DICK SMITH'S Easy Way to Programming in BASIC using the System 80 Computer

$9.95

how to load & run pre-recorded programs

40 SAMPLE PROGRAMS

HOW TO PROGRAM GRAPHICS

by John & Judy Deane

Full explanation of BASIC error messages

handy tips on debugging

Ideal for the beginner!

Cat. B-6200

Foreword from Dick Smith

Hi there! Welcome to the exciting world of computing, and to the rapidly growing ranks of people using the incredible new System 80 computer...

The idea behind this book is to provide you with a really easy to follow introduction to programming in BASIC, using the System 80 computer. The System 80 is an exciting new machine based on the "Level II" version of the Tandy TRS-80 computer, which has been the largest-selling computer of all time. The System 80 uses virtually the same components as American made computers like the TRS-80, but it is assembled in Asia to take advantage of lower labour costs. This allows it to be sold much cheaper, providing outstanding value for money and bringing the advantages of computer technology to people who could never afford it before.

Along with its lower price, the System 80 also offers an inbuilt cassette tape deck which gets around most of the troubles often caused by a separate audio tape recorder. It is also designed to give you the choice of using either a video monitor or a standard TV set for its display. So there's no need to buy a video monitor, if you already have an unused TV set -- nor do you need to lug around a monitor when you're taking your System 80 to a friend's place!

The System 80 is also designed for later expansion using the popular S-100 interconnection system -- opening the door to almost unlimited expansion. Plug-in boards for the S-100 system are made by hundreds of manufacturers all over the world.

In short, the System 80 is not only excellent value for money, but an ideal "first computer" on which to learn programming!

By the way if you're impatient like me and want to get your System 80 running with pre-recorded programs straight away, turn to Appendix D at the back of the book after reading chapter 1!

Happy computing,

Dick Smith

National Library of Australia Card Number
and ISBN 0 9595080 3 1

PRINTED IN AUSTRALIA

Dick Smith's

Easy Way to Programming in BASIC

Using the System 80 Computer

by John and Judy Deane

FIRST EDITION, 1980

Published by Dick Smith Electronics Pty Ltd,
Sydney, Australia

Preface by the Authors

While this book is all about programming the System 80 computer, it is not a Programming Manual. If you want something that is gritty, concise and complicated then this is not the right book for you.

On the other hand, if you have not even played with a computer before, then this should be just the book for you! It will be especially useful if you have a System 80 computer in front of you while you read!

The objective here is to help you make your way through the ins and outs of programming with the System 80. If you follow us all the way you should pick up enough about the BASIC language, programming and the System 80 to really do your own thing!

There are lots of fun programs published of BASIC computer games that you can have a great time copying into your System 80 (though sometimes minor changes will have to be made). However, the satisfaction of writing your own program, even a simple game, is incomparable!

Computer programming is not an arcane art or even a mysterious science. It is a creative HUMAN activity. It requires a little education (read on!) and some concentration but you can do it!!

John and Judy Deane

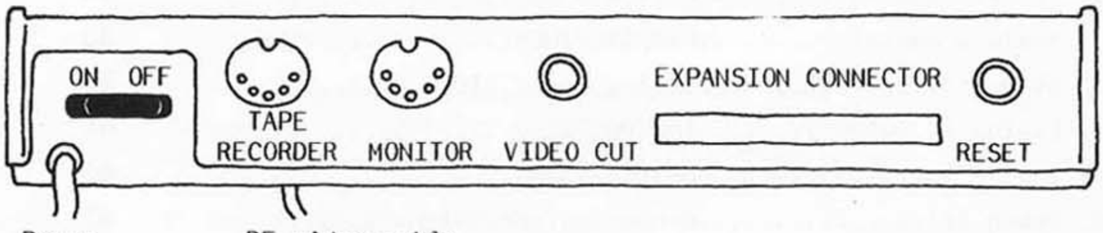
Contents

1. Can I turn it on?

Well, you've got a System 80! And I guess you'd like to use it!

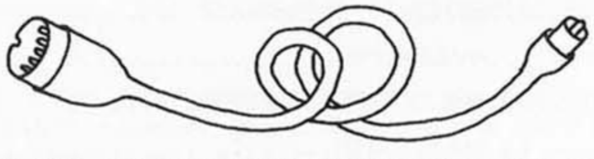
Find somewhere comfortable (you could spend some time with your System 80), secure (it would be a pity to have a card table fold up under you!) and out of the weather (computers don't like rain, dirt or tornadoes too much). Now let's see what we're doing.

This is the back of your System 80:



Power RF video cable

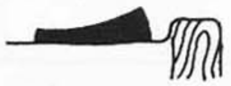
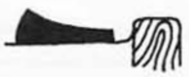
* If you're going to use a Video Monitor you'll need to use the composite video cable, ie.



Plug the fat DIN end into "MONITOR" in the back of System 80 and the smaller RCA plug into the back of your monitor.

* If you're going to use your TV set for display you can't use the above cable but the RF video cable built into System 80 will need to be connected to the aerial socket of your TV set. Select channel 1 to see System 80's display. You may find that you'll have to adjust brightness or contrast or even vertical or horizontal hold on your TV, to get a clear and steady display.

OK, you've got System 80 connected to your display, now you can turn it on. Don't forget System 80's power switch at the back right.

(from the top  = off &  = on)

System 80 should have a red light on at the left and your display should be showing

READY?_

near the top.

If you don't see that, refer to your System 80 User's Manual for more details.

You might like to use double size characters, especially if you're using a TV set for your display. At the back, press the "VIDEO CUT" button in (till it locks). When that's in, the grey "PAGE" button above the keyboard becomes active. ie.

with VIDEO CUT in

PAGE — Shows the left half of the display, 32 characters wide. This is the normal position.

PAGE — Shows the right half of the display (there's often nothing here at all!)

with VIDEO CUT out — The display shows 64 smaller characters (and PAGE doesn't do anything).

Now that you've got

READY?_

displayed at the size you want it, press the large

NEW LINE

key. System 80 will sort itself out, then display

```
READY
>_
```

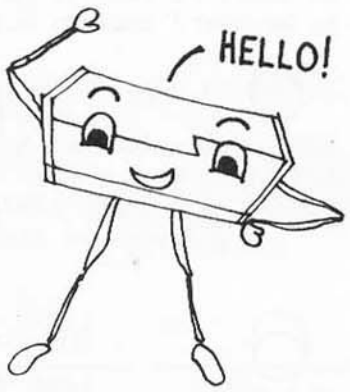
at the bottom of the screen.

NOW YOU'RE AWAY !!

We can get on with our look at the language that System 80 uses. It uses BASIC, a language formulated as the Beginners' Allpurpose Symbolic Instruction Code.

(You may think of other words to describe it later, but these are the recognised ones.)

2. Let's say hello



Now that we really have a computer let's see that it works.

At this stage the video display should be showing

```
READY
>_
```

at the bottom of your screen. The > is the prompt sign which means that the System 80 can accept your type-in now and the underline is the cursor which shows where the next character typed will go. The total display means that your System 80 is ready and waiting for you to say something. What you type in must be just right but if you make a mistake you can correct it by simply pressing

BACK SPACE

to remove the last character typed, then retyping it correctly.

Now type in the following line exactly as it appears.

```
PRINT "HELLO"
```

Can you find the single large key on your keyboard called NEW LINE ? This is to tell the System 80 that you have finished that line and your computer should look at what you have just typed in, so press it.

What happened? The display should have gone to the next line and shown

```
HELLO
READY
>_
```

Even if you left off the last quotation mark you'll still get this display but it may cause trouble later when you want to type in more, so remember to put quotation marks at both ends of the text following PRINT.

If you got some other message check that you
- spelled PRINT correctly
- left no spaces in the word PRINT
- used only one pair of quotation marks.

Try it again. Did System 80 say hello this time? You can make your System 80 say anything you like by changing the text inside the quotation marks and it will answer exactly what you type in.

NOTE that PRINT and the quotation marks did not appear. They are not supposed to be displayed and are only to tell the computer what to do.

Now type in the following line

```
1Ø PRINT "HI, I'M SYSTEM 8Ø"
```

Be sure to use numeric zero, here represented by Ø, rather than alphabetic O.

Now hit (NEW LINE)

This time System 80 didn't talk to you did it? This is because of the number at the beginning of the line which tells your computer that what you have just entered is part of a program (Yes indeed! A real program!) to be stored in memory rather than acted on immediately. READY is not displayed this time as the appearance of the word would just be a nuisance at this stage of the game. But notice that the prompt and the cursor are displayed on the next line ready for you to type in your next instruction. So type

RUN (NEW LINE)

This time you should have seen

```
HI, I'M SYSTEM 8Ø
```

appear on the next line.

If, Heaven forbid, it did not, then the System 80 could not understand something. Go back and type the line in again and, as long as you use the same line number the new line will replace the old. If RUN still doesn't get you the right thing something is thoroughly mucked up in the program. This time type

NEW (NEW LINE)

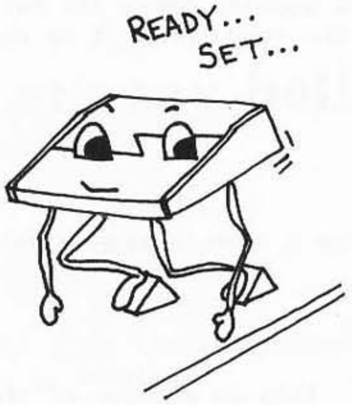
before re-trying the line. That should zap ANYTHING left over!

The command NEW tells System 80 that you want it to erase any existing program in its memory and "wipe its slate clean" ready to feed in a new program. You should always use this command before feeding in a new program.

Now did it all happen for you ?????

WELL NOW, WHAT HAVE YOU DONE?
That is your FIRST program and you have made it work!!!

We can distinguish now between 2 different things you can say to the System 80. Lines that you type in and that are executed immediately (NEW LINE) is hit are BASIC <u>commands</u>. Lines that are typed in to be executed later are BASIC <u>statements</u>. Statements make up a program and will execute when the command RUN is given.



Unlike the first lines we tried which were once-only displays, your program statement can be recalled. If you hit

RUN `NEW LINE`

a second time your System 80 will say hello again. No matter how many times you say RUN the computer will answer because the instruction you have given is still in memory. Doesn't that make you feel powerful?

What have we learnt in this chapter?

Commands	Statements	Other
RUN NEW	PRINT "text"	BACK SPACE NEW LINE READY > prompt _ cursor quotation marks line numbers

3. Programming already?

A one line program does not seem quite like the real thing!
So let's do some more. Type in

```
20 PRINT "AT YOUR SERVICE!"   (NEW LINE)
```

If that looks all right try typing

```
RUN   (NEW LINE)
```

Hmmm?? That was a bit like what we did last time, but now you can see

```
HI, I'M SYSTEM 80
AT YOUR SERVICE!
```

Right! Your first program has grown!
Surely you haven't forgotten what you typed in? Well, probably not, but anyhow let's type in

```
LIST   (NEW LINE)
```

NOW you should see

```
10 PRINT "HI, I'M SYSTEM 80"
20 PRINT "AT YOUR SERVICE!"
```

You should have figured out what PRINT does, but there is that funny little 10 and 20 before the PRINT. These are line numbers. They say where in the program lines should go. Type

```
5 REM TEST PROGRAM   (NEW LINE)
```

now try

```
LIST   (NEW LINE)
```

again.

WOW, that last line is now at the TOP of the program!!!
Aha; in line number order! You *can* write a program with line numbers as 1, 2, 3 etc., but, say you make them 10, 20, 30 instead, then up to 9 program corrections and additions can go inbetween any 2 old lines.

Actually, line numbers can be any numbers from 0 to 65529 (that's a lot of lines!).

OK, but what if I type a line number the *SAME* as one already in the program?? Well! We've already discovered that haven't we? The line just typed in completely replaces the old one. That sounds a bit dangerous! Maybe so, but that is how you correct a line with mistakes. Just retype it!

