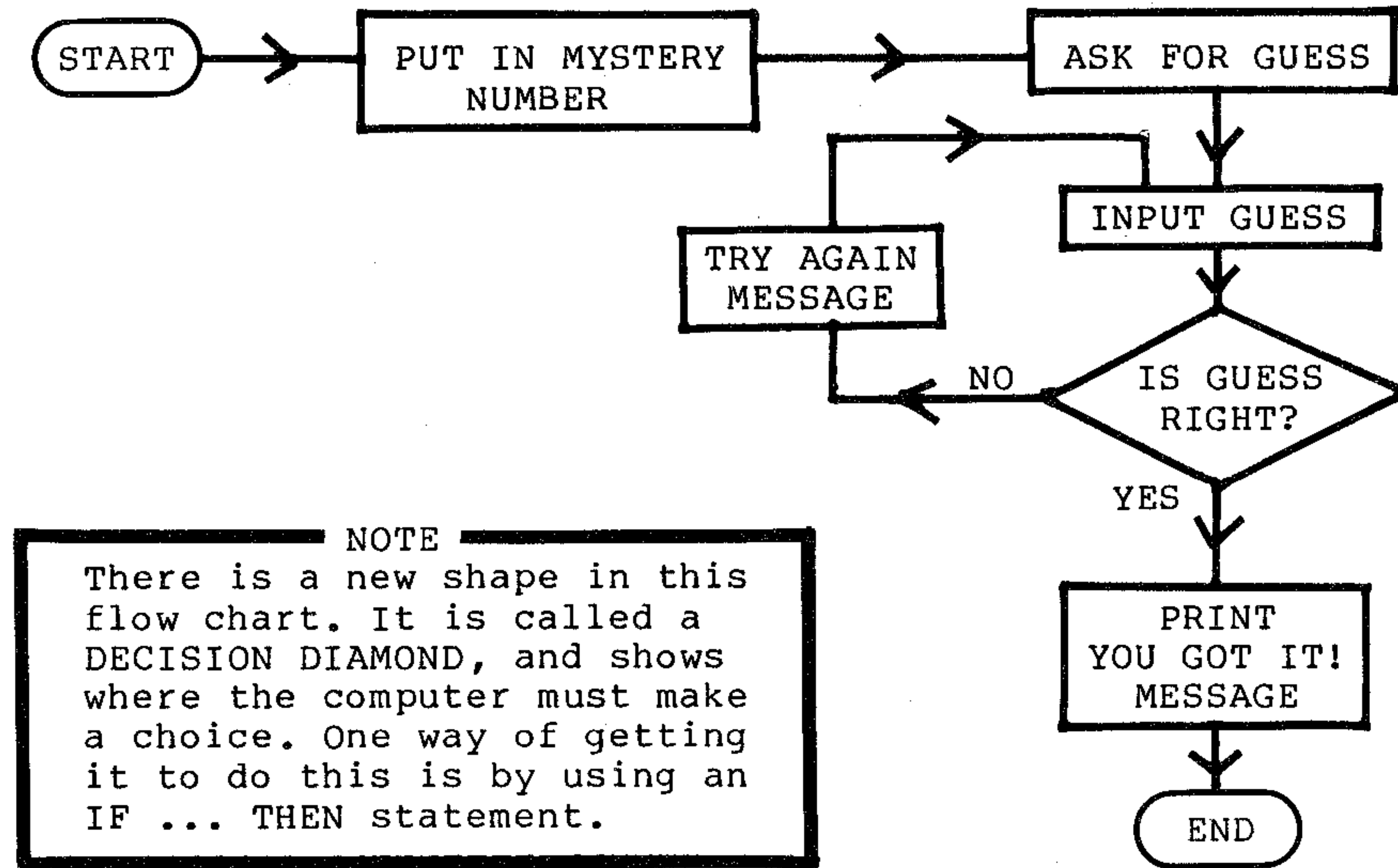
#### QUICK QUIZ 6

1. How can  $A = A + 1$ ? (pick one)
  - a. Because A is stretchy
  - b. Because programming is silly
  - c. Because you are really telling Wizzard to change the number in A to the next number up.
2. Write a program to get Wizzard to do its 13 times table, going up to 13 X 15 and then stopping.
3. What is an IF ... THEN instruction?



## CHAPTER 8 - Guessing games

Here's a flowchart for getting people to guess a mystery number:



In this case, if the number guessed is right, Wizzard will say YOU GOT IT! and stop. If it is wrong, Wizzard tells you to TRY AGAIN and goes back to let you guess again.

Let's write a program from this flowchart. If the mystery number is 7, it will go something like this:

```
10 A=7
20 PRINT "FIND MY NUMBER!"
30 INPUT B
40 IF B=A THEN GOTO 70
50 PRINT "NO - TRY AGAIN!"
60 GOTO 30
70 PRINT "YOU GOT IT!"
80 END
```

Notice we had 2 variables in this program. One was A - for the mystery number, the other was B - for the number you guess.

There is just one problem we have to solve before you can go and get someone to play this game. When you've typed it in, and probably LISTed it for checking, the program will still be showing on the TV screen (complete with the mystery number!).



So it is time to introduce another new instruction, one which tells Wizzard to "wipe its screen clean". This will hide the program from the guesser.

NOTE

The instruction CLS (short for Clear the Screen) is used to tell Wizzard to rub everything from its video screen.

You can use CLS as either a direct command, by typing it in without a line number, or as a program line. For the moment, let's make it into an extra line at the start of our program. It will look like this:

5 CLS



OK, key in the complete program now, and don't forget line 5. Then LIST it and RUN it...

How did the game go? It's a bit hard for your guesser to find the number isn't it, because there are lots of numbers to choose from.

Maybe we can get Wizzard to be a bit more helpful. A good addition to our program would be to get Wizzard to tell the guesser if the number they've picked is too big or too small.

Greater than and less than

We do this by using another IF ... THEN statement, with a greater than (>) sign.

Let's replace line 50 and add a few more:

```
50 IF B>A THEN GOTO 56
52 PRINT "TOO LOW!"
54 GOTO 30
56 PRINT "TOO HIGH!"
```

(Now that our programs are getting a bit longer, you can see how useful it is to leave spaces for adding new lines.)



OK, press CNT'L and C to stop the program (unless your guesser got the number), then type in the extra lines. Now LIST the program to check it.

Your program should now look like this:

```
5 CLS
10 A=7
20 PRINT "FIND MY NUMBER!"
30 INPUT B
40 IF B=A THEN GOTO 70
50 IF B>A THEN GOTO 56
52 PRINT "TOO LOW!"
54 GOTO 30
56 PRINT "TOO HIGH!"
60 GOTO 30
70 PRINT "YOU GOT IT!"
80 END
```

Can you see what we've done? If B, the number guessed, is equal to A, then line 40 will send Wizzard down to line 70 as before. But if the two aren't equal, it will drop down to line 50 instead.

Line 50 then looks to see if B is larger than A or not. If it is, then Wizzard will be sent down to line 56. So it will print out the TOO HIGH! message, and then loop back (line 60) to let you try again.

What happens in line 50 if B is not greater than A? Well, it can't be equal to A, or Wizzard wouldn't have got to line 50 in the first place. And if B is not greater than A, there's only one possibility left -- it must be smaller than A!

So that's why our line 52 tells Wizzard to print out the TOO LOW! message, and then loop back (line 54) as before to let you try again.



RUN the modified program now, to see how it works.

It's a lot more helpful now, isn't it?

By the way, instead of using the greater-than sign in our second IF ... THEN instruction (line 50), we could have used a less-than sign (<). But we would have had to swap the instructions in lines 52 and 56, to match the opposite way line 50 would have worked.



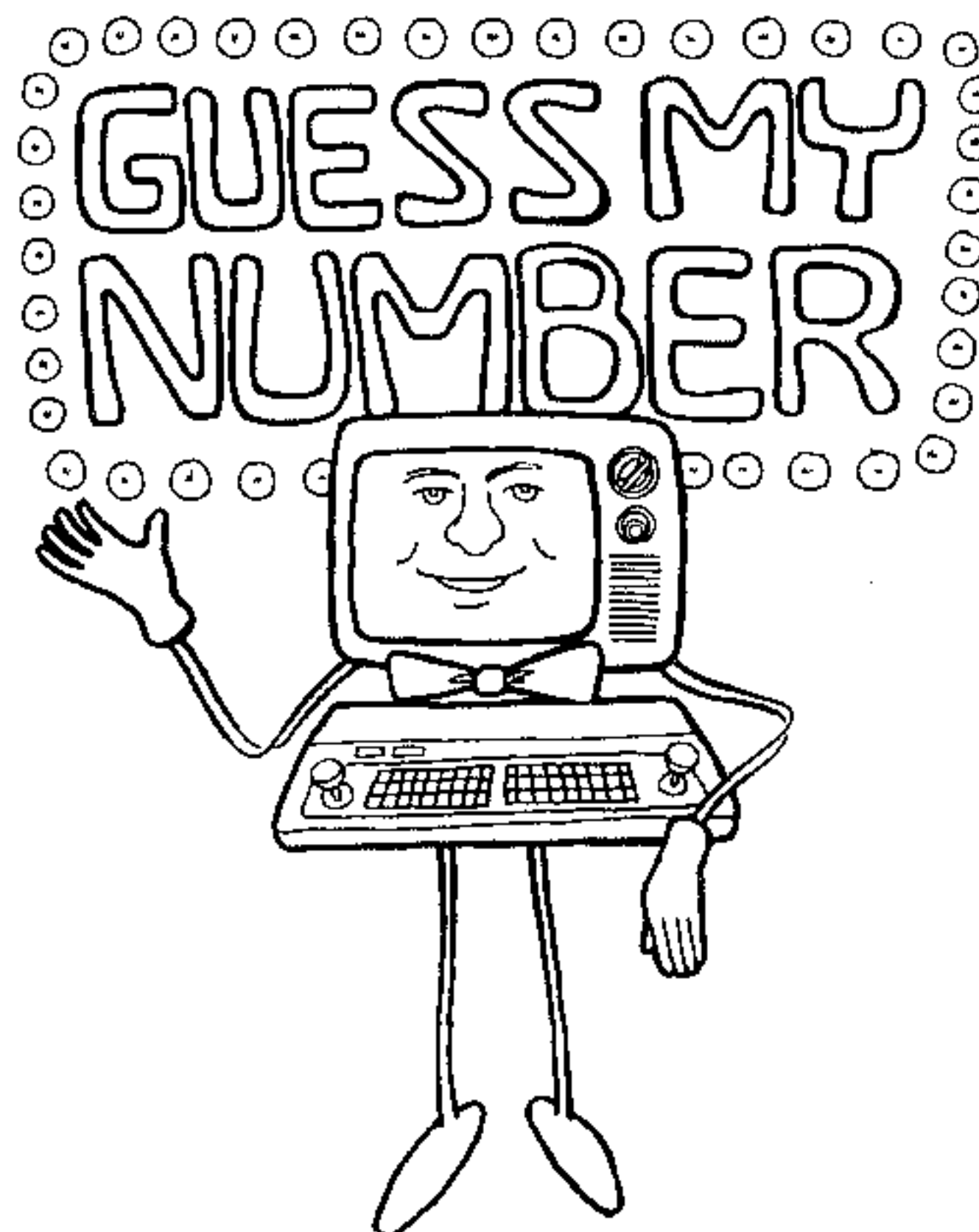
You might like to try this for yourself. Change the three lines and then RUN the program again to prove that it still works.

The IF ... THEN instruction is pretty flexible, isn't it? By using signs like equals (=), greater-than (>) and less-than (<), we can make it check for all sorts of things.

Here's another quick quiz before we go on.

#### QUICK QUIZ 7

1. What does Wizzard do when you type CLS? (pick one)
  - a. Give you a clue?
  - b. Clear everything off the TV screen?
  - c. Put you at the top of the class?
2. What does B>A mean?
3. What does B<A mean?
4. What do you put on a flow chart to show an IF ... THEN?



## CHAPTER 9 - Making Wizzard do more of the work

That last guessing game may have been quite fun for someone else to play, but you couldn't play it yourself, could you? Because you already knew the number.

Now we're going to show you a way to make Wizzard "think of its own number", so that you can play the game too.

The program you use will be exactly the same as the one you used in the last chapter, except for line 10. Instead of saying `A = 7` (or any other number you choose), we are going to use line 10 to get Wizzard to pick a number you don't know yourself.

### Random numbers

We can get Wizzard to pick a number "at random" (in other words, pick any number it likes) by using another new instruction: the RND instruction. It looks like this:

```
10 A=RND (100)
```

This gets Wizzard to pick any number between 1 and 100, and store it away as A. If you wanted a wider choice (such as between one and 500), the line will have a bigger number in the brackets, such as:

```
10 A=RND (500)
```



OK, type in your program with this new line, then LIST it and RUN it again

See what happens -- each time you run the program, Wizzard will pick a new number for you to try guessing. So now you'll be able to play the game just like anybody else!

### Doing things a fixed number of times

Remember in chapter 7 where we wrote a little program to get Wizzard to work out a number table and stop at the end? We used an IF...THEN instruction to work out when to stop looping, when our variable A had become larger than 12.

Well that certainly worked, but now we'd like to show you a rather neater way of looping around a fixed number of times.



This is by using a pair of new instructions, FOR and NEXT.

**NOTE**

The FOR instruction goes at the start of your program loop, and the NEXT instruction goes at the end.

The FOR instruction tells Wizzard that you want it to make the following instructions part of a loop, and that a particular variable is to be stepped up or down in size each time it goes around the loop, until it reaches a certain value.

The NEXT instruction tells Wizzard where the FOR instruction's loop ends.

**NOTE**

For every FOR instruction in your program, there must be a matching NEXT instruction after it...

Here's how the FOR and NEXT instructions would be used to neaten up our number table program:

```
10 FOR A=1 TO 12
20 PRINT A;" X 4 = ";
30 PRINT A*4
40 NEXT A
50 END
```

The FOR instruction in line 10 tells Wizzard that we want to have it make a loop using the instructions that follow, and that we want it to loop around 12 times with variable A=1 the first time, A=2 the second, and so on up to A=12.

The NEXT A instruction in line 40 tells Wizzard that the FOR loop ends at that line. Wizzard won't get to line 50 until it has looped back to line 10 the required number of times.



Try running this version of the program for yourself, to see how it works...

You can have any letter from A to Z for the "how many times around the loop" variable used in a FOR instruction. We only used A here because it was used before. Whatever letter you use, it's known as the loop counter variable.

The FOR instruction is even more flexible than you might think from this example, because if you want, it will let you step the loop counter variable down instead of up each time it goes around the loop. Not only that, but it will let you step the counter variable up or down not just by 1, but by a larger number if you wish.

You get it to do these things quite simply, by adding another part to the instruction -- a part showing the STEP size.

For example, say you wanted our number table program to go backwards, from 12 down, and you wanted it to work out the answer only for the even values of A. You could do this by changing line 10 to read:

```
10 FOR A=12 TO 2 STEP -2
```

This tells Wizzard that you still want to make it do the following instructions as a loop, with A as the counter variable, but that this time you want it to decrease A by 2 each time it goes around, starting with A=12 and looping until A=2.



Why not try changing line 10 to this, and RUN it again to see the difference

The FOR and NEXT instruction pair are pretty flexible, aren't they? They give you a neat way of getting Wizzard to do loops a fixed number of times, even with the basic version of the FOR instruction. If you add the STEP part, which is optional, they let you do even fancier tricks.

Before we leave FOR and NEXT, you've probably been wondering when you should use these instructions for a loop, and when you should use an IF ... THEN instruction with a GOTO. That right?

**NOTE**

As a general guide, FOR and NEXT are best if you know in advance exactly how many times Wizzard will have loop around. But IF...THEN with a GOTO is better where you want it to loop until something reaches a certain size.

But there's no rigid rule on using these instructions, so you can try both ways for a particular program, and see which you prefer. It's largely a matter of individual preference.

## Adding REMarks to your programs

As your programs get more and more complicated, it is a good idea to put headings and comments in them. Not for Wizzard's sake, but so that YOU can come back later and still see what the program is supposed to do.

This is easy to do. Just write a line with REM in the front, straight after the line number, and Wizzard will know it is a line for you to read, not an instruction.

### NOTE

Wizzard will automatically ignore any line which has REM straight after the line number. It simply skips that line and jumps to the next one.

For example in our number guessing program, you might want to put in a line like this:

```
35 REM PRESS RET'N AFTER INPUT
```

This line reminds you that after typing in your guess number, you must press RET'N or Wizzard won't know that you have finished.

## WHAT YOU HAVE LEARNT IN CHAPTERS 7, 8 AND 9

This is the new information you have learnt so far. Just run through them to see if you can remember what each instruction means, before you go on. (There's no Quick Quiz this time)

Here are the things we've covered:

```
IF ... THEN and decision diamonds
CLS      to clear the TV screen
>        (greater-than)
<        (less-than)
RND      to generate random numbers
FOR with STEP (optional)
NEXT
REM      for adding remarks to your programs
```



## CHAPTER 10 -- Saving programs on cassette

There has to be an easier way than typing in a program every time you want Wizzard to run it. And there is.

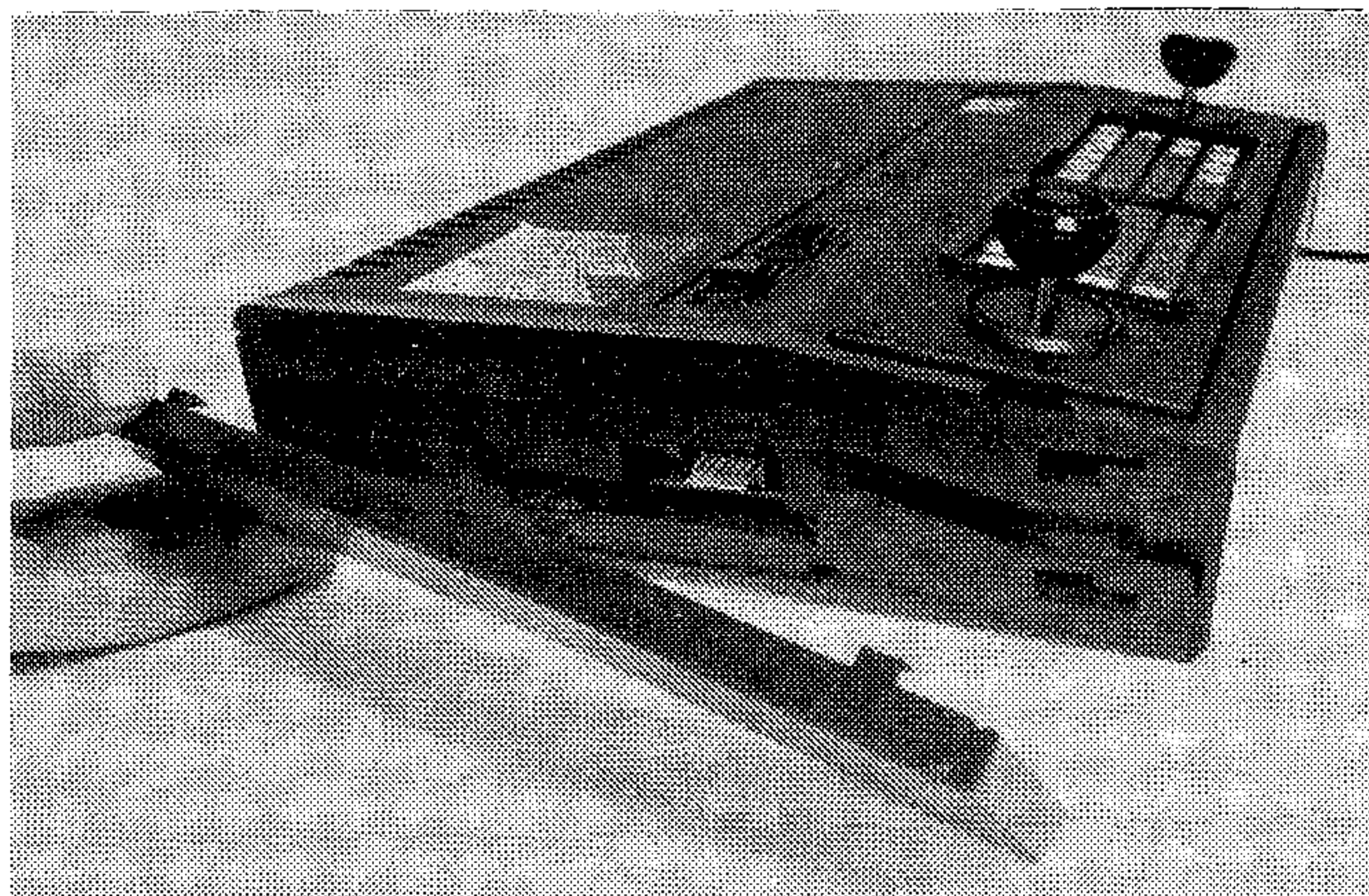
The easier way is to save your programs on cassette tape. Then all you have to do each time to want to run one is load it back into Wizzard, straight from the tape. This only takes a minute or two -- much faster than typing it in again.

To do this you will need the optional Cassette Deck/Interface for your Wizzard (Cat. No. Y-1607), and of course a cassette tape. You don't need special tapes -- an ordinary audio cassette tape will do, although we do recommend that you use a "low noise" tape of reasonable quality.

Although normal C-60 or C-90 size cassettes are OK, you'll find it easier to get short tapes (eg C-10 or C-15) and record one program on each side. This will save you the hassle of having to search through a long tape, in order to find a particular program you want.

Dick Smith stores sell C-10 cassettes that are "computer verified" to be free from faults (Cat. No. X-3502). You can also get ordinary C-10 and C-15 cassettes elsewhere (of course, they won't be as nice, but that's your business).

The Cassette Deck/Interface attaches to your Wizzard on the left-hand end, looking from the front. You clip off the Wizzard's end plate, and then connect the little 6-hole plug on the end of the Cassette unit's cable to the 6-pin socket inside the Wizzard. You can see the socket in this picture:





Here's how you connect the two together:



1. Switch Wizzard off and turn it over.
2. Locate the small plastic spigot on the bottom, near the left hand end, and turn it around until its screw-driver slot is at right angles to the end. Then you can pull it out.
3. Remove the left-hand end plate by pushing it towards the back about 10mm. Then pull it away to the left.
3. Plug the cable from the Cassette adapter into the Wizzard's 6-pin socket.
4. Sit the Cassette adapter alongside the Wizzard and clip it on instead of the original end plate. Then re-fit the plastic locking spigot.

Now your cassette adapter is set up ready to SAVE programs from Wizzard onto tape, and LOAD them back into Wizzard. But before you try to do either, there are a few general things you should know about Wizzard and cassettes:

**NOTE**

Before you try to LOAD or SAVE a program, make sure you rewind the cassette back to the start.

Some cassettes come with a plastic "leader" on the start of the tape. This is usually transparent, and a different colour from the usual dark brown of the tape itself. Unlike the tape, this leader won't take any recording, so you have to make sure that Wizzard doesn't try to record on it.

The best way to do this is to remove the rewound cassette from the deck and use a pencil in the "take up" spindle hole to wind it forward until the tape is showing in the front windows.

To SAVE programs on tape

Once you've got a rewound cassette in the deck, with the tape ready to take a recording, it's really very easy to SAVE a program that you've got in Wizzard's memory.



1. LIST your program to make sure it is what you want.
2. Press both the REC and PLAY keys down together.
3. Type in CSAVE and hit RET'N.

When the program is all saved, the cassette will stop turning and Wizzard will show you the > sign and blink its little cursor square again.

It's a good idea to remove the cassette and label the side facing up in the machine, to remind you which program you have saved there. Then put the cassette back in its cover to protect the tape and its magnetic recording.

#### To LOAD a program back into Wizzard from tape

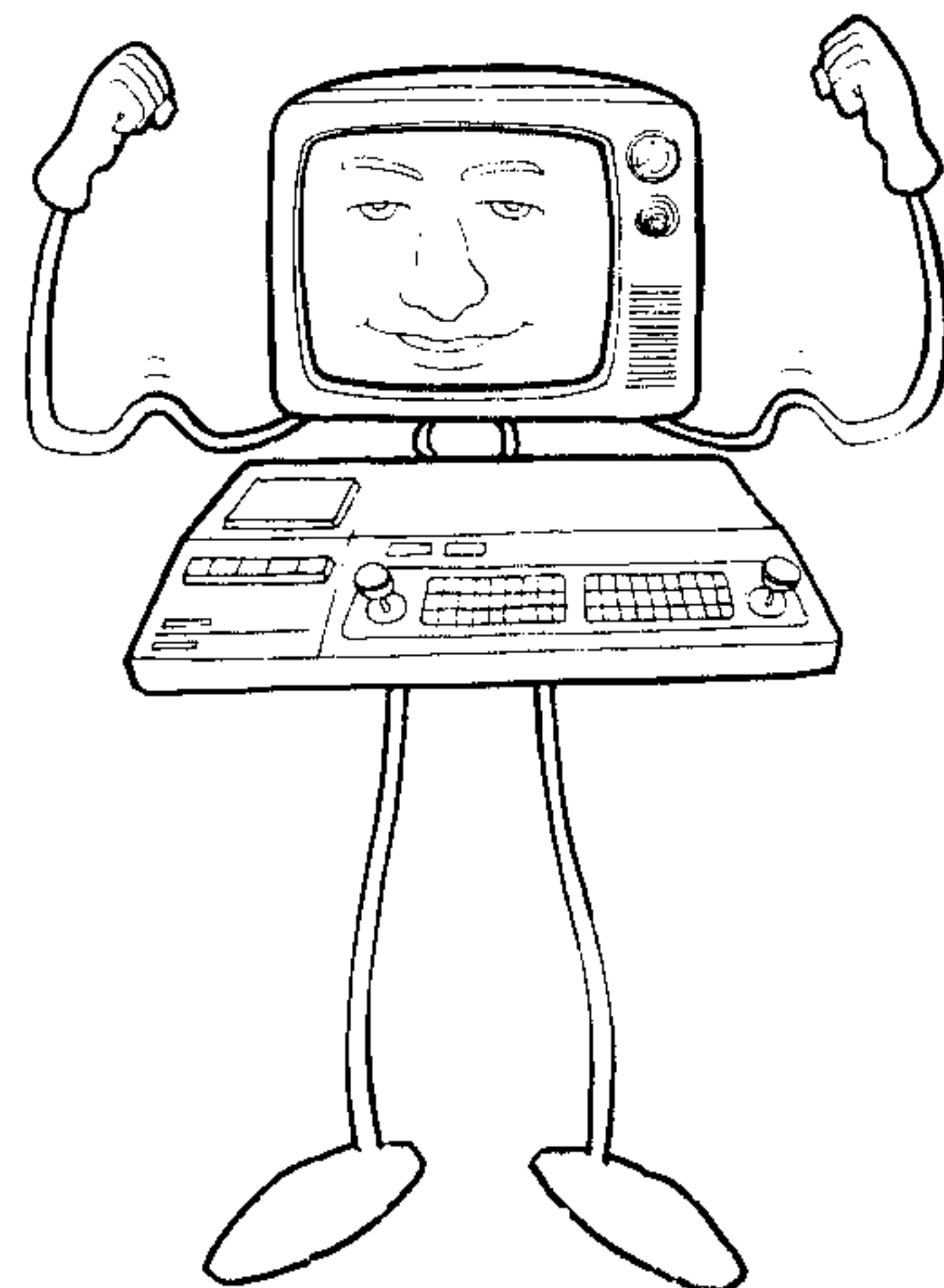
The process is equally simple when you want to LOAD the program back into Wizzard's memory so you can use it again:



1. Put the cassette in the deck with the side you want facing up.
2. Close the lid and rewind to the start of the tape.
3. Type in CLOAD and hit RET'N.
4. Press the deck's PLAY button.

When the program has been loaded into Wizzard's memory, the tape will stop. You can check that it has been properly loaded in, by getting Wizzard to LIST it for you.

Using the Cassette Deck/Interface can save you a tremendous amount of time and effort in loading and saving your Wizzard programs. It really lets you get the most out of your Wizzard as a serious personal computer.