However, if you spot the errors before you have actually finished typing in the line itself, you can erase the whole line by pressing

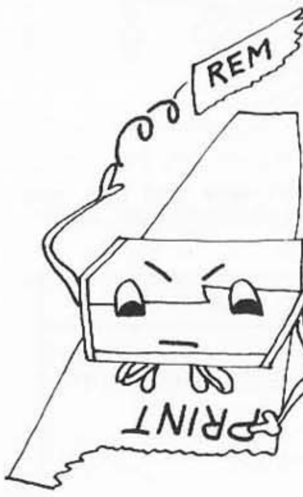
(SHIFT) with (BACK SPACE)

So much for all that stuff. What we have now is a 3 line program.

Let's RUN (NEW LINE) it.

Good Heavens, something must be wrong. We added a line but it ran just the same way! Remember that line was

```
5 REM TEST PROGRAM
```



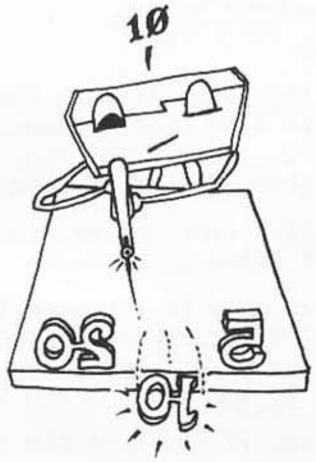
and in BASIC when you say REM that means REMark. The computer leaves such REM statements in your programs and shows them when you list them, but it completely ignores them when the programs are actually running - it understands that they are solely for the use of humans.

What good are REM statements if the computer ignores them? Simple: they allow anyone else, or yourself at a later date, to list the program and see what it is doing and how it is doing it. Program requirements change, and sometimes the computer setup does too. REMarks provide shining little gems of illumination to what could otherwise become a very misty program landscape!

Let's pretend we have done something a bit dumb (we're only pretending remember) and we want to totally annihilate a line of program. Type in

1Ø (NEW LINE)

And RUN it again. I'm beginning to get a bit weary of saying (NEW LINE) (*"Hey look at this!"*) all the time. As you can see it is always used as the terminating keystroke of a line. So from now on, even if I don't say NEW LINE at the end of a line, you will know that you must type it there.



What were we doing? Oh yes, when we ran then only the second line of text appeared! To see what we did type

LIST

(now really, you didn't forget that (NEW LINE) did you?)

Right! The middle line has gone entirely! We deleted line 1Ø just by typing its line number with no program line!

Now for a bit of "computer" type stuff. We nearly have enough bits and pieces together to have a bash at our own programs.

Very soon System 80 will start telling you things it has found that it does not like. For instance, if you asked your computer to

```
10 PRNT"HI"
```

it does not notice the missing 'I' when you press (NEW LINE). (The computer just stores the line away.) BUT when you tell it to RUN System 80 will say

```
?SN ERROR IN 10
READY
10 _
```

and stop running your program. Here 'SN' stands for SYNTAX and it means that there is something wrong in the way that the words are written in line 10; in this case,a spelling error in the word 'PRINT'.

System 80 has automatically entered "edit mode" for the line with an error and is waiting with the line for you to correct. We'll talk more about editing later but for now it is sufficient to press (NEW LINE) and you'll see the whole line containing the error. System 80 will now talk to you normally and will display the prompt and cursor showing that it is ready for you to type in the correct version.

There are many errors that System 80 can detect and these are listed, and explained, in Appendix B.

OK, now one giant step for you and a small one for mankind.
Type in NEW to remove all trace of program. The screen will be cleared and

```
READY
>_
```

will appear at the top.

NOW write your very first own program to print out your own name!
(and RUN it too!)

Pause here for violins, mounting orchestral accompaniment then, preferably, peace and quiet.

You should have something along the lines of

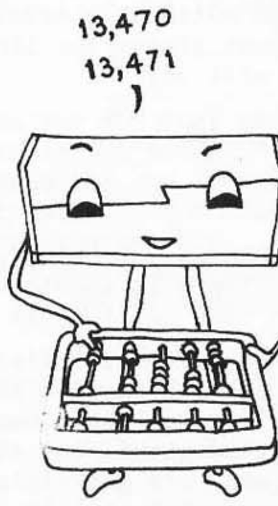
```
10 PRINT "U.R.WRIGHT"
```

Tremendous crescendo, gradually fading to close. Enthusiastic applause and we all go home for a good night's sleep.

In Chapter 3:

Commands	Statements	Other
LIST	REM	more about line numbers deleting errors

4. It really works!



Everybody knows that computers are good at mathematics. Wrong! Computers ARE good at arithmetic; add, subtract, multiply, divide, but it takes people to construct programs to solve problems in mathematics. A computer is only as good at maths as its programmer is!

Your System 80 talks arithmetic with a BASIC flavour that is not too different from "normal".

So, for add we use	+
for subtract	-
for multiply	*
and for divide	/

Hmm, those last two are a bit funny - whatever happened to X and ÷ ? Well, it might be a bit confusing (and computers *can't stand* confusion) to use the same symbol as an alphabetic X for multiply, and ÷ just did not exist on the standard typewriter layout. So, * is pretty much like a special X, and / is the way we write fractions (eg.1/2). So there we have it ....

Before we go any further, let's talk a little about maths. No don't go all squeamish on me yet, I want to say my bit! Maths is not something to be frightened of (unless you faint at English grammar say). It is just a way to describe some things. You wouldn't use maths to describe Mrs Jones' new hat! But how about things involving numbers anyway, like sizes, times or distance? For instance, instead of saying "if your car travels at a constant speed for some time you can find out how far it went by adding a unit of speed for each unit of time", how about saying

distance = speed X time

Maths is a clever plot carefully designed and constructed by people to allow shorthand descriptions of the way we think the world works. That just could be useful!

So much for that excursion. Let's get back to the world of the System 80. First, type NEW and then LIST to make sure we have a clean slate!

You didn't forget NEW LINE did you?

OK, I'll stop nagging.

Now we can try the program line

```
3Ø DI=SP*T
```

You guessed it, DI is distance, SP is speed and T is time. We multiply SP by T to get distance as DI.

DI, SP and T are called <u>variables</u> and they can have different values. Or, to put it a different way, their values can vary! Gosh.

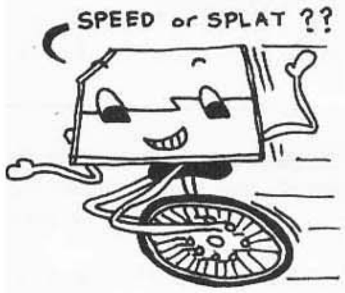
Variable names must start with a letter ie. A to Z, and the rest can only be made up of letters and numbers. Some examples of allowable variables are

```
A
Q
MAX
B4
ZING
SA569
```

and some illegal ones are

```
3ADD     (number first)
5        (that's not a variable, that's a number)
FOO!     (illegal character - not A to Z or Ø to 9)
&ALL     (not A to Z first, also illegal character)
```

BUT, but, but the System 80 only looks at the first two letters, so be careful naming variables. While saying SPEED certainly makes a program more understandable for humans than just using SP it can be hard to notice that two *COMPLETELY* different variable names look the same to your computer!
eg. System 80 would see no difference between SPEED and, say, SPLAT.



Also, if you use long variable names you have to be careful not to include any letter sequences that are used elsewhere in BASIC (these are called reserved words). So DISTANCE is illegal because TAN has a special meaning! See Appendix A for a list of reserved words.

OK, back to our program line.

```
3Ø DI=SP*T
```

What this says to the computer is to multiply (*) the variables SP and T together and put the result in the variable DI. Even though you could read that line as "DI equals SP times T" it does not *really* mean "equals". A better way to say "=" might be "takes the value of".

Now we have the 'gutsy' bit of a program, but we haven't given SP and T any values! (Actually, variables always have some value. If System 80 has not been doing anything else they should have the value zero - but that's not very useful.)

Let's say we go for a trip and each day we drive at an average speed of 7Økm/h for about 6 hours. Nice and leisurely! How far do we travel each day? That's not too tough, but let's see if your System 80 can do it. We need some program lines like:

```
1Ø SP=7Ø
2Ø T=6
```

When the program is RUN the variables on the left of the "=" are given the values of the numbers on the right hand side.

To speak pure BASIC one should really say "1Ø LET SP=7Ø" but with System 80 you can forget "LET" (or put it in, whichever you prefer). In the examples here I'll leave it out.

Numbers can be written like 1 or 345 or -82ØØ or Ø (no commas in big numbers!) for whole numbers. Or for fractional numbers 1.5 or -2395.66 or 3.14159 (this last one is "pi").

Now LIST your program; it should look like

```
1Ø SP=7Ø
2Ø T=6
3Ø DI=SP*T
```

Right, now RUN it!
But, hey,... nothing happened. It just said READY. Ready for what I ask you??
Aha,.. System 80 did what we asked it to do, and the computer knows the answer to our problem, but we did not ask it to tell us!!!
Ho hum. Let's add another program line.

```
4Ø PRINT DI
```

NOTE that there are no quotation marks around DI because we do not want the letters printed out, but the value that has just been assigned to the variable DI.

Now RUN it again.

```
If you don't see   42Ø
                   READY
                   >_
```

something is wrong! LIST the program and carefully check each line. Retype any lines that are wrong, or type NEW and go back and try this chapter again!

Now we can get an answer out!
It seems a bit bare though. Let's fancy it up some. Add

```
35 PRINT "THE DISTANCE IN KILOMETRES IS"
```

and RUN. Good! But how about writing the words and answer on the same line. Erase line 35 (by typing nothing for that line, remember?) Then try

```
4Ø PRINT DI,"KM IS THE DISTANCE."
```

HEY!! It says `42Ø             KM IS THE DISTANCE.`
That's better, but why all those spaces??
The display is divided into 4 fields or areas, each 16 positions wide. If you put commas between things to be printed then they go into these fields. This lets you compile tables easily. eg. try

```
PRINT 1,2,3,4,5,6
```

(This is an immediate command, not a program statement!) The numbers are spaced out, 4 to a line.
If you want things squashed up, use semi-colons between them. Try

```
PRINT 1;2;3;4;5;6
```

A bit different, eh? Now let's try

```
4Ø PRINT DI;"KM"
```

and RUN of course. You could even enclose your answer by the text describing it - for instance,

```
4Ø PRINT "THE DISTANCE IS";DI;"KM"
```

Also let's add

```
35 PRINT
5Ø PRINT
```

and now RUN.
That really is good! Those lines that look as if they do nothing space the result out nicely!

LIST out the program and have a good look at it. If something does not quite make sense, go back and re-read that bit of this chapter. It is vital that you understand what we have done so far.

I've changed my mind. I'm going to drive for 8 hours each day of our trip. How about you changing the computer's program to calculate how far we will go?

I'll just sit here and contemplate my navel for a bit while you adjust the program. GO ON ....

OK?? Well, my idea was to change line 2Ø to put a value of 8 into T. That makes our distance jump to 56Økm per day!

You may have already noticed that we are off to a bad start in programming - no documentation! Nothing to say what the program does!

You try saying what it does. Use a REM statement about line 5. One line can be almost 4 screen-lines long (System 80 stops accepting type-in at 24Ø characters) so that allows you to say an awful lot. However, if you want to say more just take a new line and use a new line number then say REM again and finish what you wanted to say. You can, of course, use as many REMs as you like (but don't kill any program lines; make sure you give the REMs unused line numbers).

I'll doodle again while you think of what to say ....

I thought it could go something like

```
5 REM THIS PROGRAM CALCULATES THE DISTANCE TRAVELLED GIVEN SP IN
KM/H AND T IN HOURS.
```

We've covered a lot of things in this chapter so give yourself a pat on the back and go and have a cuppa.

In Chapter 4

Commands	Statements	Other
	variable=expression LET var.=expression PRINT variable PRINT	+ add - subtract * multiply / divide variables numbers print fields & , print & ;

I'll give you a few exercises now. The answer to each is a separate program so you should type NEW before writing each. There is no "*correct solution*", there are only programs that work and "them that don't". Everybody programs a bit differently, and while some answers may be "better" for one reason (eg. faster) they may also be worse for other reasons (eg. incomprehensible).