## CHAPTER 11 -- Subroutines

As your programs get longer and more complicated, you'll quite often find that you want Wizzard to do the same sort of thing at different points in the program. So repetition starts to creep in: the same instruction line, or a group of lines, crops up a number of times in the program.

This is pretty wasteful, isn't it? You'd think there would be some way that you could put in the line or lines just once, and have Wizzard use it or them whenever it needed to.

Well, there is a way. It's called making the part of the program that you need to use over and over into a subroutine.

### NOTE

A subroutine is a part of your main program which is separate from the rest, and "called" or jumped to from various places in the main program whenever it is needed.

Let's look at an example. Here is a little program which can be used to keep track of how much money you've got in your cheque account:

```
10 CLS
20 PRINT "YOUR BALANCE";
30 INPUT B
40 PRINT "CHEQUE (1)"
50 PRINT "OR DEPOSIT (2)";
60 INPUT C
70 IF C=1 THEN 100
80 IF C=2 THEN 200
90 GOTO 40
100 PRINT "HOW MUCH";
110 INPUT A
120 B=B-A
130 GOTO 300
200 PRINT "HOW MUCH";
210 INPUT A
220 B=B+A
300 PRINT "BALANCE NOW ";B
310 GOTO 40
```

Now if you look at lines 100 and 200, you'll see they are exactly the same. So are lines 110 and 210.

To remove this wasteful duplication, we can take out these lines and have just one pair of them, separate from the main program. We do this by giving them line numbers much larger than those in the main program -- like 900 and 910, for example:

```
900 PRINT "HOW MUCH";  
910 INPUT A
```

Now how do we get Wizzard to jump to these lines from the main program, when it should? And even more importantly, how do we get it to jump back to the right place in the program, when it has carried out these instructions?

As you've probably guessed by now, we use two new instructions: GOSUB and RETURN.

**NOTE**

GOSUB tells Wizzard to jump to a subroutine from the main program. RETURN tells it to jump back to the main program at the end of the subroutine.

So in place of our old lines 100 and 110, all we need now is the line:

```
100 GOSUB 900
```

And in place of our old lines 200 and 210 we need:

```
200 GOSUB 900
```

And to make sure that Wizzard will go back to the main program properly each time it goes to the subroutine, we need this line at the end of the subroutine itself:

```
920 RETURN
```



Why not type in this program now, with these changes. Then RUN it, to prove to yourself that it works.

It works, doesn't it? But most likely at this stage you're still not quite sure how it works.



Perhaps you're wondering why we couldn't use an ordinary GOTO instruction to get to the subroutine, instead of the GOSUB.

The answer is that GOSUB is really a special sort of GOTO, which not only gets Wizzard to jump to a different place, but also gets it to remember where it's jumping from. So when it gets to the RETURN instruction at the end of the subroutine, it will know where to return to in the main program.

If we jumped to the subroutine with an ordinary GOTO, it wouldn't be able to RETURN, because it wouldn't have told it to remember where it came from. Get the idea?

The important thing to realise is that a subroutine is a part of your program that Wizzard jumps to any number of times, from different parts of the main program. But each time it finishes doing the subroutine, it RETURNS to the right place in the main program.

What is the right place? Why, the line after the one with the GOSUB that sent it to the subroutine.

Here's a simple little program which should make this clear:

```
5 CLS
10 PRINT "LINE 1"
20 B=2
30 GOSUB 100
40 PRINT "LINE 3"
50 B=4
60 GOSUB 100
70 PRINT "LINE 5"
80 END
100 PRINT "LINE ";B
110 RETURN
```



Try typing in this program  
and RUN it.

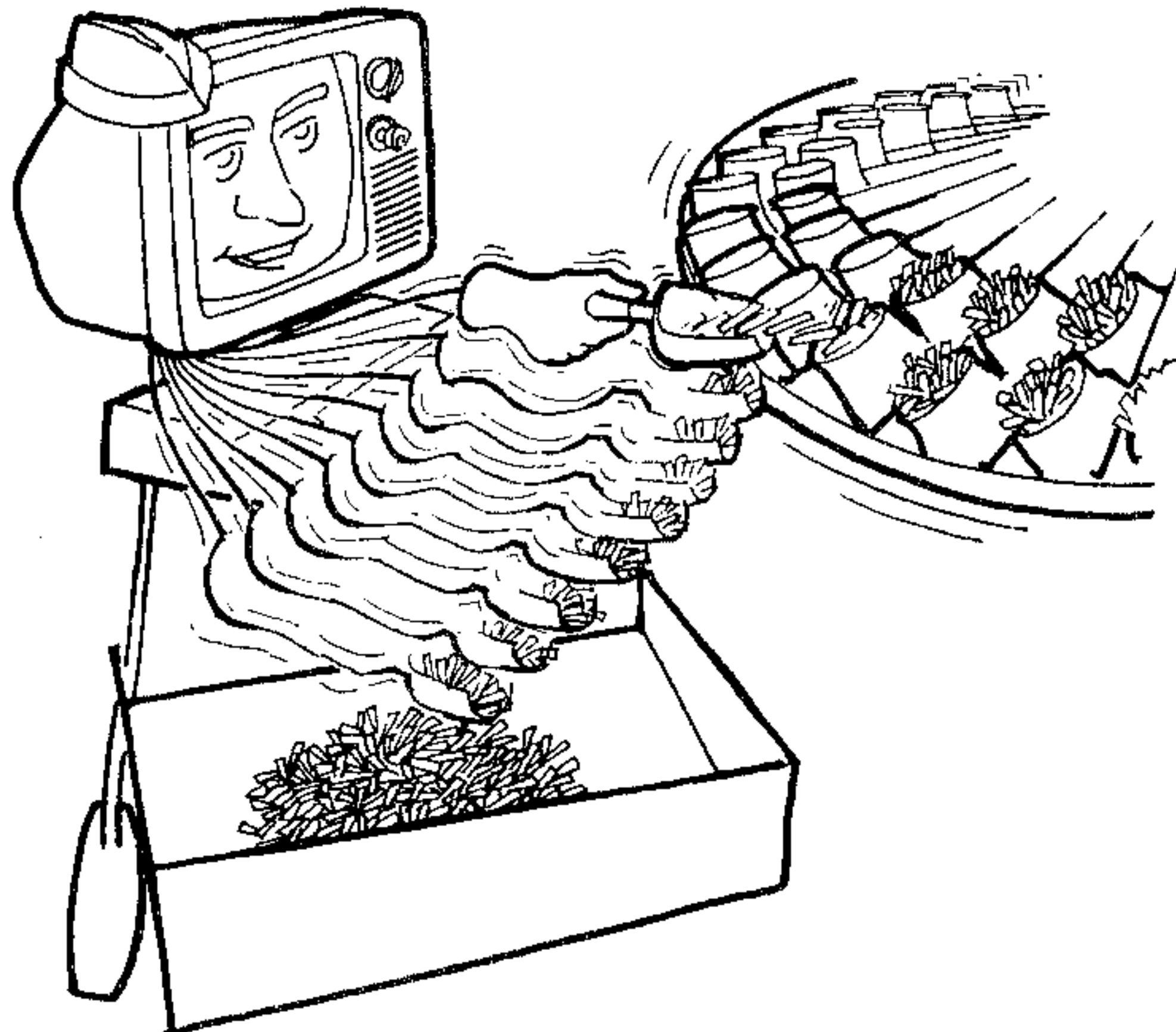
See what happened? Wizzard printed out 5 lines, with the first, third and fifth lines done by the main program, and the second and fourth lines by the subroutine (lines 100 and 110).

Using GOSUB and RETURN, you can use subroutines to make your programs really neat and tidy.

Now do this quick quiz before you go to the next chapter.

## QUICK QUIZ 8

1. What is a subroutine? (Pick one)
  - (a) A program to control a submarine.
  - (b) A part of your program that is used a number of times, at different points in the main program.
  - (c) Part of your program that isn't quite as good as the rest.
2. When is it a good idea to use a subroutine? (Pick one)
  - (a) When you want to control a submarine.
  - (b) When you're not very good at programming.
  - (c) When you find the same instruction line (or lines) cropping up at different places in your program.
3. Why can't you jump to a subroutine using a GOTO instruction?
4. How many times can you jump to a subroutine from your main program?
5. Which instruction must be used to end a subroutine? (Pick one)
  - (a) END
  - (b) STOP
  - (c) RETURN
  - (d) GO BACK



Whenever something has to be done over and over again...

## CHAPTER 12 -- Storing data in your programs

Sometimes you'll want to store some data in a program, so that Wizzard will be able to use it while the program is running -- without asking you to type it in from the keyboard.

This is done using another new pair of instructions: DATA and READ.

### NOTE

The DATA instruction gives you a place to store data (numbers, words and so on) in a program. The READ instruction tells Wizzard to get the data from your DATA line or lines.

Let's look at an example which shows how DATA and READ are used. Say we want a program which stores a list of numbers, and checks any number you type in to see if it is one of those it has on the list. The numbers on the list might be customers who haven't paid their bill, for example, or they might be the numbers of stolen credit cards. Here is a simple program which will do this:

```
10 CLS
20 DATA 132,547,798,405,634
30 PRINT "CUSTOMER NUMBER";
40 INPUT N
50 C=1
60 READ D
70 IF D=N THEN GOTO 200
80 C=C+1
90 IF C<6 THEN GOTO 60
100 PRINT "OK - NOT LISTED"
110 END
200 PRINT "BAD - ON LIST!"
210 END
```

The list of numbers is stored in line 20, as you can see. The word DATA at the beginning of the line tells Wizzard that the rest of the line contains data, which will be used later on. The commas between the numbers are simply to separate them, so that Wizzard can tell them apart.

In this case there are only five numbers, so they all fit in a single DATA line. If we had more, they would probably have to be put into a second DATA line.

Did you notice that the DATA line comes right near the start of the program, well before the READ instruction in line 60? This is important!

NOTE

The DATA line or lines must be before the READ line in your program, or Wizzard won't be able to read the data properly.

Can you see how the program works? Line 30 asks you to type in the number you want checked, while line 40 brings in the number you type and calls it N.

Line 50 now tells Wizzard to think of a variable C, and give it a value of 1. We'll explain the reason for this in a moment.

Line 60 is our READ instruction, which tells Wizzard to read a data number and call it D. The first time it does line 60, it will read the first number in line 20.

Line 70 now gets Wizzard to check and see if N and D are the same. If they are, then it jumps down to line 200 to print out the "BAD - ON LIST!" message. But if they're not the same, it goes to line 80.

Line 80 tells Wizzard to increase C by one, so since it started as 1 (from line 50), it will now be 2. Then line 90 asks it to check C, to see if it is still less than 6. If it is, Wizzard is sent back to line 60 to read another number from the list.

Line 60 then gets Wizzard to read another number from line 20, which line 70 will again compare with N. If they're the same this time, it will jump down to line 200. But if they're still not the same, lines 80 and 90 will send it back to try a third number on the list. And so on...

NOTE

What we're doing in lines 60, 70, 80 and 90 is getting Wizzard to keep reading the numbers in the list, until it either finds one that is equal to N, or reaches the end of the list.

How does it know when it has reached the end of the list? That's the reason for lines 50, 80 and 90. We're using the variable C to count the number of times we go around our loop.



If we try to go around more than 5 times, line 80 will make C equal to 6, so that line 90 will stop the looping back. Wizzard will drop down to line 100, and print out the "OK - NOT LISTED" message.

Note that the number we compare C with, in line 90, is 6 because this is one bigger than the number of numbers in our list. If we have more numbers in the list, line 90 would have to be changed.



OK, why not type in this program and RUN it a few times, to see how it works. Try typing in a number that you know is on the list, the first time you RUN it, then try typing in a number that you know isn't on the list.

Not bad, is it? Wizzard seems to take no time at all to check the number you type in against those on the list, and gives you the verdict as soon as you've finished typing in.

The only problem at the moment is that you have to RUN the program again, each time you want it to check another number. This is a bit clumsy.

There has to be a neater way, and there is. But we have to add a third new instruction, to tell Wizzard that we want it to go back and start READING from the start of the DATA line again. The instruction to do this is RESTORE.

**NOTE**

RESTORE tells Wizzard to go back to the start of your DATA line(s), before it does a READ instruction.

The place to add the new instruction is between our existing lines 20 and 30, so we can make it line 25:

25 RESTORE

Once we've done this, it's easy to make the program loop around again to let us type in another number for checking, without having to RUN again.

All we have to do is change lines 110 and 210, so that instead of ENDing, they loop Wizzard back to line 25:

```
110 GOTO 25
210 GOTO 25
```



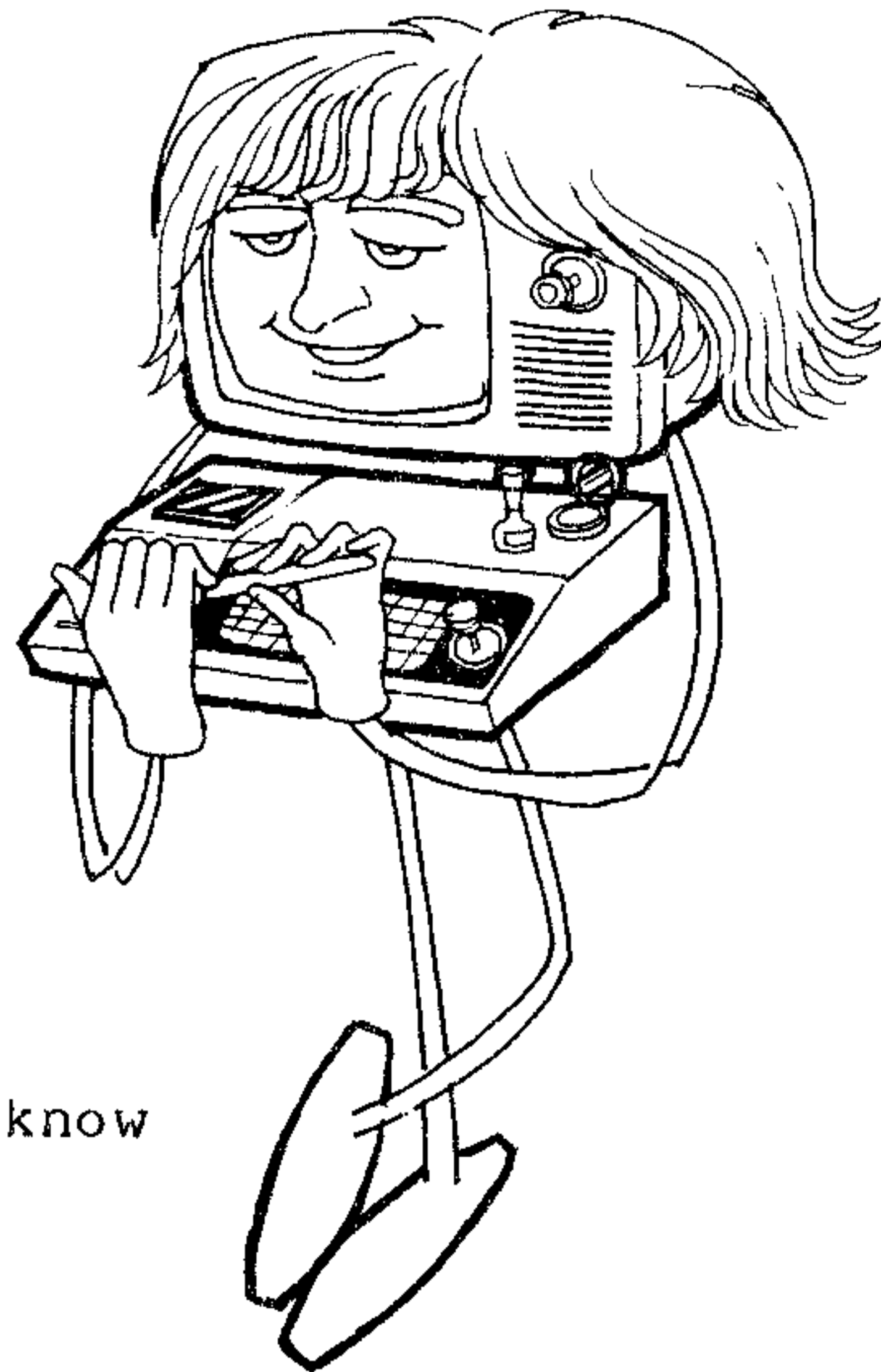
OK, type in these three lines  
and RUN the program again.

Quite a bit neater now, isn't it? The program will keep running as long as you like now, checking any number you feed in against the 5 numbers in its list. In fact the only way to stop it is to press CNT'L and C, or hit the RESET button!

Hopefully you can see from this that DATA, READ and RESTORE are quite powerful instructions. They give Wizzard the ability to store quite a lot of data in its programs, and do all sorts of comparisons and checks with it.

There's no quick quiz for this chapter, because you've had to do quite a bit of thinking already. But you're going to enjoy the next chapter -- we show you how Wizzard can play music!

## INFORMATION



Need to file some information?  
I'm not just a pretty face, y'know

## CHAPTER 13 -- Getting Wizzard to play a tune

One of the really good things about Wizzard is that unlike most other computers, you can get it to play music through your TV set. It has a powerful built-in music generator, which is easy to work from your BASIC programs.

All you need to do to become a virtuoso of the Wizzard keyboard is learn one new instruction. It looks like this:

```
SOUND A;X, B;Y, C;Z
```

The word SOUND is logical enough, isn't it? But you're no doubt wondering about those letters after it.

Notice that the letters are in three groups, separated by commas. This is because Wizzard's music generator has three channels. In other words, it can make three different notes at the same time, if you wish.

So you aren't limited to playing just simple one-note-at-a-time melodies. You can play music with chords as well!

The first letter in each of the three groups above (A, B and C) stands for the musical note that you want Wizzard to play in each of the three channels. In each of these positions you can put a code number, standing for the note you want.

How do you work out the code numbers for the notes? Easy. Just find them using this table:









WIZZARD'S MUSICAL NOTE TABLE	
1 = Rest (silence)	17 = D# (Eb)
2 = C (below middle C)	18 = E
3 = C# (Db)	19 = F
4 = D	20 = F# (Gb)
5 = D# (Eb)	21 = G
6 = E	22 = G# (Ab)
7 = F	23 = A (440)
8 = F# (Gb)	24 = A# (Bb)
9 = G	25 = B
10 = G# (Ab)	26 = C (high C)
11 = A	27 = C# (Db)
12 = A# (Bb)	28 = D
13 = B	29 = D# (Eb)
14 = C (middle C)	30 = E
15 = C# (Db)	31 = F
16 = D	32 = Rest (silence)

As you can see from this, Wizzard can play any note in a full 2-1/2 octave range: from C below middle C (tenor C) up to the F above top C. This is 30 notes, enough to play a tremendous amount of music.

There are also two code numbers (1 and 32) which you can use to make any of the channels in Wizzard's music generator stay silent for a while. This lets you put "rests" in your music.

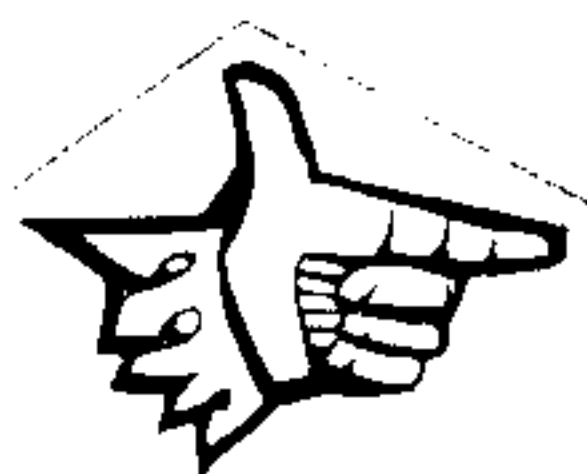
The second letter in each of the three groups (X, Y and Z) stands for numbers which you use to tell Wizzard how long the notes are to last. In other words, whether it should play a full note, a half note, a quarter note and so on.

Just as with the notes, here is a table to show you the code numbers for the different note lengths that Wizzard can play:

NOTE LENGTH TABLE		
0	= a sixteenth note or rest	(  )
1	= an eighth note or rest	(  )
2	= a dotted 1/8 note or rest	(  )
3	= a quarter note or rest	(  )
4	= a dotted 1/4 note or rest	(  )
5	= a half note or rest	(  )
6	= a dotted 1/2 note or rest	(  )
7	= a whole note or rest	(  )

As you can see, you have eight different note lengths, ranging from a sixteenth note to a whole note. This is a very wide range, more than enough for most music. And these can be used for rests (periods of silence) as well as for the actual notes themselves.

OK, let's see if we can play a few notes.



Type in SOUND 14;7,1;7,1;7  
and then press RET'N

You should have heard a single note. If you didn't, it's probably because you hadn't turned up the sound volume on your TV set. So do this, and try again.

The note you heard was middle C (note code 14), played for a whole note (length code 7) through Wizzard's first sound channel. The other two channels were silent because we sent them code 1, with the same length code 7.



To play a 3-note chord, you simply send the right note codes to all three channels at the same time, like:

```
SOUND 14;7,18;7,21;7
```



Try this out by typing it in,  
then pressing RET'N...

Did you recognise the chord? It was the C-major chord C-E-G.

There are a couple of different ways that you can use to get Wizzard to play a complete tune. One way is to have all of the note codes in DATA instructions, and use a READ instruction to feed them into a SOUND instruction. This is quite a good way of doing it if you just want to play a simple melody through one of Wizzard's sound channels -- with the other two channels staying silent.

Here's a little program that shows how it's done:

```
10 REM MYSTERY TUNE
20 DATA 16,1,16,1,18,3
30 DATA 16,3,21,3,20,5
40 DATA 16,1,16,1,18,3
50 DATA 16,3,23,3,21,5
60 DATA 16,1,16,1,28,3
70 DATA 25,3,21,3,20,3
80 DATA 18,3,26,2,26,2
90 DATA 25,5,21,5,23,5
100 DATA 21,7,0,0
200 READ N,D
210 IF N=0 THEN END
220 SOUND N;D, 1;D, 1;D
230 GOTO 200
```



Type this in, then LIST it to  
check that it's right.  
If it is, RUN it to learn the  
identity of our mystery tune!

Can you see how this little program works? The program itself is just lines 200 to 230, after all the DATA lines with the note codes for the tune. All it does is keep reading code numbers (line 200) and playing them (line 220), looping around until it reads out the 0,0 code stored at the end of the tune (line 100). Line 210 then makes it stop.

Notice how we made the READ instruction bring out two numbers at a time from the DATA lines, calling them N (for note) and D (for duration)? We also used D as the note length code for the second and third channels in line 220, so that these channels stayed silent for the same time as the notes.

This way is fine if you just want to play a simple melody in one channel. But it gets a bit messy if you want to go a bit further and play music with chords and different note lengths.

To do this, it's generally easier to use a new SOUND instruction for each new note. Like this:

```
10 REM MYSTERY MUSIC
20 SOUND 16;1,1;3,1;3
30 SOUND 16;1
40 SOUND 18;3,13;6,9;6
50 SOUND 16;3
60 SOUND 21;3
70 SOUND 20;5,16;6,14;6
80 SOUND 16;1
90 SOUND 16;1
100 SOUND 18;3,16;6,14;6
110 SOUND 16;3
120 SOUND 23;3
130 SOUND 21;5,13;6,9;6
140 SOUND 16;1
150 SOUND 16;1
160 SOUND 28;3,13;6,9;6
170 SOUND 25;3
180 SOUND 21;3
190 SOUND 20;3,14;5,9;5
200 SOUND 18;3
210 SOUND 26;2,1;5,1;5
220 SOUND 26;2
230 SOUND 25;5,13;7,9;7
240 SOUND 21;5
250 SOUND 23;5,16;5,14;5
260 SOUND 21;7,13;7,9;7
270 END
```



Why not type this in and RUN it? You'll be surprised how much better it sounds than our last program...

Get the idea? Now you should be able to make your Wizzard play all sorts of impressive music.

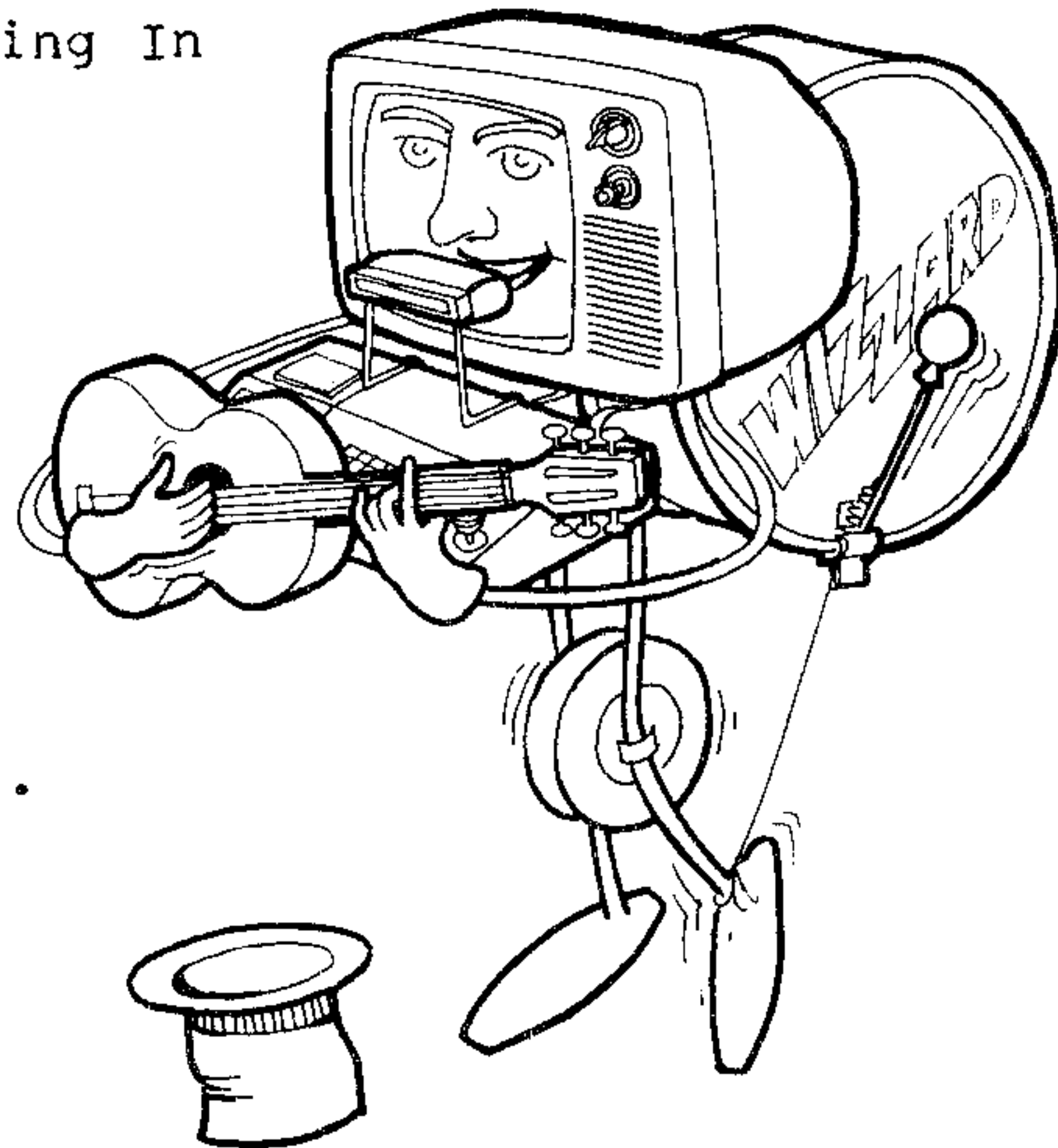
There's just one more thing to note. If you give Wizzard notes of different lengths to play in its three channels, like line 100 in that last program, the next note you give it will go to the channel which had the shortest note. So the notes in lines 110 and 120 will automatically go to channel 1, because it had the shortest note (length code 3) from line 100. The other two channels will still be playing their longer (code 6) notes.