Exercise 1.
You have to fly from Alice Springs to Perth (blimy!). You know that the aircraft flies at about 4ØØkm/h and that the trip takes about 5 hours. But how far is it?
(We already know that distance = speed * time.)

Exercise 2.
You are paying off a $25,ØØØ house loan at $28Ø per month for 25 years. How much are you actually paying the lender?

Exercise 3.
You are Science Officer on the Starship Enterprise and you are about to fire a proton torpedo at a fleeing fiend. Given that the Enterprise is travelling at 5Ø,ØØØkm/sec and proton torpedoes accelerate at 2,ØØØkm/sec^2. And knowing that V=U+A*T (V final velocity, U initial velocity, A acceleration and T time). How fast will the proton torpedo be going after 5 seconds?

Next, with the same conditions, how far will it have gone? We also know that S=U*T+1/2*A*T*T (here S is distance, and the others are as above).

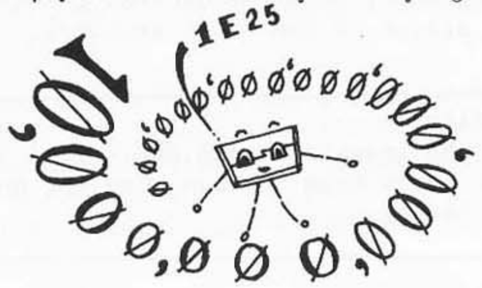
Good luck! See Appendix C for one set of answers to these exercises.

5. Big numbers

When you want to talk about very big numbers or very little numbers it is helpful to have a shorthand way of doing it. In fact we use shorthand methods all the time! For example, we buy material in metres, travel in kilometres and measure small things in millimetres, and here *metre* is the basic unit with *kilo* as a shorthand way of saying *thousands* and *milli* as a shorthand way of saying *thousandths.*

Another shorthand form of 1ØØØ is $1Ø^3$. All the superscript 3 means is 1Ø*1Ø*1Ø, 3 lots of 1Ø multiplied together and this "3" is called the power, or exponent, of 1Ø. System 80 cannot write above the line, so instead it uses E notation, meaning exponent of 1Ø. In this format $1Ø^3$ could be written as E3. So if you want to talk about the distance to the sun, rather than saying 15Ø X $1Ø^9$ metres, you can express it for the computer as 15ØE9, which is simply a different way of saying the same thing and is understandable to your System 80. It's also easier than saying 15ØØØØØØØØØØ isn't it?

If that number of zeros is hard to write correctly, think about 1E25!!
(1 followed by 25 zeros)



The same number can be written many different ways; thus 15ØE9
is the same as 15E1Ø
or 1.5E11 .

Your computer will accept any of these forms as the same number, but it likes to print in the last form, and does so automatically for numbers bigger than 999,999. That is, the number is expressed as a decimal between 1 and 1Ø with a positive exponent.

```
For instance,  1,732,ØØØ,ØØØ  would be printed as  1.732E+Ø9
         and  689,ØØØ,ØØØ,ØØØ,ØØØ             as  6.89E+14   etc.
```

OK, that's big numbers, now little ones.

One millimetre = $\frac{1}{1ØØØ}$ metres or Ø.ØØ1 metres.

This can also be written as $1Ø^{-3}$ where the "-" means that the number is divided (not multiplied) by the given exponent.... just another bit of shorthand. So if you want to talk about a millimetre you can enter it as 1E-3 and your System 80 will know what is meant.

Again, System 80 will accept many different forms of the same number.

```
For instance;       3.2E-8
                 or 32E-9
                 or 32ØE-1Ø  will all be accepted as Ø.ØØØØØØØ32.
```

But it will only be printed in the first form by System 80. That is, as a decimal between 1 and 1Ø with a negative exponent.

Expression of numbers in this form is called scientific notation, because science often deals with very large numbers (eg. astronomy) or very little numbers (eg. atom research).

You may have noticed that when you enter a number using scientific, or E, notation if the exponent is positive you need not give it a sign but when System 80 prints it out the "+" is always there. The computer has a standard format for printing these numbers which is;

```
                           x E ± ab
        number expressed      ↑    exponent given
        in decimal format   sign   in two digits,
        between 1 and 1Ø           eg. Ø6, 26 etc.
   (may be either positive
            or negative)
```

If the exponent is positive it means that the decimal point should be moved as many places to the right as the exponent says. If it is negative, then the decimal point must be moved to the left as many places as the given exponent.

Exercise 4.
If light travels at 3ØØ,ØØØ km/sec and takes about 5ØØ seconds to reach Earth from the sun, how far apart are they? Write all numbers in E format.

Exercise 5.
Try Exercise 3 again, using E format numbers, and changing 5 seconds to 25 seconds.

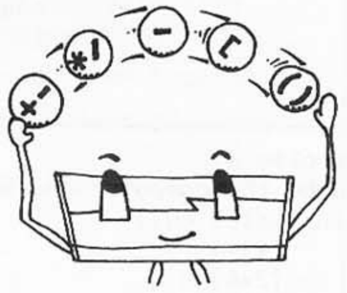
In Chapter 5

Commands	Statements	Other
		E format

6. Getting complicated

(*NOTE:* Don't worry if you find some of the maths in this chapter a bit beyond you. Just skip over it for now and go to Chapter 7. You can come back and refer to the maths later....)

Now we can put an arithmetic expression together, such as 1+2*3. BUT do we mean 1+2 then *3 OR 2*3 then +1? To answer this question we have an established order for performing arithmetic operations. This is called operator precedence or what comes first.



System 80 does * and / before + and - .
Hmmm.... * and / have the same precedence - which one gets done first? Well, they just get done from left to right. So the answer to 6/2*3 is definitely 9 - not 1! And 1+2*3 is 7.

Sometimes these rules will mean that we don't get the answer we want without some juggling. Thus 1+2 times 3+4 cannot be written 1+2*3+4. Why? Think about it...., yes, of course, it's because 2*3 is done first, then +1 and +4. We can use parentheses to get over this! Anything inside ( ) will be done first. So, to get the answer we want we could say (1+2)*(3+4).

Parentheses cannot be used for multiplication as they are in written maths. For instance, 2(3+4) will not be calculated as twice the answer to 3+4. If you want this, you must write 2*(3+4). You would still use parentheses to compute 3+4 first but you need a multiplication sign too.

That's all under control, so let's add a few more functions to our repertoire. If we want to multiply a number by itself we can write T*T. In everyday use this is T squared or T^2. System 80 understands T[2 as the same thing, that is two Ts multiplied together.

NOTE: There is no [key on the keyboard. To get [on the screen you must press ESC . ↑

```
Thus;     A[5  means A*A*A*A*A
        TWO[2  means TWO*TWO
                 (the contents of variable TWO multiplied by itself)
```

This is a lot like E notation since it involves exponentiation, but it is more general because the number to be raised to the given power is not automatically taken to be 1Ø.

How could you write 9.8E5 using [? Yes, simply 9.8*1Ø[5.

Another operation is unary minus or negation. This is similar to the minus we all know about, but refers to a complete expression. So if we say -A we mean negative the value in variable A. This has the effect of making a positive number negative, or a negative number positive.

Now we can complete a table of precedence. Operations are done in this order;

1. expressions inside ( )
2. [exponentiation
3. - negation
4. * / left to right
5. + - left to right

Exercise 6.
You be the computer and work out the value of se (then check them with System 80!);

```
1.     1+3*4
2.     (2+4)*2
3.     2[3-1
4.     -3[2     (careful here!)
```

While we're talking about this sort of thing it is appropriate to add that System 80 has a whole lot of numeric functions available.

With all of them you feed a number in (the argument) and get another one out (the result).

eg. with the function ABS, the *argument* is returned as a positive number *(the result)*.

See the System 80 Basic Manual for a complete list of available functions. These are some of the more common ones.

absolute value	=	ABS(arg.)	gives argument as a positive value eg. ABS(3.9) is 3.9 ABS(-19.6) is 19.6
integer part	=	INT(arg.)	gives whole number less or equal to argument eg. INT(5.9) is 5 INT(-3.1) is -4
sign	=	SGN(arg.)	result is -1 if<Ø, Ø for Ø, 1 if>Ø eg. SGN(-515Ø) is -1 SGN(19.5682) is 1 SGN(Ø) is Ø

square root	= SQR(arg.)	result [2 equals argument eg. SQR(16) is 4
exponential	= EXP(arg.)	result is *e* [argument (*e*= 2.71828)
log base *e*	= LOG(arg.)	reverse of EXP, ie. argument = *e* [result
sine	= SIN(angle)	for a right-angled triangle, sine is the ratio of opposite side length over hypotenuse length
cosine	= COS(angle)	ratio of adjacent side over hypotenuse
tangent	= TAN(angle)	ratio of opposite side over adjacent
arc tangent	= ATN(arg.)	reverse operation of TAN, result is an angle

NOTE: angles are in radians

```
radians = pi/18Ø * degrees
degrees = 18Ø/pi * radians
    and pi = 3.14159
          eg. TAN(45*pi/18Ø) is 1
```

Exercise 7.
You have to replace the cord in the Girl Guides' flagpole. It is a bright sunny day so you can get to work. With trusty protractor you measure the angle made by the end of the pole's shadow with the top of the pole as 55^{o}. Also the shadow is 6 strides long (say metres). How much cord will you have to buy?

(Hint - look at TAN & degrees to radians!)

In Chapter 6

Commands	Statements	Other
		rules of precedence
		[exponentiation
		( ) parentheses
		functions; ABS
		INT
		SGN
		SQR
		EXP
		LOG
		SIN
		COS
		TAN
		ATN

7. Decisions, decisions

So far our program only goes zoom, clunk and ends. The whole point of a computer is to be able to do boring, or entertaining, things over and over (and over and over....)! System 80 has some statements that let repetitive things be done easily.

Type in NEW to clear things out, then

```
1Ø PRINT "DECISIONS!"
2Ø GOTO 1Ø
```

and tell it to RUN.

NOTE: GOTO means exactly what it says. Here line 2Ø tells the computer to go back to line 1Ø and do that line again. By the way, some versions of BASIC require you to write GOTO as two separate words, ie. GO TO . With System 80 BASIC you can write it either way, but for convenience we'll write it as one word in this book.

Well it sure goes!! Look at that! Do you want to stop it?? I guess that *would* be sensible. Press the

BREAK

key. You should see

```
BREAK IN 1Ø
READY
>_
```

Peace again! Now type CONT. That means CONTinue and that is exactly what it does! Try the BREAK and CONT operations a few times.

BREAK is a command to *BREAK OFF* what System 80 is doing and jump back to BASIC and show READY.

That was all very exciting, but not awfully useful. We need some more things to control that GOTO stuff. A way of stopping it after enough has been done would be useful. Try

```
15 IF COUNT=1Ø THEN END
```

That's a start. We could control the print with a count. Now we must start our count somewhere

```
5 COUNT=Ø
```

and also do the counting.

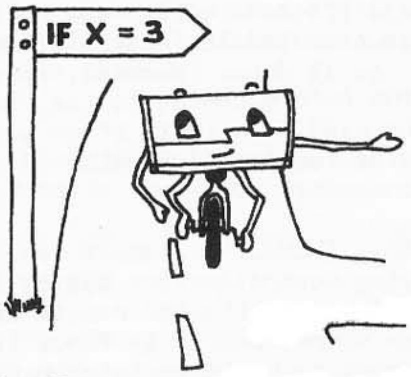
```
11 COUNT=COUNT+1
```

Put those lines in and type RUN.

Look at that! It printed 1Ø times and stopped! Wow! That IF/THEN statement did all the work, and it's a beauty. Let's have some more fun with it. Add

```
14 IF COUNT=7 THEN PRINT"I'M NOT FINISHED YET"
```

Then RUN it. Well, fancy that! DECISIONS! was still printed 1Ø times even though there was another line included and extra words to print.