So it's quite easy to have Wizzard playing a melody line with short notes in channel 1, while channels 2 and 3 play long chords.

Finally, here's a quick quiz before we go on.

#### QUICK QUIZ 9

1. How many different notes can Wizzard play? (Pick one)
  - (a) only one -- but a nice one!
  - (b) thirty -- from tenor C to F above high C
  - (c) two -- sweet and sour
  - (d) eight, ranging from very short to very long
3. What is a musical rest? (Pick one)
  - (a) When Wizzard has to stop to catch its breath
  - (b) A stand that holds up your music book
  - (c) A pause in the music, for just the right length of time.
4. What was our mystery tune on page 58? (Pick one)
  - (a) Jingle Bells
  - (b) When the Saints Go Marching In
  - (c) Happy Birthday
  - (d) Amazing Grace



Music? It's in my blood...

## CHAPTER 14 -- Which colour would you like?

Until now, Wizzard has been "talking" to you through your TV screen with black letters and numbers, on a green coloured background. Or if you only have a black and white TV, on a bluish-white background!

If you do have a colour TV set, it's easy to get Wizzard to vary this colour scheme. You can change not only the background colour on the screen, but the colour of the letters and numbers, and even the little squares of screen around them -- all separately.

And since you can vary these colours from your programs, this lets you use Wizzard to produce all sorts of colourful screen patterns and effects!

It's all done with a single new instruction -- the COLOR instruction (makes sense, doesn't it?), which looks like this:

COLOR N, C, B

N stands for a code number which tells Wizzard whether you want to change the colour of the whole screen, or of a particular group of characters (letters, numbers and so on).

C stands for a code number which tells Wizzard which colour you want for the screen or characters. And B stands for an extra code number which you use to tell Wizzard the colour you want for the small square of screen around the characters.

Sound a bit complicated? It's easier than you think.

Let's say you just want to change the colour of the screen background, from green to a rich, deep red. All you need to do this is the instruction:

COLOR 0,7

Here the first number 0 is the code number for the screen background, and 7 is the code number for deep red.

Why not try it?



Type this instruction in now,  
then press RET'N.

Quite a change from the green, isn't it!



You can change the screen background to a lot of other colours, just by using the same instruction with different numbers in the second position. There are 16 different colours, and here are their code numbers:

WIZZARD'S COLOUR CODES	
1 = Transparent	9 = Medium Red
2 = Black	10 = Light Red
3 = Medium Green	11 = Deep Yellow
4 = Light Green	12 = Light Yellow
5 = Dark Blue	13 = Dark Green
6 = Light Blue	14 = Magenta
7 = Deep Red	15 = Grey
8 = Cyan	16 = White

The same colour code numbers are used when you tell Wizzard to change the colour of the characters. But to do this, you also need to know the code numbers for the various characters.

The characters are divided into 16 groups or sets, each with eight characters in it. Each set is given a number, and it is these numbers you use in your COLOR instruction.

How do you work out the character set you want? Well, the characters are grouped into the various sets according to the code numbers that Wizzard uses to represent them inside itself. These are known as the ASCII codes, and they're listed for you in Appendix D at the end of this book.

Here are the character set codes for the 16 groups of characters, showing the ASCII codes in each group:

CHARACTER SET CODES	
5 = ASCII codes 32-39	13 = ASCII codes 96-103
6 = ASCII codes 40-47	14 = ASCII codes 104-111
7 = ASCII codes 48-55	15 = ASCII codes 112-119
8 = ASCII codes 56-63	16 = ASCII codes 120-127
9 = ASCII codes 64-71	17 = ASCII codes 128-135
10 = ASCII codes 72-79	18 = ASCII codes 136-143
11 = ASCII codes 80-87	19 = ASCII codes 144-151
12 = ASCII codes 88-95	20 = ASCII codes 152-159

OK, let's look at an example. Say you want to have the numbers from 0 to 7 printed out on the screen in dark blue, on little white squares, instead of black on green. To do this, you would give Wizzard the instruction:

COLOR 7,5,16

Here 7 is the character set code, which you find by first getting the ASCII codes for the characters you want (from the list in appendix D), and then looking for these codes in the table we've just given you.

The other two numbers are the codes for the colours you want: 5 for dark blue characters, and 16 for the white squares around them.



How about trying this yourself? Type in this instruction, then press RET'N. Then test that it works, by typing in the numbers from 0 to 7...

Getting the idea? If we wanted to change the letters from P to W, so they were printed in deep red on little squares of deep yellow, we would use the instruction:

```
COLOR 11,7,11
```

Here's a little program that gets Wizzard itself to show you the wide variety of colour combinations that are possible:

```
10 CLS
20 FOR A = 1 TO 16
30 COLOR 0,A
40 FOR B = 1 TO 16
50 FOR C = 1 TO 16
60 COLOR 9,B,C
70 COLOR 10,B,C
80 PRINT "HELLO ";A,B,C
90 NEXT C
100 NEXT B
110 NEXT A
120 END
```

Diagram illustrating the nesting of loops:

- outside loop:** Loop A (lines 20 to 110)
- second loop:** Loop B (lines 40 to 100)
- third loop:** Loop C (lines 50 to 90)

What this does is use three FOR-NEXT loops, each one inside the one before it. This is called nesting loops.

The outside loop (lines 30 and 110) uses line 40 to change the screen background through each of the 16 possible colours. Then for each of these colours, the second loop (lines 50 and 100) changes the characters in character sets 9 and 10 through all 16 colours too. And finally, for each colour of these characters, the third loop (lines 60 and 90) changes the little squares around the characters through all 16 colours.

So you get virtually all the possible colour combinations, one after the other. But don't take our word for it!



Try typing it in yourself, and  
RUN it to see how it works...

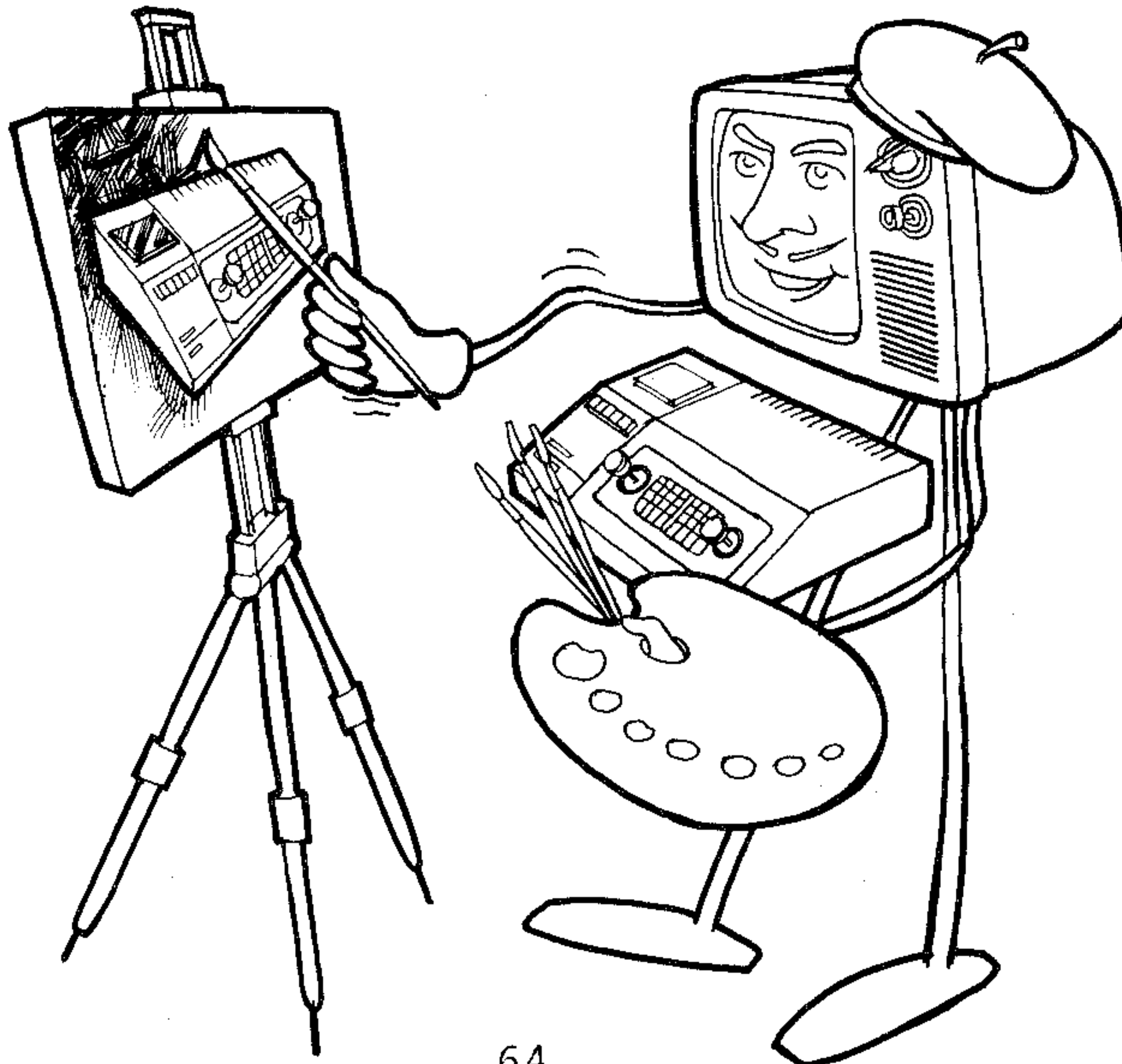
You mightn't see much for a while, because the first two screen background colours are very dark. But things soon brighten up, as you'll see.

Notice that line 80 prints the three loop counter variables, as well as the HELLO message. But because we aren't changing the colour for the number character sets, these stay in black. So you don't even see them for a while!

By the way, this program makes Wizzard go through 16 times 16 times 16 or 4,096 different loops, it takes quite a while to run...

As you can see, Wizzard really gives you a lot of flexibility when it comes to changing the screen colours. When you add to this the ability to draw pictures or "graphics", there's almost nothing you can't make Wizzard put on your TV screen -- except perhaps an episode of "Star Trek"!

We're going to look at graphics next, so stay with us...





## CHAPTER 15 -- Drawing on the screen (graphics)

Up until now, the only things we've been able to get Wizzard to show on your TV screen have been letters and numbers. Wouldn't it be nice if we could make it draw shapes and patterns on the screen as well?

As it happens, we can. With a little ingenuity, you can get Wizzard to draw all sorts of interesting "graphics" on the screen. So let's see how it's done...

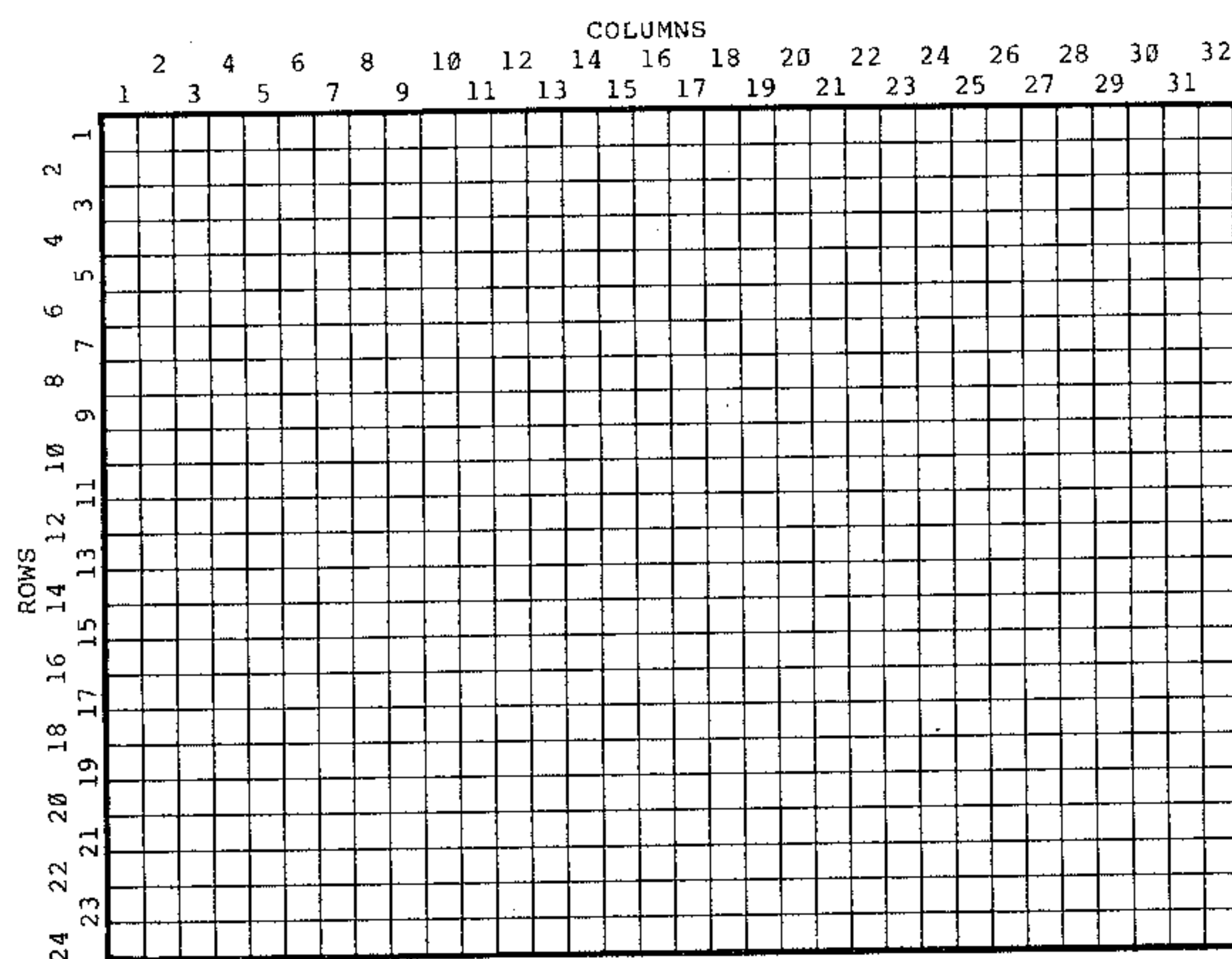
When you tell Wizzard to PRINT something on the screen, it simply puts what you tell it to print on the next available "line" on the screen. This isn't much good if we want to draw shapes or patterns, because it won't let us draw things at particular places -- like the middle of the screen, or the top right-hand corner.