Now that we've added these bits and pieces it might be a good idea to LIST the program. We have

```
5 COUNT=Ø
1Ø PRINT "DECISIONS!"
11 COUNT=COUNT+1
14 IF COUNT=7 THEN PRINT"I'M NOT FINISHED YET"
15 IF COUNT=1Ø THEN END
2Ø GOTO 1Ø
```

Line 5 is only acted on once and serves to start the counting at the chosen starting point. (It *initialises* the count.)

Line 1Ø is the fundamental line where we really carry out the prime objective of the program.

Line 11 keeps track of how many times we have gone through this part of the program. In this instance, it is how many times the PRINT statement has been executed.

Line 14 makes use of the count that we are making in line 11 and introduces the first of our decisions.
When the condition COUNT=7 is fulfilled, in other words, after lines 1Ø and 11 have been executed 7 times, then the extra message I'M NOT FINISHED YET is printed, as you saw when you ran the program. *IF*, however, the count does *not* equal 7, the extra message will *not* be printed and the computer will just move on to the next line.

Line 15 also uses the count that we've set up and stops the program when we have had enough.
ie. when COUNT=1Ø, and *only* then, END is encountered and System 80 has finished making decisions.
If the count does *not* equal 1Ø, then your computer will *not* act on the rest of the line and will simply continue.

Line 2Ø is the next line to be executed if the count is not 1Ø. So, until the program has been gone through 1Ø times, System 80 will keep going back to line 1Ø.
NOTE that it does not begin again at line 5 but only repeats the PRINT at line 1Ø then adds to the COUNT and then reconsiders the conditional statements at lines 14 and 15 with the new value of COUNT.

Lines 14 and 15 are examples of *CONDITIONAL BRANCHING* statements. The program will do something else (ie. *branch* from the rest) if it passes the test given in the *condition*. (eg.in line 14 IF COUNT=7)

Line 2Ø is an example of an *UNCONDITIONAL BRANCHING* statement. Any time that System 80 reaches the GOTO it will *branch* to the given destination (in this case, line 1Ø) *without condition*.

Not all programs will need an END statement (line 15). When System 80 has executed the last statement there is nothing else to do but end - so it does. However, as you can see here, if you want to END the run before coming to the end of your listing an END statement can be used. In fact, often you *need* to include END to stop your computer running on to other lines when they're not wanted.

The IF...THEN... statement can obviously be very useful and is worth studying more closely. Basically you can use it to make the computer check for a particular result or "condition", and then do something (or go somewhere) if it finds it. In other words, you can use this statement to build decision making into your programs.

In general, it takes the form;

IF condition THEN statement

The condition is written;

expression relation expression
where the "expression" can be;
a variable,
a number,
or a calculation like (A+6)*SIN(Z) etc.

and the "relation" can be;

=	equals
<	less than
>	greater than
<=	less than or equal to
>=	greater than or equal to
<>	not equal

The statement can be any BASIC statement and it will be acted on when the condition is fulfilled.

It can be PRINT or END as in our program, or GOTO (etc.) or any calculation.

Exercise 8.
Change what we have so far put together to print "MORE" after the fifth line.

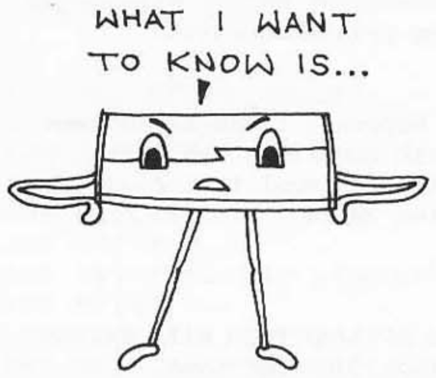
In Chapter 7

Commands	Statements	Other	
CONT	IF...THEN...	BREAK	
	GOTO		
	END	=	<=
		<	>=
		>	<>

8. Question time

It might seem a bit annoying that whenever we want to print a different number of "DECISIONS!", or print "MORE" in a different place, or possibly do something useful(!) we must change the program!

Along the same lines, our program can talk to us (PRINT) but we can't talk to it. Oh yes we can! And what allows that is the INPUT statement.



Let's type NEW and try this.

```
10 PRINT"I WANT A NUMBER"
20 INPUT NUM
30 PRINT "I SAW";NUM
40 GOTO 10
```

Type RUN and try that out for a while. You will see a question mark whenever System 80 expects a number typed in.

Now how about negative numbers?... Well, that's OK.

Big numbers? Did you remember not to use commas in big numbers? If you *do* include commas System 80 will take your input as more than one number; the first one being the digits before the comma and the remaining digits making up a different number. Since System 80 only asked for one number, when it reads two or more, it will display ?EXTRA IGNORED and continue with your program taking the first digits as the whole INPUT number.

Now try some E format input (well, up to about 1E38 anyhow. Anything much above that will produce an ?OV ERROR ie. overflow, or too big!)

Have you tried long numbers? We might lose a bit off the end, eg. 123.456789012 just comes out as 123.457 (Yes, even if you *do* type it in correctly as ....56.. it will be rounded up to ....57) This is the precision of the number which is determined by how many digits of each number can be stored. Actually it is the normal working precision of the computer, correct to 6 digits, but System 80 can work with greater precision. However, you need to specially ask for it before you can get this additional accuracy of 16 significant figures. Refer to the System 80 Basic Manual for a full description of single and double precision numbers.

Now try some garbage. For instance, answer System 80 with

```
GARBAGE
```

System 80 says ?REDO. Hey that's good! It *knew* that was garbage!

If you make an illegal entry such as typing in letters when numbers are wanted as you just did, then System 80 gives you another chance to type in the correct response.

Now change line 1Ø to

```
1Ø PRINT "A NUMBER PLEASE";
```

It's better to be polite isn't it? But a more important change is the semi-colon at the end. RUN it and see..... Well! System 80 is waiting for your number on the same line that he printed the request. That looks nice!

Now we can see the importance of the semi-colon. What it does as the last thing in a PRINT line is to wait, without taking a new line as would usually be done after a PRINT. So subsequent PRINTs will go on the same line, and input will go next to the printed question!

That is useful in its own right but, as it happens, there is an even simpler way to write the input. Programs ask questions and want answers so often that BASIC allows the INPUT statement to include the question too. Delete line 1Ø and change line 2Ø to

```
2Ø INPUT "A NUMBER PLEASE";NUM
```

Make sure that separator is a semi-colon as *nothing else* will do, not even a comma! RUN it again and it should look just the same.

Oh, did you notice? We took out line 1Ø, but left in 4Ø GOTO 1Ø. So we got the message ?UL ERROR IN 4Ø ie. unlabelled statement, or "1Ø" does not exist. Change 4Ø to

```
4Ø GOTO 2Ø
```

Now we're back in business.

Exercise 9.
Write a program to include an INPUT that can control how many times you print "QUESTIONS!".

You can ask for more than one item to be typed in with any INPUT line. When you're typing them in you'll need to use a comma to separate the items and this time the comma won't mean the rest will be ignored as it was before because System 80 will be expecting more than one INPUT item this time. As an example here is a short program that uses what we have looked at up to this stage.

```
1Ø INPUT "MAY I HAVE TWO NUMBERS";A,B
2Ø IF A>B THEN GOTO 5Ø
3Ø PRINT B;"IS THE LARGER NUMBER."
4Ø GOTO 1Ø
5Ø PRINT A;"IS THE LARGER NUMBER."
6Ø GOTO 1Ø
```

When you RUN this you may discover another aspect of System 80's operation. If two numbers are expected and only one is typed in System 80 will display two question marks on the next line to remind you that more input is wanted. So you really get quite a few chances from System 80 to type in acceptable input.

Exercise 1Ø.
Go back to an earlier exercise that you liked and use INPUT to control the numbers going into the program. Make the program continue by going back to ask the input question(s) again instead of stopping.

Now for some fun and games!
Here is another program that makes use of just about everything we have done so far. Astute observers are invited to plug the various loopholes in this little game! And, oh yes, it is possible to land without cheating!

```
100 REM SPACE LANDER
110 REM SETUP
120 HT=50000
130 SP=800
140 FUEL=8000
150 PRINT
160 PRINT"MANUAL LANDING ON TITAN-REPORTS EVERY 10 SEC"
170 PRINT
180 PRINT"HEIGHT","SPEED","FUEL","RATE"
190 PRINT"  M","M/SEC"," KG","KG/SEC"
200 REM MAIN LOOP
210 PRINT INT(HT),INT(SP),INT(FUEL),
220 RATE=0
230 IF FUEL>0 THEN INPUT RATE
240 IF FUEL=0 THEN PRINT
250 REM PHYSICAL MODEL
260 ACC=1.64-1932*RATE/(12000+FUEL)
270 HT=HT-10*SP-0.5*ACC*100
280 IF HT<0 THEN HT=0
290 FUEL=FUEL-10*RATE
300 IF FUEL<0 THEN FUEL=0
310 SP=SP+10*ACC
320 IF HT>0 THEN GOTO 200
400 REM END OF RUN
410 IF SP<15 THEN PRINT"SAFE LANDING AT";SP;"M/SEC"
420 IF SP>=15 THEN PRINT"THERE IS NOW A NEW CRATER";
INT(SP/15);"M DEEP"
```

If there is anything here that you can't grasp now, don't worry. Wait till you've had more experience with programming and you can come back later and work it out. However, this program shows what you can do with *only* our limited repertoire. It shows how powerful System 80 can be and what a terrific help, not to mention time-consumer. When you start playing and writing games like this you can spend *hours* on it and never be seen again.

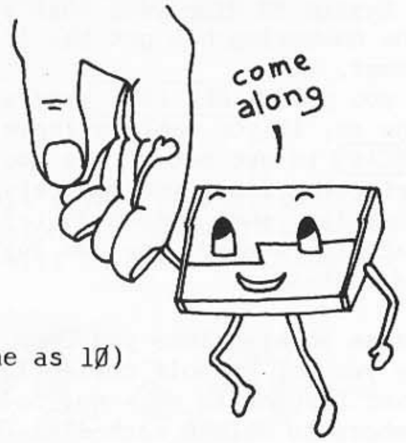
In Chapter 8.

Commands	Statements	Other
	INPUT variable	commas in input
	INPUT"question";var	?EXTRA IGNORED
		?REDO
		PRINT with ; at end

9. A little help

Sure, you have to do the programming, but System 80 provides a couple of commands to cut down on the amount of typing you have to do.

The first one lets System 80 number the lines for you! Type



```
AUTO
```

System 80 says

```
1Ø_
```

to ask for the statement for line 1Ø.
For instance, we could say (on the same line as 1Ø)

```
PRINT "HI!"
```

After you press NEW LINE , System 80 will store away that line as line 1Ø then ask for the next line by displaying

```
2Ø_
```

If you just press (NEW LINE) then, since you haven't entered anything, there won't be *any* line 2Ø in your program. System 80 just goes on to the next line and asks you to put in something for line 3Ø. Let's write a few lines such as;

```
3Ø PRINT "NOW IS THE TIME"          (NEW LINE)
4Ø PRINT "FOR ME TO SHOW YOU"       (NEW LINE)
5Ø PRINT "THAT I'M NOT JUST"        (NEW LINE)
6Ø PRINT " A PRETTY FACE."          (NEW LINE)
7Ø PRINT                            (NEW LINE)
8Ø PRINT "I CAN HELP YOU IN"        (NEW LINE)
9Ø PRINT "ALL SORTS OF WAYS."       (NEW LINE)
```

Well, that's enough isn't it? But System 80 doesn't seem to think so. Gee, we've got to stop this somehow! The computer just keeps asking for new lines! OK, to break out of this sequence just press (BREAK) and you'll see the good old

```
READY
>_
```

Now you could ask for a LIST or whatever seems appropriate.

You can also use AUTO to add on to what has already been done by typing

```
AUTO line number
```

where "line number" is the number of the next line to be entered.

For example, AUTO 1ØØ will continue from where we stopped before.

As if that's not enough, you can also say

AUTO line number, increment

this lets you change the AUTO stepping amount or increment from the normal 1Ø to any size you like.

If System 80 discovers that you already have a line where the AUTOmatic line numbering has got to, it displays an asterisk (*) after that line number.
If you press NEW LINE you will delete the previous contents of that line so, if you want to leave it undisturbed, you will need to press BREAK to get out of the routine. Then, if you want to get past the particular line, you can retype AUTO with a later line number. Otherwise, when System 80 tells you that you already have a line at that line number, you can just type in a new statement to replace the old contents.

System 80 also lets you undo all that good work you've done with AUTO. Say you put a whole collection of lines in the wrong place. (That never happens to me - no, no!) You *could* type in each of their line numbers to delete each line OR to delete the lot type in

DELETE start line - end line

So DELETE 1Ø-9Ø will remove what we've done so far. We could also delete those lines by typing

DELETE -9Ø

which will delete everything from the first line of the program, up to and including line 9Ø. The "end line" must be an existing line or you will get ?FC ERROR meaning System 80 can't work with the given number.

However the "start line" need not actually exist and System 80 will look for the next line following that number. For instance,

DELETE 5-2Ø	will give ?FC ERROR because there is no line 2Ø in our program.
DELETE 5-3Ø	will clear lines 1Ø and 3Ø, and
DELETE 55-8Ø	will clear lines 6Ø, 7Ø and 8Ø.

You can also delete one line by typing DELETE line number. eg, DELETE 7Ø would remove the spacing line from our program. So, now we know of two ways to delete a line. We can be positive and say DELETE line number or just type in its line number with nothing following.

In Chapter 9

Commands	Statements	Other
AUTO line number, increment DELETE line - line		BREAK with AUTO line* in AUTO

10. Just a calculator

Remember how we have said things like PRINT "HI" and PRINT 3 just to check how things work? (In Chapter 2) This use of commands is called *calculator mode* or *immediate mode operation* and, as the name implies, you can use that sort of operation to do calculations.

So try typing in

```
PRINT 1+2
```

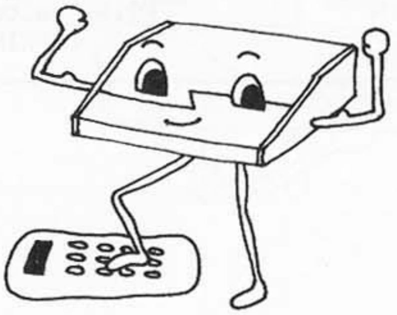
You should see the answer immediately! That's very significant! System 80 has worked out the little expression you gave it (ie. 1+2) before doing the PRINT of the result. It will do the same thing for very complicated expressions too. Try

```
PRINT 5E4*5+1/2*2ØØØ*5E2
```

Wow, that's the whole of Exercise 3 part 2 in one line! Or

```
PRINT 2*6*TAN(55*3.14159/18Ø)
```

That's Exercise 7 in one line.



If you are doing a lot of calculator type work, saying P-R-I-N-T might get a bit tedious, so System 80 has a quick way of saying PRINT which is just the character ?. Try typing

```
?"THIS IS STILL SYSTEM 8Ø SPEAKING."
```

or

```
? 1+2
```

which is a lot shorter than our first entry in this chapter but still gives the same result!

> Exercise 11.
> Get an immediate solution to give a rough value for your age in hours

Although we're talking about System 80 operating in immediate mode, it is a consistent machine, and some of what we have done can be applied to programs. For instance, a PRINT statement does not have to only be given "TEXT" or variables to display. An expression in a PRINT line in a program will be calculated and the result printed.

The one-line solution to Exercise 7 can similarly be programmed as, for instance,

```
10 REM ALT SOLUTION TO EXERCISE 7
20 PRINT "CORD IS";2*6*TAN(55*3.14159/180);"M LONG."
```

That's more to retype if you get the expression wrong but it *is* a lot shorter.

Just another little thing. We used ? as shorthand for PRINT. Well, the question mark can be used in a program too! You can enter the short form, but if you type LIST then System 80 will expand it to PRINT. Question marks in other places will stay question marks, but as a statement they become PRINTs.

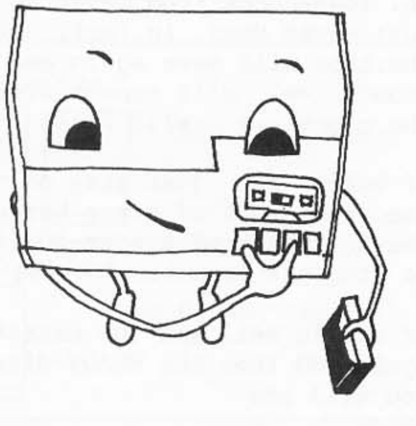
In Chapter 10

Commands	Statements	Other
PRINT ?	PRINT calculation ? (=PRINT)	immediate mode

11. Program saving

You can write quite complicated programs now but, if you turn off the System 80 everything will be lost! So.. you *could* leave your computer on *all* the time and preserve ONE program (not *really* recommended), ... OR you *could* type your program back in each time you want to use it (that's a daunting prospect, not to mention a boring and time-wasting exercise).

BUT, of course, there is another way and System 80 is already prepared for it. Programs can be recorded or "saved" on an audio cassette and System 80 has its own cassette recorder built in!



This single fact places System 80 ahead of quite a few other small computers because there are no troublesome adjustments of volume or tone settings to mar its usefulness.

To check out the procedure for program saving we'll need a little program. How about

```
1Ø PRINT "IT'S ME!"
```

Wow, that's really sophisticated!

First step, obtain your cassette tape. An ordinary audio tape is fine, though preferably not *too* cheap - faults that wouldn't worry your ear too much could wreck a program!

Lock down the grey key [F1], and the red light next to it should light. This puts the cassette under local (or your own) control, rather than System 80's influence.

Insert your cassette and rewind it. Now eject it again(!) and manually wind it past the non-recording leader until you see the brown tape coming through. Then insert it again.

Press [F1] again to release it (and the red light goes off). The cassette is now (partially) under System 80's control.

Hold down the "RECORD" key on the cassette player and lock down "PLAY". Both should stay down but the tape shouldn't move yet. Type in

```
CSAVE "M"
```

This is the instruction to save your program as a file called "M".

NOTE that quotation marks must be used or you'll get a ?TM ERROR ie. your type-in doesn't match what is wanted.

The tape should run for about 3 seconds with the red light on. Then the System 80 will stop it and display that it's ready again. Press "STOP" on the cassette player.

Well, that seemed OK, but did it *really* work? Let's check.

Press F1 down again for local control and rewind the tape. Release F1 and press "PLAY" (not "RECORD" this time). Then type

```
CLOAD?"M"
```

ie. load-check from cassette file named "M" to see that what you have just saved does, in fact, correspond with the program in System 80. The tape will move again and when the beginning of the program is found ** will appear at the top right of the display screen. The right * will flash for each program line recognised.

If both ** just stay on continuously, or they don't ever appear, then some sort of error has occurred. Press the "RESET" button at the left back of System 80 (from the top) to stop the computer looking at the tape.

If all is well and the cassette program matches the program inside System 80 then the READY display will be shown. If they do *not* match you will see

```
BAD
READY
>_
```

In this case, go back and save your program again.

When your program is saved and the check is satisfactory, the next step to try is to load the program from cassette into System 80's memory. Type NEW to clear things out, rewind the cassette and set it to "PLAY" with F1 up. Type

```
CLOAD"M"
```

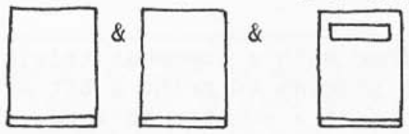
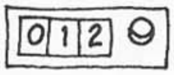
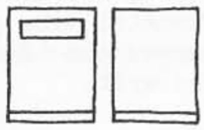
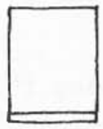
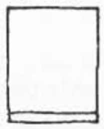
Those ** should appear again, then at the end of the load System 80 will let you know that it's ready to accept type-in again. Type RUN and System 80 will run your test program and print

```
IT'S ME!
```

That's full cycle:- out to cassette,
check,
clear System 80
and read the cassette back in!

When you have saved more programs on a tape and you specify one to be loaded, System 80 will search for and load only the program that you have asked for by name. Besides being enclosed in quotation marks that name must be a single character only. Then, to let you know what is happening, System 80 will replace the left * of the display with the letters of the programs it finds on the tape before it gets to the program you have specified.

To summarise

	F1 key	red light	
to position the tape	down	on	local tape control with REW & FF & STOP-EJECT  and  tape counter
to record on cassette	up	off	RECORD+PLAY  then CSAVE"name"
to check	up	off	PLAY  then CLOAD?"name"
to load into System 80	up	off	PLAY  then CLOAD"name"

The System 80 User's Manual has further details on cassette use.

As an added bonus, System 80's cassettes are compatible with those for the Tandy TRS-80 Level II computer. This means that any of the large number of program cassettes available for the TRS-80 Level II may be used with your System 80.

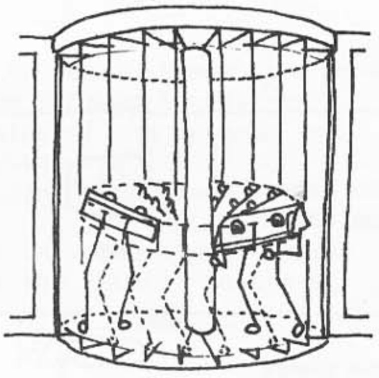
In Chapter 11

Commands	Statements	Other
CSAVE "name" CLOAD? "name" CLOAD "name"		cassette operation F1 key RESET button

12. Again & again

There are many times when you will have a little bit of program that has to be gone through a number of times. Often this number of times is known, or at least, can be worked out.

To continue with a somewhat trivial example. Some time ago you wrote a little program to print a bit of text ten times (in Chapter 7). There you set a counter to zero,
  did the print,
  counted it,
  then checked that the required part of
   the program had been done ten times.



Perhaps that seemed a bit complicated for what we were doing, though easier than 1Ø different PRINT statements! The people who designed BASIC agreed and to cut down the work a bit, we can write

```
1Ø FOR C=1 TO 1Ø
2Ø PRINT"AGAIN"
3Ø NEXT C
```

Try typing that in and running it. It's a much more satisfactory way of achieving the same result isn't it? Just as before, we can add something, like

```
25 IF C=5 THEN PRINT "AND"
```

and the extra word will be printed using the same counter "C" that we set up for the earlier part, without disrupting the loop.

What we have here are two statements, in this case lines 1Ø and 3Ø, that must be used in association:

```
FOR variable=start TO end
:
:
 statements to be executed
:
:
NEXT same variable
```

The FOR statement sets the variable to the start value (a number, another variable or calculation) and remembers the end value.

The NEXT statement adds one to the counting variable, and compares it to the end value. If it is less or equal, then the statements between FOR and NEXT are done again. If greater, then the statement after the NEXT is done.
    ie. that set of statements between FOR and NEXT is executed
        for each possible whole number from start to end.

If you miss out the FOR, or more probably, get the variable name wrong in the NEXT you'll get a ?NF ERROR message ie.NEXT without FOR.

If you miss the NEXT you won't get an error, but you also won't get any looping.

Exercise 12.
Print the values of X for X=1 to 1Ø.

That was a terribly easy exercise wasn't it? But it was just to make sure that you could set up a working loop. And it also serves to introduce our next point, which is that the variable used to count the number of passes through the FOR/NEXT loop may also be printed out or used for other purposes. BUT it should *not* be changed.

For example:
If a man is paid $1 for his work on the first day, $2 on the second day, $4 on the third day, and so on, with his pay doubling each day this FOR/NEXT loop could be used to work out how long you could afford to employ him if you have $1ØØØ for paying him.

```
1Ø PAY=Ø
2Ø FOR A=1 TO 5ØØ
3Ø PAY = PAY + 2[(A-1)
4Ø IF PAY = 1ØØØ THEN GOTO 8Ø
5Ø IF PAY > 1ØØØ THEN GOTO 7Ø
6Ø NEXT A
7Ø A = A-1
8Ø PRINT"HE CAN WORK";A;"DAYS."
9Ø END
```

If you enter this program and run it you may be surprised at the answer. But if you don't believe it try adding some lines to print out some other bits of information like his pay on his final day.