Here's where another new instruction comes to the rescue. It's called PLOT, and it lets us tell Wizzard to put a character anywhere we like on the screen. It looks like this:

PLOT X, Y, C

Here X stands for a number telling Wizzard which column of the screen we want it to put the character in. And Y stands for a number which tells Wizzard which row of that column to put the character in.

Between them, X and Y give you a lot of possibilities. Wizzard's screen is divided into 32 vertical columns, and 24 horizontal rows:





So there are 32 times 24, or 768 different positions on the screen that you can put a character. The only slight complication is that with some TV sets, you may not be able to see all of the positions in the columns at the far left and far right (columns 1, 2, 31 and 32).

This means that depending upon your TV set, you may only be able to use 28 of the columns -- from column 3 to column 30. But this will still give you 672 different positions!

The third letter after the PLOT instruction, C, stands for the ASCII code number of the character you want. So to plot a letter or number you would find its ASCII code number from the table at the back of the book in Appendix D, and use that.

Let's look at an example. Say you want to get Wizzard to put an asterisk or star (\*) character at the centre of the TV screen.

First of all, you work out from the chart on page 64 the column and row numbers for the centre of the screen. These would be 16 (half of 32) and 12 (half of 24). So these would be the numbers for X and Y. Then you would look up the ASCII code number for the asterisk character, from Appendix D. This turns out to be 42. So the instruction to do this would be:

PLOT 16,12,42



Try this out for yourself. Type it in, then press RET'N...

Get the idea? The PLOT instruction lets you put any character you like, anywhere on the TV screen.

But that's not all Wizzard can do in the graphics department. You aren't just limited to the normal letters, numbers and punctuation marks, because Wizzard also lets you teach it your own special characters!

You do this using another new instruction, the CHAR instruction, which looks like this:

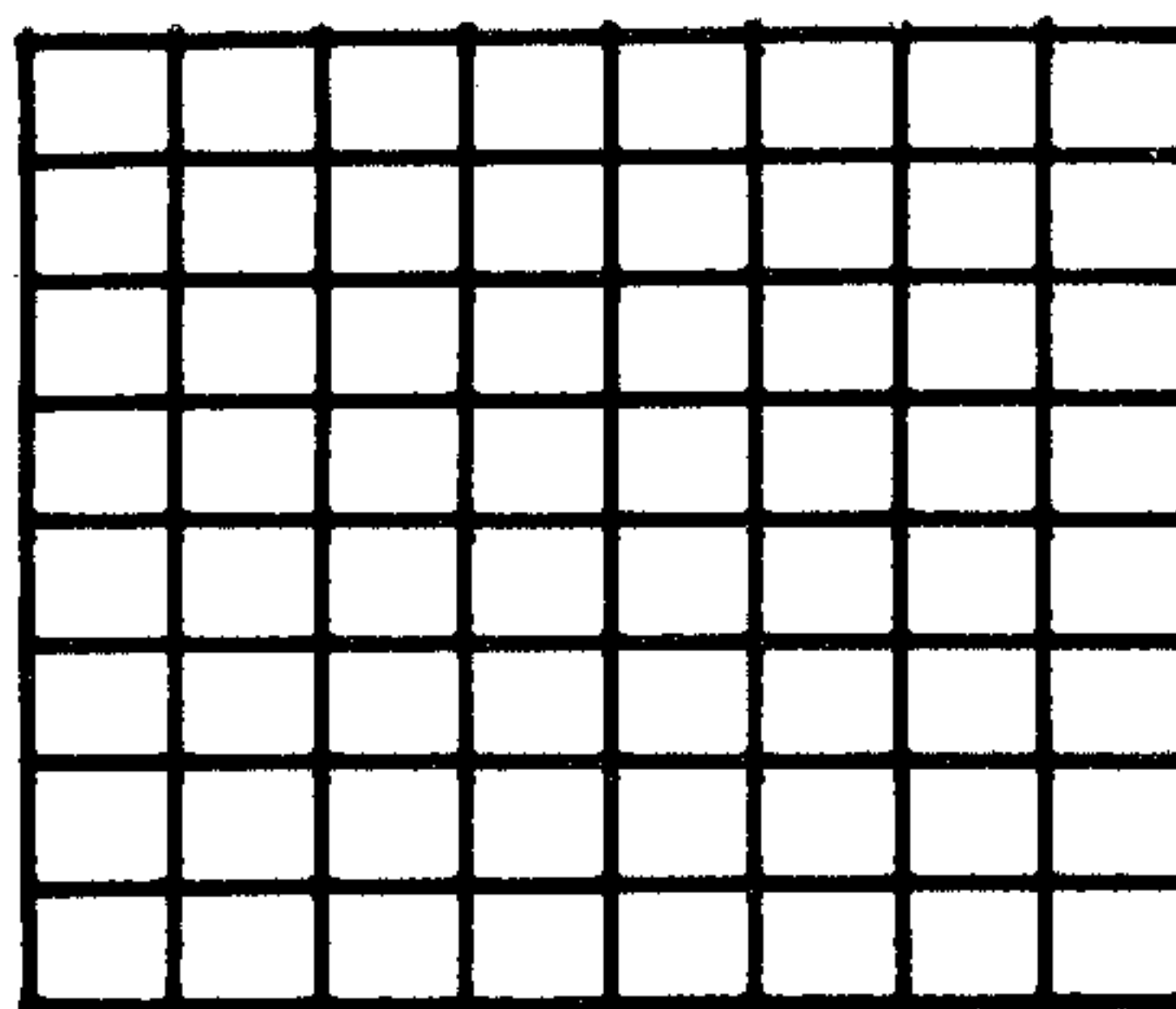
CHAR N, XXXXXXXXXXXXXXXX

Here N stands for the code number you want Wizzard to give your new character. This number can be anything between 0 and 255, and it becomes the number you use as C in the PLOT instruction, to get Wizzard to put your new character on the screen.

The 16 X's after the comma in the CHAR instruction stand for a string of 16 code number digits or letters which you use to give Wizzard the pattern of your new character.

This part is easier than it sounds. If you look carefully at the TV screen, you'll see that all of the characters that Wizzard displays on the screen are made up of little dots. They're just different patterns of dots, in the same 8-dots-wide by 8-dots-high square:

Wizzard thinks of these 64 little dots as being in 16 groups or segments, each of 4 dots. The left half of the top row, then the right half, then the left half of the second row, and so on.



The 16 code number digits or letters in the CHAR instruction are just to tell Wizzard what you want the dot pattern to be in each of these 4-dot segments.

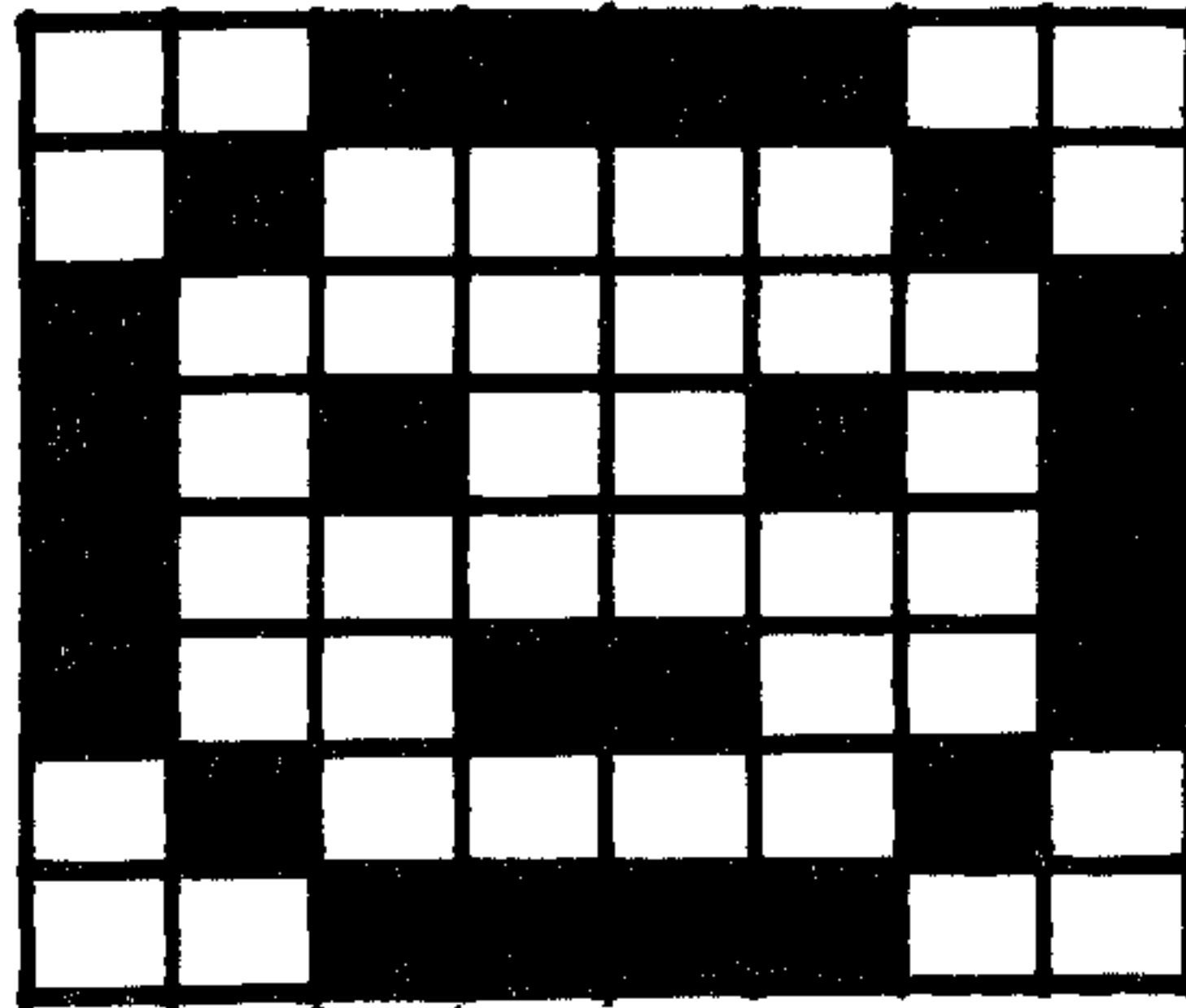
How do you work out the code number or letter for each of the 16 segments of your new character? Easy! From this table:

PATTERN	CODE	PATTERN	CODE
	0		8
	1		9
	2		A
	3		B
	4		C
	5		D
	6		E
	7		F

So to teach Wizzard a new character, you first draw up a grid of 64 little squares. Next you work out the pattern you want for your new character, by filling in as many squares as you need. Then you find the code number or letter for each of the 16 segments, from the table above. Finally you decide on the code number that you want Wizzard to give the new character, and type it all in as a CHAR instruction.

Getting a little confused? Don't worry. Let's look at an example, and it should all become clear.

Say we want to teach Wizzard a new character which is a little face. The first thing to do is draw up our grid of 64 little squares, and fill them in to show the dots that we want to make up the new character:



Right! Now we look up the table on page 67, and work out the code number or letter for each of the 16 segments of 4 dots.

These turn out to be 3, C, 4, 2, 8, 1, 8, 1, 8, 1, 8, 1, 4, 2, 3 and C.



Don't just take our word for this.  
Work them out for yourself, to  
make sure you understand them...

All we need to do now is decide what code number we want to give this little face character -- say code 5. Then it's just a matter of putting all of this into a CHAR instruction:

```
CHAR 5, 3C4281818181423C
```



Try typing this in, and then pressing  
RET'N. Then get Wizzard to show you  
what it looks like, by typing in  
PLOT 16,12,5 and then RET'N again.

Get the idea now? Using the CHAR instruction you can teach Wizzard how to draw all sorts of new characters -- anything that can be made up from 8 rows of 8 dots.

You can teach it up to 255 of these new characters, if you like. There's only one small complication. If you give a new character the same code as one of the normal letters and numbers (as shown in Appendix D), you'll find that Wizzard will replace the normal character with your new character.

This isn't permanent -- the normal character won't go forever. It will come back again if you turn Wizzard off, and then back on again.

All we're really saying is that Wizzard can only remember 225 different character codes at the one time. So if you want all of these for special characters, it has to temporarily "forget" the normal letters and numbers.

Why would you need so many special characters? Well, you might need them if you want to get Wizzard to draw big and complicated things on the screen. To do this, you'd need to teach it lots of "building block" characters, each one making up a part of your bigger pattern.

For example if you wanted to get Wizzard to draw a large face, you'd need to teach it perhaps 4 or 6 characters, which go together to form an eye. Then perhaps another 6 or 8 characters which go together to form a nose, and another 6 or 8 which would form an ear, and so on...

Using the PLOT and CHAR instructions, you can get Wizzard to draw all sorts of things on its screen. Here's a very simple little example you might like to try:

```
10 CLS
20 CHAR 0, 1818FF3C7EFF2466
30 FOR A = 10 TO 14
40 FOR B = 13 TO 20
50 PLOT B, A, 0
60 NEXT B
70 NEXT A
80 END
```



Try typing this in. Then type in RUN, and see what the program puts on the screen...

Now you know how it's done, you should be able to write programs of your own, to draw all sorts of things on the screen.



## CHAPTER 16 -- A bit more about Wizzard's maths

This chapter is really only for the person who wants to write programs so they can get Wizzard to work out the answers to maths problems. If you're not ready for this yet, don't worry about it -- just use what we've shown you so far to have fun!

### Decimals

Wizzard can work with numbers up to 7 figures in length and it lets you put in a decimal point as well.

If you don't put in a decimal point Wizzard will assume you don't want one -- but you can use them.

#### NOTE

Write decimals the usual way and Wizzard can read them up to 7 figures in a number. Wizzard will also give you decimals when you get it to PRINT the answer to a sum.

### Relational operators

Earlier in this book, we mentioned  $>$  (greater than) and  $<$  (less than). These are called relational operators because they are used to describe the RELATIONSHIP between two variables or numbers.