The program as written does the following:

Line 1Ø sets a variable that we wish to use in calculations, to the starting point of zero (initialises the variable).

Line 2Ø begins the FOR/NEXT loop and sets the end value of the count variable to 5ØØ. This value is determined by the conditions of the problem. Since $1ØØØ is the limit of the pay he can get and his pay will double each day, then $1ØØØ/2 should give an ample number of passes through the loop. It doesn't matter if this number is too big because the looping will be by-passed at the appropriate stage by the IF...THEN... statements of lines 4Ø or 5Ø.

Line 3Ø uses the count variable in a calculation to work out his pay so far. On the first day, when A is 1, his total pay is his previous pay ($Ø) plus $2[(1-1) which is $1. On the second day, when A is 2, total pay is previous pay plus $2[(2-1) etc.

Line 4Ø says that if the total pay has reached the available financial limit, then no more calculations need to be done and the answer can be given.

Line 5Ø says that if the total pay has exceeded the limit without actually reaching exactly $1ØØØ then we have gone too far. Obviously no more calculations need to be done so we'll branch out of the loop and get the answer.

Line 6Ø is reached if the total pay is not yet enough to cause branching at lines 4Ø or 5Ø. This is the end of the loop and simply says that if A is not more than 5ØØ the program returns to line 2Ø.

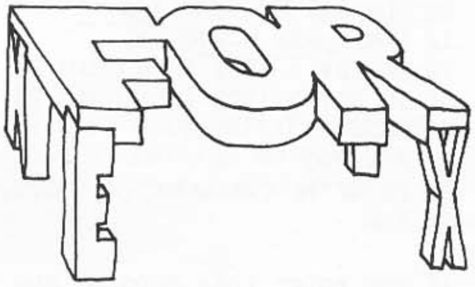
Line 7Ø is the line jumped to if we have gone too far (ie. from line 5Ø). It says that A should be 1 less because we can't afford to pay so much.

Line 8Ø is reached both by ordinary progression of the program from the previous line and by the branching in line 4Ø. It simply gives the answer using the variable A to tell us how many days he can work, ie. how many times we had to pass through the loop.

Line 9Ø ends the run.

Exercise 13.
Print the values of X and X[2 for X=1 to 1Ø.

Hey, look at that exercise.
You've generated a table!
It's not very complicated but it sure shows how to go about it. You get an expression for one entry of your table then use FOR/NEXT to run through all the values you want in the table.



Now try this one.

Exercise 14.
Generate a table of SIN(X) for X as Ø, Ø.1, Ø.2..... 2.Ø
(Hint - think about X from Ø to 2Ø.)

Now you can use your System 80 to generate 6 figure log. and trig. tables. Didn't you always want to do that?

Actually there is another little fiddle to let you do things like Exercise 14 a bit more easily. You can say

```
1Ø FOR X=Ø TO 2 STEP Ø.1
2Ø PRINT X, SIN(X)
3Ø NEXT X
```

Not bad eh? You get a default step size of 1 if you don't specify a STEP number, but you can state precisely what you want if you'd rather something different. You can use any step size you want, including negative steps!

However some combinations don't work so well, for instance STEP Ø never finishes or changes! Also, for a negative step, the end should be *smaller* than the start. eg.

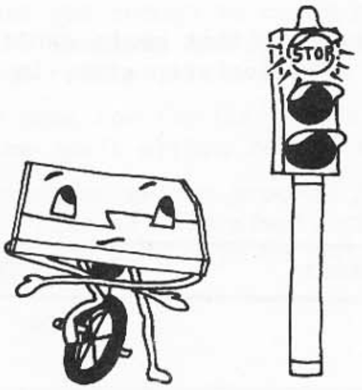
```
... FOR C=12 TO -6 STEP -1.5
```

You can see what that means can't you? It counts backwards, down to -6 with a negative step size, in other words, subtracting 1.5 each step.

Chapter 12

Commands	Statements	Other
	FOR : NEXT FOR....STEP... : NEXT	

13. Stop it!



Here we will talk about different ways of stopping different things. Delaying, pausing or terminating programs or listings... wow!

We've already talked about using the END statement to terminate a program, and pressing (BREAK) to stop a running one. However, there is another sort of pause. You can use the statement STOP (in a program) which will transfer you to immediate mode. When you want to go on, type CONT and your program will continue at the statement after the STOP.
eg. type NEW and

```
10 PRINT"STOP"
20 STOP
30 GOTO 10
```

When this is run you should see

```
STOP
BREAK IN 20
READY
>_
```

Type in CONT to go round again.
If you didn't have line 20 and the program didn't stop, then you could have used the (BREAK) key to stop the run and you would have seen the message

```
BREAK IN 10
READY
>_
```

and to continue from that you could also have typed CONT.

If you type CONT at the wrong time, for instance when System 80 hasn't previously come to a stop ((BREAK) or STOP), you'll get ?CN ERROR ie. how can I continue when I haven't even stopped?

You can do a similar kind of stop just from the keyboard. Let's change the program by deleting line 20. Now it will just run, and run... Try holding down (SHIFT) and pressing the key with (@). The program is now "suspended". It has stopped but did not show BREAK or READY.

In this state System 80 is not ready to accept any input from you, it just lets you have a good look at what's on the screen (though this lot isn't very interesting). In fact, you would most probably find if you tried to type something in that the program would just start running again because pressing almost any key will send it on its way once more (only SHIFT doesn't do anything and BREAK will give a fully-fledged break plus a return to immediate mode).

You can do another sort of pause with a delay loop. In this loop the program will look as if nothing is happening, but your ever-faithful System 80 will still be working away. To see this, try typing NEW and

```
10 PRINT"WAIT A SECOND."
20 FOR C=1 TO 345
30 NEXT C
40 GOTO 10
```

Here lines 20 and 30 form a delay loop, because line 30 tells the computer to add 1 to the counter C, then jump back to line 20 until it reaches the limit in the FOR statement, in this case 345, ie. it has to "loop back" 345 times and this takes about a second. To vary the delay time, change the limit from 345. For example, a limit of 3450 gives about 10 seconds.

OK, now to listings. Someday, with luck, you should have a listing longer than 14 lines. When that happens, and you type LIST, the first lines will disappear from the top of the screen before you have time to see them! If you want to make the listing pause, press the SHIFT and @ keys. Yes, that works for listings as well as programs. To continue press a key as before. Another way to stop it is to press BREAK and not only will the listing stop but you will also get the old familiar message

```
BREAK
READY
>_
```

meaning that it's your turn to type in something. But, unfortunately, since we've broken out of the listing you can't just type CONT to see the rest. However you *can* see the remainder of your program listing by asking for a LIST from the line number that you stopped at. eg. if the last line displayed was 1050 ... you can get the rest by typing

```
LIST 1050-
```

Yes, that little dash is part of something bigger. You can say

LIST
to list a whole program or

LIST start-end
to list from line "start" to line "end", then you can break that up so

LIST start-
lists from the chosen "start" line to the end of a program and

LIST -end
lists from the beginning to the chosen "end" line.

Also LIST line number
(without the dash) will list a single line only.

Now we can go back to our original problem and have another look at it. Unless you move very quickly, you will not be able to stop a listing in time to see the first few lines of your program. But now we know a better way don't we? Right! Ask for a listing of the starting lines only.

Easy isn't it? You can just list any part of a program that you want instead of having to start at the beginning every time. For instance, if you are writing a program and you have already checked the first half, you can ask for only the later lines to be listed and System 80 will jump to the lines you want. You can use any line number.. if it doesn't exist System 80 will just take the next one.

At this point I will add that something similar can also be done for running a program. You can RUN from a line number, so that again, you don't have to start at the beginning every time but can just look at the relevant part immediately. However with a RUN, the line number used has to be one that exists in your program or you will get a ?UL ERROR ie. no such line.

Let's summarise this bit.

	Program		Listing
	Statement	Type-in	Type-in
Terminate	END	BREAK	BREAK
Long pause	STOP	BREAK	BREAK
& Restart	type in CONT	RUN line no.	LIST line no.
Delay	FOR... NEXT loop	SHIFT @	SHIFT @

Exercise 15.
Write a program to be an interval timer. Ask for theinterval length and indicate its end.

What if your program somehow manages to go really berserk, so that nothing happens when you try to stop it running with the SHIFT @ keys or the BREAK key? This *can* happen!

When it happens, don't panic. You still have a card up your sleeve. At the left back of System 80 there is a small button marked RESET. If you press it System 80 should stop whatever it is doing and display READY. RESET returns control to you and will leave the program in System 80 intact.

If even this fails you'll have to turn System 80 off!! Sorry about that. To add insult to injury, don't forget that you need to wait 15 seconds before you turn it back on or you won't get any sense from System 80.

In Chapter 13

Commands	Statements	Other
CONT	STOP	delay loops
RUN line number		(SHIFT)(@)
LIST line-line		(BREAK) & LIST
		RESET

14. Laying it out

Now let's have a look at some things to help in setting out PRINTs. We've already learnt a few ways to let us make a printout look more readable and ordered. For instance, we know about including blank Print statements to space out lines nicely and we can make a table of our results by using commas in the PRINT lines, or we can keep relevant things together by using semi-colons.

So what else do we need to know?
One more instruction that can help you to set out your printout is the TAB function.

```
...PRINT...TAB(column number)...
```

where the subject material of the PRINT that follows the TAB will be printed beginning at the indicated column number (columns are numbered Ø to 63 from left to right).

The TAB instruction is similar to the use of commas in a PRINT where things will be printed in "fixed zones". But with TAB you can determine for yourself how far apart the columns will be. Try typing and running the following. (Don't use commas, because System 80 will get confused about whether to use the fixed zones or your chosen columns. You *can* leave out the semi-colons and get the same result, but I put them in for the sake of readability.)

```
1Ø PRINT"A";TAB(15);"B";TAB(3Ø);"C";TAB(45);"D"
2Ø PRINT 1;TAB(15);2;TAB(3Ø);3;TAB(45);4
3Ø PRINT -1;TAB(15);1TAB(3Ø);-1;TAB(45);1
```

You can see that with the same TAB numbers, the digits are offset one space from the letters. This is because the System 80 allows a space before any number, positive or negative, to give its sign, and you can see that in the third line this was needed.

The TAB function provides a simple way to plot out a numeric function, eg.

```
1Ø REM PLOT SIN(X)
2Ø FOR X=Ø TO 3 STEP Ø.5
3Ø PRINT X;TAB(SIN(X)*3Ø);"*"
4Ø NEXT X
```

Voila! A sine curve. Since these sine values will range between Ø and 1 we multiply SIN(X) by 3Ø to make the argument for TAB (ie. a column number) a reasonable size. In this example the "*" will appear between columns Ø and 3Ø, depending on the value of SIN(X).

A complete plot of sine will include negative values of SIN(X) too. To plot these they must be converted to positive arguments for TAB and the earlier values plotted will need to be "moved over" on the screen.

So if we add the same value to all of them (say 3Ø, to approximately centre the zero value on the screen) we will be able to get a continuous plot. With these points in mind, see if you can change the program to give one complete cycle.

These lines would do the trick.

```
1Ø REM FULL PLOT OF SIN(X)
2Ø FOR X=Ø TO 6 STEP Ø.5
3Ø PRINT X;TAB(SIN(X)*3Ø + 3Ø);"*"
4Ø NEXT X
```

Ah! At least line 4Ø didn't need to be changed.
You see that some of the curve got flattened on the left. Here, the TAB column number was less than what had already been printed and the TAB call was ignored. If you want to see the complete curve the argument for the TAB call will need to be changed - you could change the scaling factor (ie. multiply SIN(X) by something smaller than 3Ø) and/or alter the offset (add 35 instead of 3Ø).

Could you tell what determined the end value of the loop? Well, since the function SIN(X) works with values of X in radians, X needed to be *pi* to plot the positive values, and for our purposes 3 was near enough. To give the full plot a value of 6 or 6.5 (when you're stepping by Ø.5) would give an end value near 2 *pi* which is 6.28... I chose 6 simply because it was twice 3 and gave as much again as we had in the first half.

TAB arguments greater than 63 go on to the next line, but 255 is the largest legal argument! A negative number or more than 255 gives a ?FC ERROR ie. function call error - the argument given will not work in that function.

Exercise 16.
Generate a plot of X[2 for X from 1 to 7 with Ø.5 steps.

Exercise 17.
Plot SIN(X)/X for 1 to 14.

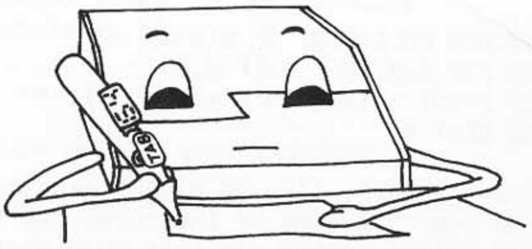
If, in the course of your program, you want to find out the current character position in a PRINT line then you can use the POS function. The argument for this function need have no real relevance in your program; it can be any letter or number. eg. try

```
1Ø INPUT"NUMBER";N
2Ø FOR I=1 TO N
3Ø PRINT "*";
4Ø NEXT I
5Ø PRINT POS(Ø)
6Ø GOTO 1Ø
```

and you will see a row (or more) of asterisks on the screen plus a number telling you how many character positions in the line have been used. For this function the positions are numbered 1 to 64 inclusive from left to right across the screen and begin again for each new line.

The TAB function gives you an easy way to set out results in an intelligible manner but of course there are other things you can do to make your printout look good or to emphasise the headings or just to separate things. Try typing

```
1Ø PRINT "THIS WILL BE FUN."
2Ø PRINT
3Ø FOR X=Ø TO 6Ø
4Ø PRINT "*";
5Ø NEXT X
6Ø PRINT
7Ø PRINT
8Ø PRINT "DID YOU LIKE IT?"
9Ø PRINT
```



Line 6Ø had to be included to finish off the row of "*" because the statement in line 4Ø ends in a semi-colon (so would print the next thing on the same line). Line 7Ø was included to PRINT another space to set things out clearly.

You can play around with these statements to give all sorts of effects in your printout and to make the information in your display easier to read.

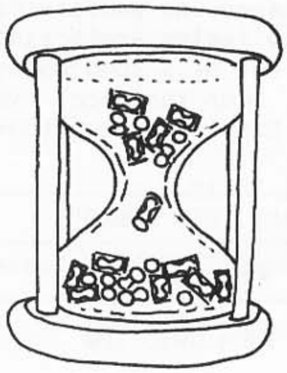
In Chapter 14

Commands	Statements	Other

		TAB(column)
		POS(dummy arg.)
		function plot

15. Inside out

We've already talked about FOR/NEXT loops, but this chapter will continue our acquaintance. Here we will have a look at "nested" loops. That just means putting one inside another, like a nest of coffee tables...



Using the one-second timer from Chapter 13 we can now make an egg timer.

```
1Ø REM EGG TIMER FOR 3 MINUTES
2Ø FOR S=1 TO 3*6Ø
3Ø FOR C=1 TO 345
4Ø NEXT C
5Ø NEXT S
8Ø PRINT "TAKE IT OFF!"
```

Here lines 2Ø and 5Ø form one delay loop and lines 3Ø and 4Ø another delay loop.

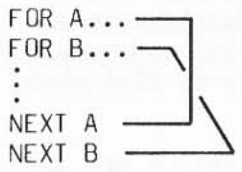
NOTE: the 3Ø/4Ø loop is fully *INSIDE* the 2Ø/5Ø loop! This means that the 3Ø/4Ø loop must be fully completed (345 loops) each time the computer goes around the 2Ø/5Ø loop (and it goes around this outer loop 3 times 6Ø, or 18Ø times). So the outer loop counts the number of seconds wanted, while the inner loop provides the basic delay for each second.

Exercise 18.
Using nested loops write a program to give a countdown to blast-off, counting off the minutes and marking the final time.

FOR/NEXT loops can be nested to any desired depth (ie. you can have as many loops inside loops, inside loops as you can work out how to use). BUT, the FORs and NEXTs must all be matched up correctly. So

```
FOR A...
FOR B...
FOR C...
.
NEXT C
NEXT B
:
:
FOR D...
:
:
NEXT D
:
:
NEXT A
```

(WOW!) is OK, but



```
FOR A...
FOR B...
:
:
NEXT A
NEXT B
```

is not nested and will fail with a ?NF ERROR message (ie. NEXT without FOR) at the line that says NEXT B, since when System 80 reaches that line, it will have completed the A loop which contains the FOR B line and it will be inaccessible.

Now we've seen how you can use nested loops and that you can't put half a loop inside another and the other half outside. However you can break out of a loop quite legally by using a conditional branching statement. For instance, type NEW (or if you don't want to type it in again, CSAVE your countdown and we'll use it later) then try

```
1Ø FOR A=1 TO 1Ø
2Ø PRINT"ALL THIS WORK!"
3Ø FOR B=1 TO 2
4Ø PRINT"  OH BOY!"
5Ø NEXT B
6Ø IF A=4 THEN GOTO 1ØØ
7Ø NEXT A
1ØØ PRINT"I'VE HAD ENOUGH!"
```

Actually, that program just zips through without really looking as if System 80 has to work *too* hard, doesn't it? It might be better to include some delay loops to make it look more effective. Add

```
15 FOR X=Ø TO 25Ø
16 NEXT X
25 FOR X=Ø TO 25Ø
26 NEXT X
```

(It was just as well we had spare line numbers wasn't it?) That looks better. System 80 said that it had done enough after only 4 cycles, even though we'd originally set it at 1Ø cycles, because line 6Ø told it to jump out of the loop - so it did.

Exercise 19.
Let's have some more fun with printouts and launch some rockets! We'll do this in a series of steps to build up a program. (This is not only a fun game, but also a serious programming exercise!)

1. Starting at line number about 2ØØ write PRINT statements to make a rocket. You may want more than one element in the body.
   (Hint - use "A" for the top,
   "H" for the body elements
   and "<A>" for the tail.)

   Run this bit of the program, and make changes if you need to, till your rocket looks OK.

2. Starting at about line 3ØØ, make a loop to generate an exhaust. Put in a delay loop of about 1ØØ counts between each exhaust element to slow it down a bit. (Hint - make about 8 lines with "*".)

   Again,run this till it looks good.

3. Start at about line 4ØØ and make a similar loop, but with blanks (and delays) to space between rockets.
4. Put a final loop right round the program to launch 1Ø rockets.

Wasn't that worth it? Now you can add Exercise 18 to the beginning of your rockets program and have a dandy little display that you wrote yourself, to show all your admiring friends.

In Chapter 15

Commands	Statements	Other
		nested FOR/NEXT loops

16. Debug it!

BUGS, the perpetual enemy of computer programmers! In this chapter we are going to talk about a few types of fly swatters.

Actually, we're not dealing with insects, arachnids, or shop specials on plastic swatters. "Bugs" in a program are errors that are not catastrophic; the program just doesn't work properly! Really, catastrophic errors are not too bad. If you get a message like "?SN ERROR IN 6Ø", then you know that it is line 6Ø that needs looking at. However, bugs hide. You know something doesn't work, but where is the error?

On with that in a moment. But first I should say that the presence of bugs is an integral part of programming. Not a desirable part, but for programmers from rank amateurs to hardened professionals, ever present and unavoidable. The trouble is that computers do exactly what you tell them to do. System 80 can't tell whether a sequence of statements doesn't quite make sense, or if it is OK. It just does it!

Down to business. Let's assume that we've written a program, but now that we run it we can see that it doesn't work (or at least not the way it was meant to). Oh, the *pain* of it all! What now? Well, the thing to do first is to look carefully at what did actually happen, and to try to work out from that what went wrong. Then try listing the program and checking your hunches.

In fact, an examination of your program could be enough. Have a good look at it, for statements that send the program to the wrong destination, necessary statements that aren't there, or calculations that take the wrong argument or ... No luck? Well, now you'll have to pretend to be the computer, and work through your program line by line doing just what the statements say, until the error becomes obvious.

To make this easier, you can temporarily put in some extra PRINT statements, and try running it again. Statements like

```
55 PRINT 55;"A=";A
```

to show the value of variable A, while also giving the line number to remind you what part of the program it came from, and identifying text to say what value we are printing. If you just want to see that the program is going to the right places in the right order, you could include

```
55 PRINT 55;
```

ie. just show the line number. That semi-colon at the end lets a lot of line numbers fit in one screenfull. Remember that you can press (SHIFT)(@) to check how these numbers are going at any time.

To get the hang of the sort of debugging that needs to be done, we can look at a little program with inbuilt errors! The program we've got is one intended to print out the prime numbers (those that can't be made up by multiplying two smaller numbers together). The idea behind the program is to take every odd number - obviously even numbers can't be prime, because they are divisible by 2 at least - and divide it by each number up to its square root to see if there is any whole number (integral) result. Those numbers that give no whole number result should be printed out as the prime numbers.

```
1Ø REM PRIME NUMBERS
2Ø FOR N=1 TO LIMIT STEP 2
3Ø FOR P=1 TO SQR(N)
4Ø IF N/P=INT(N/P) THEN GOTO 7Ø
5Ø NEXT P
6Ø PRINT P
7Ø NEXT N
```

Run this ..... and nothing happens! Try

```
25 PRINT 25;"N=";N;
```

and run it again. You'll see " 25 N= 1" only! It went one time round the main loop!! Why?...

Oh yes, we didn't set LIMIT to anything! Add

```
15 INPUT "LIMIT";LIMIT
```

Run it again. A limit of 5 should produce 1,3,5. Our debug line shows that we *really did* the outside loop 5 times with 5 different values but we didn't get any printout from line 6Ø. So, let's check the inside loop with

```
35 PRINT 35;"P=";P;
```

and run. Gosh,

```
 25 N= 1  35 P= 1  25 N= 2  35 P= 1 ..................
```

Um, well, OK. We saw all our Ns, but P only ever got to 1. So let's look further at the inside loop. It starts from 1, so what happens on the first cycle? P is 1 and, of course, any number divided by 1 is still integral and will satisfy the condition in line 4Ø. So every time we'll jump to line 7Ø and never find any indivisible numbers to print out. So we shouldn't be looking at P=1. Try

```
3Ø FOR P=2 TO SQR(N)
```

Another run with LIMIT=5 and we get 3 numbers printed out (as well as the diagnostic prints) BUT they are all 3! Try another LIST. What are we printing?

In line 6Ø we are printing P .... Is that what we want?.... A moment's thought...... Aha! No. Let's try N.

```
6Ø PRINT N
```

At last we get the right numbers. Now we can remove the debugging lines (25 & 35). Run again with some bigger limits. Hold it! Since we expect a lot of numbers to be printed out it might be better to

```
6Ø PRINT N;
```

(with the semi-colon) so that there will be less zipping around of the results on the screen. Right, now! My run looks OK. If yours doesn't then the rest is up to you!

There is another fairly similar way to do some debugging. You can insert a STOP statement in your program. eg.

```
25 STOP
```

When the program reaches this you will see

```
BREAK IN 25
READY
>_
```

Now you're in immediate mode and you can see what your program is doing by typing something like this

```
PRINT N
```

ie. an immediate print of any of your program's variables! Remember you can also say

```
?N
```

here - a little bit faster! To go on with your program type CONT.

If you want to change your program at this stage (not entirely unknown), you certainly can, BUT you cannot then CONTinue (you would get a ?CN ERROR). You must RUN it again.

System 80 has a function to give you some help with debugging. It is especially useful if you don't have a *clue* about where your program is going. With it you can trace the route that your creation is taking. If you type

```
TRON
```

to BASIC before you RUN your program you will get a *mass* of information about which line numbers are encountered. For each line executed you will see

```
<line number>
```

You'll probably *have* to use SHIFT @ to see what is happening!