$A > B$ , for example, says that A is greater than B. In other words, it says that in relation to B, A is bigger.

There are other relational operators besides  $>$  and  $<$ , and Wizzard can use them all.

What if you just want to say in an instruction line that A is not the same as B? You can use  $<>$ .

#### NOTE

$A <> B$  means that variable A does NOT equal variable B.

If you want to describe a number that is either the same as another number or less than it, you can use  $<=$ . And if you want to say that a number is the same or bigger than another one you can use  $>=$ .

NOTE

A<=B means that A is less than OR equal to B.  
A>=B means A is greater than OR equal to B.

Squares, cubes etc

Let's say we have a line in a program where we want to square a variable A. We could just write A\*A, so that Wizzard multiplies A by itself. And if we want to cube A, we could write A\*A\*A.

That's all very well for a short things like squares and cubes, but what if we want to get A to the power of 6 (or A X A X A X A X A X A)? It would probably be too long to put in a single instruction line to begin with!

Luckily there is a simpler way, using the double asterisk (\*\*).

NOTE

Wizzard uses \*\* to mean "raise to the power of". So A\*\*3 means A cubed, A\*\*5 means A to the 5th power, and A\*\*2.76 means A to the 2.76th power.

And you can find square roots of numbers using SQR.

NOTE

SQR(A) will get Wizzard to find you the square root of A.

If you want cube roots or more, you can get Wizzard to use logarithms to get them for you.

Logarithms

If you have learnt about logs and anti logs at school, you will already know why you can use them in the way we describe. If you haven't -- this book isn't long enough to explain them, so we will just tell you how to use them.

Wizzard will do natural logs (or logs to the base e) for you using LOG, and antilogs (turning logs back into ordinary numbers) using EXP.

NOTE

To get the log of A, use LOG(A). To get the anti-log of B, use EXP(B).

NOW - to get the cube root of a number, you normally get its log, divide that log by 3 and then get the antilog of the result. If you wanted the fifth root you would divide the log by 5 and get the antilog of the result.

So to get the cube root, of, say, 17, use these commands (using line numbers to fit in with your program of course):

```
10 A=LOG(17)
20 A=A/3
30 PRINT EXP(A)
```

Sine, cos and tan

Wizzard will work out sine cos and tan for you, but it measures angles in RADIANS instead of degrees. To get the right answer you have to divide your angles in degrees by 57.2958. This is not as hard as it sounds -- after all, you have a computer to do the work for you!

NOTE

To get the sine of A, use SIN(A/57.2958).  
To get the cosine of A, use COS(A/57.2958).  
To get the tangent of A, use TAN(A/57.2958).

For example, you can easily make a little program so that Wizzard can save you from looking up your sine tables:

```
5 REM WORKS OUT SINES
10 CLS
20 PRINT"WHAT IS ANGLE";
30 INPUT A
40 A=A/57.2958
50 PRINT"SINE IS";SIN(A)
60 GOTO 20
```

You could have very similar little programs to save you looking up cosine and tangent tables. If you wanted to be really smart, you could write a program to find all three -- or any of them, whichever you ask for!

### INT, ABS and SGN

If you have a number with a decimal point in it, you can use INT to give you just the whole integer part of the number (it chops off anything after the point).

If you want the ABSOLUTE value of a number (without + or - signs) -- use ABS.

If you don't need to know how big a number is, but just whether it is positive or negative (+ or -) -- use SGN.

#### NOTE

SGN(B) gives you the sign of B.

ABS(B) gives you the size, or absolute value of B.

INT(B) gives the integer, or whole number part of B.

Using these maths functions, you can get Wizzard to work out the answers to all sorts of things.



## CHAPTER 17 -- Some final comments

We're now at the end of our first-time journey into the fascinating world of computers and programming. If you're still with us, and we certainly hope you are, you've hopefully learnt quite a bit while working your way through the last 70-odd pages. We hope you've had some fun, too!

One of the things we'd like to say just before we do end up is that although we've shown you quite a bit about programming your Wizzard, it's only been possible to cover the very basic ideas. There's still a good many things that Wizzard can do, that we just haven't had room to explain here.

Things like advanced string handling, data arrays, and PEEK and POKE. Wizzard can do all these things, as you'll see from the summary of its BASIC instructions and commands.

But for explanations of these more complicated aspects of Wizzard's operation, you'll have to read one of the many excellent textbooks around on BASIC programming. Our aim in this book has been to help you get started, and give you a good grounding.

If we've done our job properly, you should now be able to read the more advanced books, and understand them.

But don't be in too much of a hurry. The tricks we've shown you here are much more powerful than you might think. Even if you don't read a more advanced book for a while, they'll let you write programs to get Wizzard to do all sorts of useful and educational things.

So go ahead and experiment. If you make a mistake, all that can happen is that Wizzard will give you an ERROR message -- telling you that it just doesn't understand.

And don't forget to have FUN!

## APPENDIX A -- WHAT THE UNFAMILIAR WORDS MEAN

- ASCII     A code used by computers like Wizzard. Numbers between 0 and 127 are used to stand for each of the letters of the alphabet, number digits and punctuation signs. The letters ASCII stand for "American Standard Code for Information Interchange". Pronounced "ass-key".
- BASIC     The computer programming language used by Wizzard and most other small computers. BASIC is short for "Beginners All-purpose Symbolic Instruction Code".
- CHARACTER     A letter, number digit, punctuation sign or other symbol used to communicate information.
- CONSTANT     Something that stays the same in value. In computers there are generally two types of constants: numeric constants like 2 or 528.57, and string constants like "FRED" or "HELLO".
- COUNTER     A numeric variable whose value is used to count the number of times something happens -- like the number of times the computer loops around in a program.
- DATA     Another word for information. Anything which tells either you or the computer the size of something (like a numeric constant or variable), or a message about something (like a string constant or variable).
- DECISION     A situation where either you, or the computer, must look at some data, and decide what to do or which way to go in the program.
- DIRECT COMMAND     An instruction to the computer to do something straight away, in contrast with a program instruction which it does not execute until told to RUN.
- EXECUTION     What the computer does when it carries out your instructions, whether they are direct commands or statements in a program.
- FLOWCHART     A diagram used to show the path(s) that the computer will follow in working through the various steps in your program.
- GRAPHICS     Drawings and diagrams, in contrast with text etc.
- KEYBOARD     The set of keys that you use to feed information into the computer.

INSTRUCTION A message given to the computer to get it to do something.

LOOP A part of your program where the computer is forced to keep jumping back to follow some of the instructions over and over again.

MEMORY The part of the computer which stores both program instructions and any data they need.

PROCESSOR The part of the computer which actually carries out your instructions, and executes your program.

PROGRAM A series of instructions which are written by you, or somebody else, to get the computer to do a useful job.

RELATIONAL OPERATOR A mathematical or logical sign used to show the relationship between two pieces of data. Examples are equals (=) and greater-than (>).

REMARK A message in a computer program which is purely to explain what is going on, for the benefit of humans. REMarks are ignored by the computer itself.

STATEMENT Another name for INSTRUCTION.

STRING A piece of data which consists of one or more letters or other characters, and which the computer treats purely as a group of characters.

SUBROUTINE A part of a computer program which is separate from the rest, and arranged so that the computer "jumps" to it and carries out its instructions from various points in the main program.

VARIABLE Simply, the name of a location in memory where the computer stores either a number (numeric variable) or a string (string variable). Wizzard gives numeric variables single-letter names like A and F, and string variables similar names ending in a dollar sign -- like A\$, J\$ and P\$.

VIDEO SCREEN The screen which the computer uses to show you things. With Wizzard, it is the screen of your TV set.



## APPENDIX B -- A SUMMARY OF WIZZARD'S BASIC

### 1. DIRECT MODE COMMANDS:

- CLOAD loads a program that has been saved on cassette, back into the computer's memory.
- CON tells the computer to continue running the program in its memory, from the point that it was told to STOP.
- CSAVE saves a program that is in the computer's memory, by recording it on a cassette tape.
- LIST prints out on the video screen a listing of the program lines in the computer's memory, in line number order.
- LLIST very similar to LIST, except that it prints out the program listing on paper, using your printer -- if you have one. (If you don't have one, LLIST will cause the computer to go into a "dead loop". The only way out of a dead loop is to turn the power off, and then back on again -- losing your program.)
- NEW tells the computer to "forget" the program in its memory, and get ready to store another one.
- RUN tells the computer to start executing the program in its memory, starting at the line with the lowest number.
- (CNT'L) + C tells the computer to STOP executing the program in its memory, and wait for a further command.

### 2. MAIN PROGRAM STATEMENTS

- DATA tells the computer that the numbers following it on the instruction line are data, to be used by a READ line later in the program.
- DIM X(n) tells the computer to keep a part of its memory to store n numbers in an array or matrix called X.
- END tells the computer that it has come to the end of your program.
- FOR variable = m TO n STEP p  
tells the computer to execute the following instruction lines (down to the line which says NEXT and the same variable name) a certain number of times. The first time



it should give "variable" the value  $m$ , then  $(m + p)$ , then  $(m + 2p)$  and so on, until its value is greater than  $n$ . If "STEP" and "p" are not given, the computer assumes you want a step size of 1.

**GOSUB n** tells the computer to go to the subroutine starting at line number  $n$ . Also tells it to "remember" where the GOSUB is, so that it can come back to the following line when it RETURNS.

**GOTO n** tells the computer to go to line  $n$ , instead of going on normally to the line following the GOTO.

**IF expression THEN statement**  
tells the computer to work out "expression", then if the answer is "true" or "1", to execute "statement". If the answer to "expression" is "false" or "0", the computer ignores the rest of the line and just goes on to the next line.

**INPUT n** tells the computer to print a question mark "?" on the screen, then wait for a number variable  $n$  to be typed in from the keyboard. If  $n$  is replaced with a string variable symbol like \$A, INPUT can be used to bring in a string variable.

**LET variable = expression**  
tells the computer to give "variable" the same value as "expression". If it doesn't already have a number variable with the name "variable", it will create one. The word "LET" is optional, so you can leave it out if you wish. (I.e., "A=5", or "M=N+2")

**LPRINT variable(s) or string(s)**  
tells the computer to print out on the printer the variable(s) or string(s) you specify. (NOTE: if you don't have a printer connected, LPRINT will cause the computer to go into a "dead loop" -- from which the only escape is to turn off the power and turn it back on again. This loses your program!)

**NEXT variable**  
tells the computer that this point in your program marks the end of the lines which it should loop around in following the "FOR" statement, on an earlier line, which uses the same variable name.

**PRINT variable(s) or string(s)**  
tells the computer to print out on the video screen the variable(s) or string(s) you specify.

PRINT TAB(n) works like the normal PRINT instruction, except that it starts printing n spaces in from the left.

READ a tells the computer to fetch the next data number it can find in a DATA line earlier in the program, and call the number a.

REM tells the computer that it should ignore the rest of this line, as it is only a "remark" to explain things to anyone who looks at your program.

RESTORE tells the computer that the next time it executes a READ instruction, it should go back to the first number in the first DATA line of the program.

RETURN tells the computer that it has reached the end of a subroutine, and should go back to the next line in the main program after the GOSUB that sent it there.

STOP tells the computer to stop running the program, and return to direct command mode. Similar to END, but STOP allows the computer to CONTINUE.

### 3. GRAPHICS AND SOUND

CLS tells the computer to "clear" the video screen, leaving it blank.

CHAR n,xxxxxxxxxxxxxxxx gets the computer to create a new graphics character and give it the code n. The 16 code characters after the comma are used to define the dot pattern of the new character.

COLOR n,c,b Changes the colour the computer uses to display on the TV screen the set of characters with character set code n. c is the code number for the new character colour, and b is the code number for their background squares.

PLOT x,y,c tells the computer to place a character on the video screen, in column x and row y. c is the code number for the character.

SOUND a;x, b;y, c;z tells the computer to play musical notes in its three music channels. a, b and c are the code numbers for the notes wanted, while x, y and z are code numbers for the length of the notes.

#### 4. FUNCTIONS

- ABS (x) tells the computer to find the absolute value of x.
- COS (x) tells the computer to find the trig. cosine of x.  
(x must be in radians)
- EXP (x) tells the computer to find e (=2.71828) raised to the power of x. Equivalent to the natural anti-logarithm of x.
- INT (x) tells the computer to find the integer or "whole number" part of x.
- LOG (x) tells the computer to find the natural logarithm of x, using the base e (=2.71828).
- RND (n) tells the computer to find a random number between 1 and n.
- SGN (x) tells the computer to find the sign of variable x. If x is positive, the answer will be 1; if x is negative, the answer will be -1.
- SIN (x) tells the computer to find the trig. sine of x.  
(x must be in radians)
- SQR (x) tells the computer to find the square root of x.
- TAN (x) tells the computer to find the trig. tangent of x.

#### 5. STRING FUNCTIONS

- ASC (string) tells the computer to find the ASCII code number, in decimal, for the first character in "string".  
"string" is a string variable.
- CHR\$ (expression) tells the computer to work out "expression", then find the character (number or letter) whose ASCII code corresponds to this number.
- LEFT\$ (string, n)  
tells the computer to form a new character string, made up from the same characters as the first n characters of the string "string".
- LEN (string)  
tells the computer to find the number of characters in "string".

MID\$ (string, p, n)  
tells the computer to form a new string of n characters, by copying n characters of "string", starting with the one p characters from the start.

RIGHT\$ (string,n)  
tells the computer to form a new string of n characters, by copying the last n characters of "string".

STR\$ (expression)  
tells the computer to work out the value of "expression", then form a string of digits which when printed out, show this value as a number.

VAL (string)  
tells the computer that "string" is a set of digits which represent a number, and that it should work out that number.

## 6. SPECIAL FUNCTIONS

PEEK (a) tells the computer to find the number stored in the memory location with address a. a must be between 0 and 65536.

POKE a,b tells the computer store the number b directly in its memory location with address a. b must be between 0 and 255 and a must be between 0-65536.



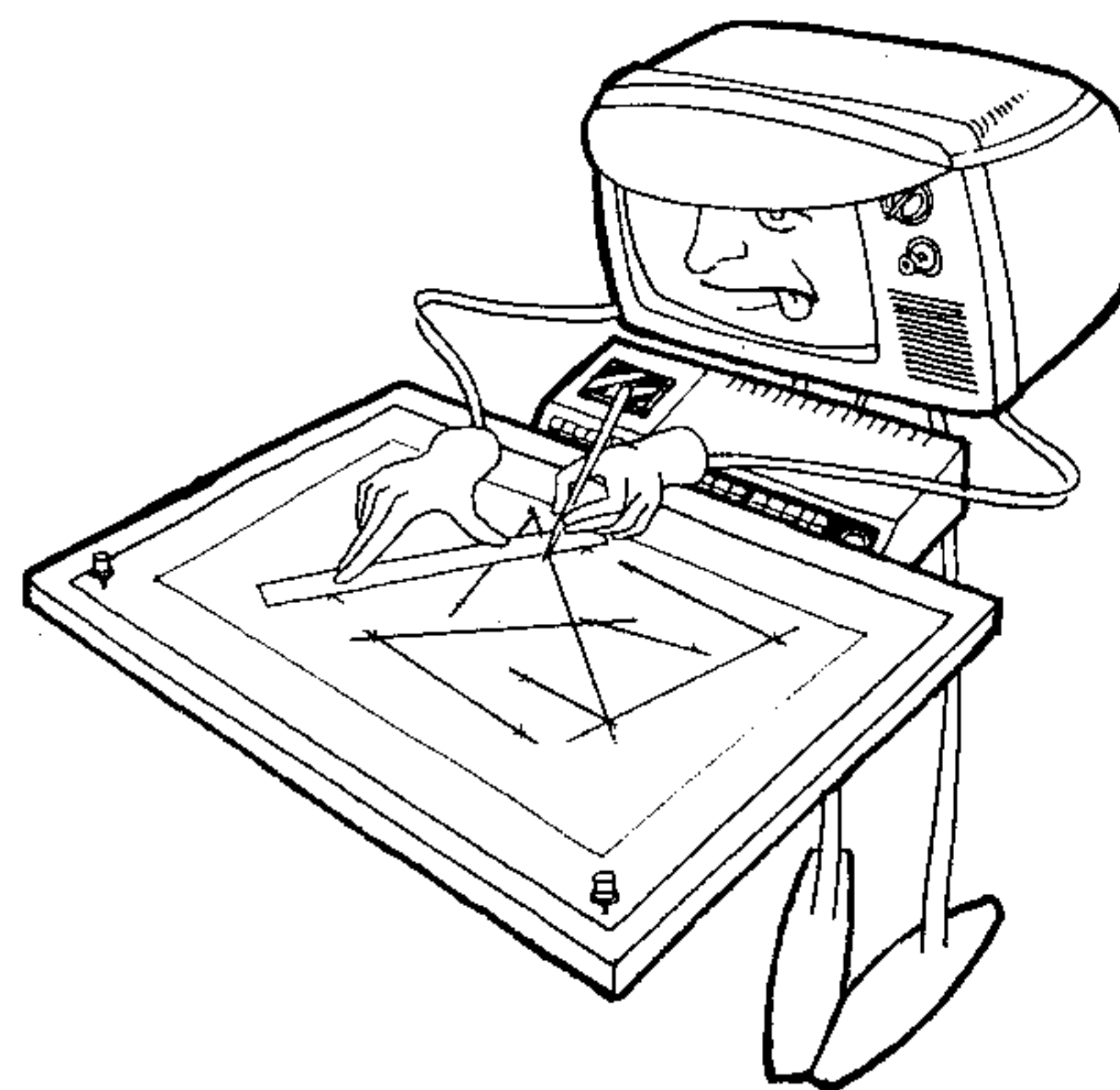
## APPENDIX C -- WIZZARD'S ERROR MESSAGES

If Wizzard comes across an error in your program while it is running, it will stop and give you an error message. This will be a number code, for the type of error, together with the program line which has the error. Here are the meanings of the various error codes:

- 01: NEXT WITHOUT FOR -- You have put a NEXT in your program without a matching FOR somewhere ahead of it. Either get rid of the NEXT, or put in the missing FOR!
- 02: RETURN WITHOUT GOSUB -- You have put a RETURN in your program at a place where there hasn't been a GOSUB. So either get rid of the RETURN, or add the missing GOSUB!
- 03: MISSING LINE NUMBER -- You have left a line number out of an instruction, or referred to a line number that doesn't exist -- like GOTO 150, when there isn't a line 150!
- 04: MISSING OPERAND -- You have used a function that needs an operand (the variable the function acts on), but you have left out the operand. Put it in, and try again.
- 05: SYNTAX ERROR -- You've made some sort of typing error, like spelling a word wrongly, or using wrong punctuation.
- 06: OVERFLOW ERROR -- Either you've typed in a number or a string that's too big for Wizzard to handle, or your program is too big to fit in its memory.
- 07: ILLEGAL NESTED FOR -- You have wrongly "nested" two FOR ... NEXT loops. Both the FOR and NEXT lines of the inner loop must be inside the outer loop, not just one of them.
- 08: ILLEGAL NESTED GOSUB -- You've probably got two subroutines which call each other, which would send Wizzard chasing its tail forever! Or you've forgotten a RETURN.
- 09: SYSTEM ERROR -- Either you've made an error Wizzard can't identify, or it has made a mistake itself (not likely!). Try turning it off and starting all over again.
- 10: STACK OVERFLOW -- Usually this means that you have tried to "nest" too many FOR..NEXT loops or subroutines. Wizzard just can't keep track of them all!
- 11: ILLEGAL OPERAND -- You've probably got a number variable as the operand of a string function, or vice-versa.

- 12: ILLEGAL IF ERROR -- You've made a mistake in the expression that Wizzard has to work out in an IF...THEN. It may not give an answer of "true" (1) or "false" (0).
- 13: PARENTHESIS ERROR -- You've made a mistake with your parenthesis brackets. Perhaps you haven't matched all the opening brackets with closing brackets.
- 14: PARENTHESIS LEVEL ERROR -- You have too many levels of parenthesis (brackets inside brackets inside...). Split the expression into two parts, in separate lines.
- 15: STRING NOT FOUND -- Your line specifies a string variable that Wizzard can't find. Probably a typing error.
- 16: STRING EVALUATION ERROR -- Wizzard has struck trouble in trying to find the VAL of a string. Probably you've told it to find the VAL of a string that isn't just number digits.
- 17: DIVIDE BY ZERO -- Your line has a expression that is asking Wizzard to divide a number by zero. It can't!
- 18: OUT OF DATA -- You have a READ line that has told Wizzard to fetch more data items than you have in DATA lines.
- 19: DATA AREA OVERFLOW -- Your program is trying to store too many DATA items. Wizzard has run out of storage space!
- 20: DIM ERROR -- Your line tells Wizzard to fetch an array element that wasn't defined in a DIM statement.
- 21: STRING LENGTH ERROR -- You have a string variable that is too long, or have made a mistake in a string function -- like telling Wizzard to find the 10th character in a string only 6 characters long.

Graphics? of course Wizzard  
can do graphics...



## APPENDIX D -- WIZZARD'S ASCII CHARACTER CODES

ASCII CODE	CHARACTER	ASCII CODE	CHARACTER
32	(space)	64	@ (at sign)
33	! (exclamation mark)	65	A
34	" (quotation marks)	66	B
35	# (hash or no. sign)	67	C
36	\$ (dollar sign)	68	D
37	% (percent sign)	69	E
38	& (ampersand)	70	F
39	' (apostrophe)	71	G
40	( (opening bracket)	72	H
41	) (closing bracket)	73	I
42	* (asterisk)	74	J
43	+ (plus sign)	75	K
44	, (comma)	76	L
45	- (minus sign)	77	M
46	. (full stop or point)	78	N
47	/ (slash)	79	O
48	0 (zero)	80	P
49	1	81	Q
50	2	82	R
51	3	83	S
52	4	84	T
53	5	85	U
54	6	86	V
55	7	87	W
56	8	88	X
57	9	89	Y
58	: (colon)	90	Z
59	; (semicolon)	91	[ (open sqr bracket)
60	< (less than)	92	\ (backslash)
61	= (equals sign)	93	] (closing sqr brkt)
62	> (greater than)	94	^ (up arrow)
63	? (question mark)	95	_ (line)

## APPENDIX E -- ANSWERS TO QUICK QUIZ QUESTIONS

### Quiz 1

- 1: Nothing -- a computer has to be told what to do because it can't even think for itself.
- 2: (c) A computer is like a slave -- it follows our instructions but it does what we tell it to do very fast.
- 3: The set of instructions given to a computer is called a PROGRAM.
- 4: Because it was programmed by a chicken (this is not a serious part of the quiz -- we just thought you might like it).
- 5: (b) AND (d). When telling a computer what to do or PROGRAMMING it, you must tell it in detail AND in the right order everything you want it to do.

### Quiz 2

- 1: (d). Baluns (pronounced like "balance") are used to connect Wizzard to your TV if you have the wrong sort of aerial socket.
- 2: (b). You must turn Wizzard off before plugging in your BASIC cartridge.
- 3: Your TV should be on VHF channel 1.
- 4: Because he was out of baluns (balance -- get it?)

### Quiz 3

- 1: (a). A processor or CPU follows the orders in the program you have given it.
- 2: (c). Executing an instruction is the same as obeying it.
- 3: BASIC is a simple language, very like English, that Wizzard can understand.
- 4: (b). LOADING a program means putting it into the computer's memory.
- 5: (c). When a computer RUNS a program it executes the instructions in the program.
- 6: (c). Wizzard lists the program in its memory by showing it to you.
- 7: Because it got loaded and had to run.

### Quiz 4

- 1: (d). Wizzard needs line numbers to know the right order for following your instructions.
- 2: (b) AND (d). A command, like RUN and LIST doesn't have a line number. This is because it is not part of the program, it simply tells Wizzard what to do with the program.
- 3: 10, 20, 30 etc.
- 4: Lines are given numbers with gaps between them so that you can add extra lines later if you forget anything.
- 5: (c). A programming loop is what happens when Wizzard goes back to an earlier line. One way of getting Wizzard to loop round is to put a GOTO instruction in your program.



### Quiz 5

1: Change lines 5 and 10 to the following:

```
5 PRINT "5 + 3 =";  
10 PRINT 5+3
```

(Now make the changes and run the program)

2. Change lines 5 and 10 again to this:

```
5 PRINT "7 - 4 =";  
10 PRINT 7-4
```

(Make these changes and run it again)

3: (c). END tells Wizzard that your program has ended.

4: (b). A semicolon at the end of a print line tells Wizzard that it should not shift down to the next printing line on the screen, after printing.

### Quiz 6

1: (c). A = A+1 tells Wizzard to take the number in A, make it one bigger, then put it back in A.

2: Your program should look like this:

```
10 A=1  
20 PRINT A;" X 13 =";  
30 PRINT A*13  
40 A=A+1  
50 IF A=16 THEN END  
60 GOTO 20
```

3: An IF...THEN instruction tells Wizzard to look at something, and if it has reached a certain size then follow the instruction on the second part of the line. Otherwise it ignores the rest of the line and goes to the next line.

### QUIZ 7

1: (b). CLS tells Wizzard to clear the TV screen.

2: B>A means B is greater than A.

3: B<A means B is less than A.

4: A decision diamond.

### QUIZ 8

1: (b). A subroutine is a part of your program that is used a number of times, by "calling" it from the main program.

2: (c). It's a good idea to use a subroutine when you find the same instructions at different places in your program.

3: Because Wizzard wouldn't know where to go back to in the main program, when it came to the RETURN instruction.

4: As many times as you need to -- there's no limit!

5: (c). A subroutine must end with the RETURN instruction.

### QUIZ 9

1: (b). Wizzard can play 30 different notes, from tenor C (an octave below middle C) to the F above high C.

2: (c). A rest is a musical pause.

3: (c). The tune is Happy Birthday, of course.





# DICK SMITH Electronics

SHOPS OPEN 9AM to 5.30PM  
(Saturday: 9am till 12 noon)  
BRISBANE: Half hour earlier.  
ANY TERMS OFFERED ARE TO  
APPROVED APPLICANTS ONLY



NSW	145 Parramatta Rd	
	T55 Terrace Level	
	613 Princes Hwy	
	552 Oxford St	
	818 George St	
	531 Pittwater Rd	
	147 Hume Hwy	
	162 Pacific Hwy	
	396 Lane Cove Rd	
	30 Grose St	
	6 Bridge St	
	125 York St	
	173 Maitland Rd	
	263 Keira St	
	Tamworth Acda & Kable Ave.	
ACT	96 Gladstone St	

AUBURN	648 0558
BANKSTOWN SQ.	707 4888
BLAKEHURST	546 7744
BONDI JUNC.	387 1444
BROADWAY	211 3777
BROOKVALE	93 0441
CHULLORA	642 8922
GORE HILL	439 5311
NORTH RYDE	888 3200
PARRAMATTA	683 1133
SYDNEY	27 5051
SYDNEY	290 3377
TIGHE'S HILL	61 1896
WOLLONGONG	28 3800
TAMWORTH	66 1961
FYSHWICK	80 4944

VIC	399 Lonsdale St	MELBOURNE	67 9834
	260 Sydney Rd	COBURG	383 4455
	656 Bridge Rd	RICHMOND	428 1614
	Springvale & Dandenong Rds	SPRINGVALE	547 0522
	205 Melbourne Rd	GEE LONG	78 8766
	Ross Smith Ave & Nepean Hwy	FRANKSTON	783 9144
QLD	293 Adelaide St	BRISBANE	229 9377
	166 Logan Rd	BURANDA	391 6233
	842 Gympie Rd	CHERMSIDE	59 6255
SA	60 Wright St	ADELAIDE	212 1962
	435 Main North Rd	ENFIELD	260 6088
	Main South & Flagstaff Rds	DARLINGTON	298 8977
WA	414 William St	PERTH	328 6944
	Wharf St & Albany Hwy	CANNINGTON	451 8666
TAS	25 Barrack St	HOBART	31 0800
Mail Order Centre: PO Box 321, North Ryde 2113. Ph: (02) 888 3200			