We're all writing lengthy programs now aren't we? Or perhaps you're stuck in a loop! (Remember, press almost any key to go on.) To understand this display you'll have to make a list of the trace numbers to compare with your program.

Once your problem is sorted out (or you've had enough anyway!) you can stop the trace by typing

```
TROFF
```

There is another source of possible programming problems. The System 80 is a real computer, not a theoretical one, and one consequence of this is that it cannot hold all numbers with an exact value. As I said in Chapter 8, System 80 usually works correct to 6 places, so whole numbers from -999,999 to 999,999 are OK (as long as they *stay* integral). But System 80 may give unexpected results if you ask it to deal with numbers with *great* accuracy. To have a look at this sort of thing, try

```
10 BEG=100000
20 UP=0.01
30 FOR N=BEG TO BEG+UP*10 STEP UP
40 PRINT N;
50 NEXT N
```

Line 30 indicates that there *should* be 10 runs through the loop (the limit is 10 step sizes more than the starting value) and we can also see that the variable N *changes* by a small amount each loop.

However, even System 80 has its limits, and what actually happens is the number printed does *not* change, yet the run *does* finish. And from the number of PRINTs we can tell that the loop was gone through 14 times, not 10! Wow!

I'm not going to say any more about that than to emphasise that we are trying to work with numbers that are beyond the ability of System 80 to cope with exactly. There are not any clear guidelines here, except to be careful of long numbers, and when debugging, to print anything that controls the way a program works. It's a good idea to read the System 80 Basic Manual's details of high precision numbers so you can *fix* problems like this.

While we're talking about precision of numbers and unexpected results let's have another look at our program to plot a sine curve (in Chapter 14). In the first half, when the limit was 3, our step of 0.5 included a print for the value X=3. You would expect this since the value 3 can be made up exactly by adding our step sizes of 0.5 to the beginning value. What you would *not* expect is that if the step size is changed to 0.2 the final value of 3 will *not* be printed out. If you want a step size that small *and* the final value of 3 to be included in the printed results then this can be done by adding a small amount to the limit (say 0.05) and although this makes no theoretical difference to the values calculated, it *does* make a practical difference. So, once again, be careful with numbers that control the workings of your program. (You may even feel it's worthwhile making a habit of adding a tiny amount to the end value of a loop so that you don't encounter this problem.)

I've spent a fair bit of time on debugging, and that reflects the amount of time that you are likely to spend on it. Debugging is not simply a nuisance, but a facet of programming in its entirety.

In Chapter 16 (actually we've done most of these before)

Commands	Statements	Other
CONT (after debug STOP)	STOP	big numbers
? variable	PRINT line;"var=";var	
TRON		
TROFF		

17. Editing

Now that you know how to find errors in your programs it's nice to know that System 80 will help you to fix them. Instead of retyping *all* of a line containing an error you can just change the part that is wrong.

Let's start off with a line to change, say

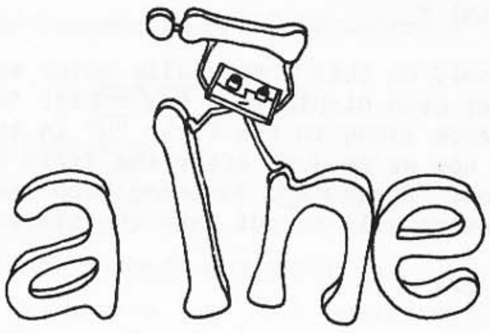
```
10 PRIINT "HI"
```

First we select the line we want to edit with

```
EDIT 10
```

When you type that you'll see

```
10_
```



No, you haven't destroyed line 10. That display just means "ready to edit" (in this case, line 10). This is just like what you see when there is an error - System 80 automatically enters edit mode!

You can move backwards or forwards in the line by pressing the space bar (forwards) or the

BACK SPACE

key (backwards). Try it! You can't *just* retype the line. You'll see that as you move along the line, the characters that you have previously typed in will appear but if you try to type in a new letter nothing will happen. Space and Backspace are the first commands that System 80 as an editor will recognise.

Now move to the first "I". (You did type two "I"s in "PRINT" didn't you?) OK, we have an "I" too many. To remove the next one type D for delete. You don't need to tell the editor what you want to delete. By just typing D you have indicated that you want to delete the next single character. So now you should see

```
10 PRI!I!_
```

Whenever you delete something, edit will tell you what it was with exclamation marks around it.

If you press NEW LINE at this stage System 80 will go to a new line and will show you the rest of line 10 as it will now be in your program.

Now if you decide that, after all, you would like to say a bit more you will have to type EDIT again since NEW LINE returned us to BASIC and left edit mode. So type

```
EDIT
```

Oops! I forgot to say which line I wanted so System 80 said ?UL ERROR. So this time let's say

```
EDIT 1Ø
```

That's better. Now how about inserting some characters? Space to

```
1Ø PRINT "_
```

You could do this a specially quick way by saying, after the line number has been displayed, (S)(H) ie. "search for 'H'". The computer will zoom along to the first "H" in the line and stop just before it. Well, now we're just after the first quotation mark, let's do the insertion. Press (I). Nothing happened. That's right, but any further characters will be put *into* the line. Type

```
LET'S SAY
```

(with a space at the end). Then we'll have to tell editor that this insert is finished. To do so press (SHIFT)(ESC) ie. "escape from insert". Again, nothing happened. But if you press the space bar now you will see that you're back to looking at the rest of the line and you can give other edit commands now if you wish.

If we wanted to add something to the end of the message, we could insert it before the last quotation mark or we could space to

```
1Ø PRINT "LET'S SAY HI_
```

and press (H) for "hack"! This will delete the rest of the line (in our case it is only the quotation mark) and allow us to insert new material. I think I'll add

```
 TO ALL OUR FRIENDS."
```

with a space at the beginning and a quotation mark to finish the message replacing the one we have just deleted.

Now you can press (NEW LINE) and return to BASIC.

Well! You can select a position, then delete and/or insert character(s) which covers a fair proportion of editing. But let's look at one more thing. If we go back to our sine curve plot again (Chapter 14), you will remember that the latest version of line 3Ø that I offered was

```
3Ø PRINT X;TAB(SIN(X)*3Ø + 3Ø);"*"
```

If I had decided to make the changes that I suggested to move the plot over on the screen so that the left-hand side was not flattened, instead of having to type in a whole new line I could have said EDIT 3Ø, spaced along to the first "3" (after the line number, of course), typed

(C) to change the next single character and then typed 2, to make the scaling factor 2Ø instead of 3Ø. If you wanted to add more than 3Ø at the end of the TAB argument, then you could just space till you see 3, press (C) and type 5 to change the number added into 35. Now when you press (NEW LINE) you should see

```
3Ø PRINT X;TAB(SIN(X)*2Ø + 35);"*"
```

This is a very convenient way to make small changes to see how they affect the printout that you're generating.

To summarise.

EDIT line number	immediate mode command to BASIC to select a line to edit
space	move right one character
backspace	move left one character
(S) c	search for next occurrence of character "c", and stop at position before it
(D)	delete next (undisplayed) character
(I)	insert typed characters into next positions (BACK SPACE) will now *delete* ended by (SHIFT) (ESC)
(H)	delete remainder of line and insert typed characters into next positions
(C) c	change next character to "c"
(NEW LINE)	display whole line, finish edit and return to BASIC

I think that this is a good "working set" of edit commands. When you've got these working smoothly have a look at the "Text Editing" section in the Basic Manual and you'll find a whole lot more commands for advanced users.

In Chapter 17

Commands	Statements	Other
EDIT line number		Edit commands: space backspace (S) character (D) (I) text (SHIFT)(ESC) (H) (C) character (NEW LINE)

18. Sub-programs

There are two distinct types of sub-program that we'll look at here. Both are usually of greatest benefit as programs get more complicated. One provides a way of splitting a long program into bite-size pieces, and the other lets you use the same series of instructions from wherever you like, without rewriting them!

To illustrate the first kind, let's say we want to write a program to play around with money! That could get a bit complex, but we'll take the essential things we want to handle as;

1 Debit	(bills)
2 Credit	(pay!)
3 Balance	
4 Adjustment	

(If that doesn't really grab you, then insert Warp, Map, Fire, Status instead for you science fiction freaks.)

Now we need to have a line in our program to distinguish the categories we're dealing with. (Of course, for a real program we should do some initialisation too, like set some balances to zero, or position your spaceships! So I'll leave some space at the start of our program.) Instead of writing four separate sections we could use this line

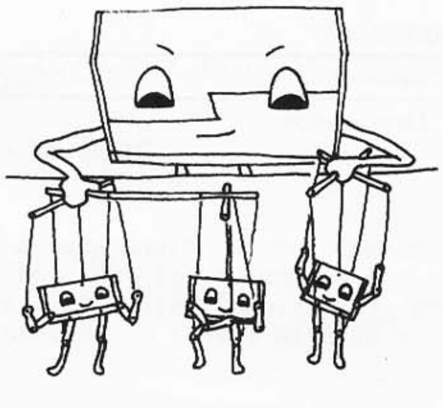
```
1ØØ INPUT"1 FOR DR,2 FOR CR,3 BALANCE,4 ADJ"; OPT
```

Then to link this to the rest of our program we could simply write

```
11Ø ON OPT GOTO 1ØØØ,2ØØØ,3ØØØ,4ØØØ
```

That looks impressive! What System 80 does here is to look at the value of OPT.

```
For OPT = 1  it will  GOTO 1ØØØ
for       2           GOTO 2ØØØ
for       3           GOTO 3ØØØ
for       4           GOTO 4ØØØ
```



Simple really, isn't it?
We have a multiple GOTO statement.

ON var. GOTO .... takes a variable (or expression) and uses its value to count along the list of line numbers to select one to GOTO. Actually the computer takes INT(var.) so you don't have to use strictly integral values (but remember that, say, 3.9 is taken as 3). This variable may have any positive value up to the number of line numbers in the list. If you exceed the number of possibilities (ie. the number of lines to GOTO) by any number up to 255, or have a zero value, then System 80 will just do the next line! Negative numbers, or those above 255 will give a ?FC ERROR (argument incompatible with that function). You can have as many line numbers in your list as you can fit on a line. But they must exist!! (If not you'll get a ?UL ERROR ie. unlabelled line.)

The rest of the skeleton program in the above example would be something along these lines

```
1000 REM DEBIT SUBPROGRAM
:
1999 GOTO 100
2000 REM CREDIT SUBPROGRAM
:
2999 GOTO 100
```

etc. (you can fill in the details - only the hard work is left to do!) Each subprogram performs the desired task, then causes a jump back to the INPUT line to get a new entry.

As a summary of the possibilities and restrictions of this statement, type in this program.

```
10 REM TRY-OUT OF ON...GOTO
20 INPUT "WHAT IS YOUR NUMBER";NUM
30 ON NUM GOTO 70, 80,90
40 PRINT"WELL. YOU DIDN'T EXCEED 255 BUT"
41 PRINT"YOUR NUMBER WAS GREATER THAN 3,"
42 PRINT" OR WAS IT ZERO?"
50 END
70 PRINT"YOUR NUMBER WAS TAKEN AS 1"
75 END
80 PRINT"YOUR NUMBER WAS TAKEN AS 2"
85 END
90 PRINT"YOUR NUMBER WAS TAKEN AS 3"
```

This is also an example of the use of the END statement, in lines 50, 75 and 85, used to stop the program running on to later lines. It wouldn't make any sense, would it, if each of the various PRINT lines was displayed every time? But, gee it gets a bit much to have to restart the program for each trial. It would be better, don't you agree, to replace these lines with GOTO 20 ? We'll also need to add

```
100 GOTO 20
```

Try various different numbers and you should get a good idea of how the ON....GOTO.... works.

If you have more categories (and thus more line numbers to GOTO) than you can fit on one line, then you can adjust the variable suitably, and write more ON....GOTO.... statements.

eg. if you have 15 different classes that need separate treatment, you could take the first 8 with the first ON....GOTO.... you write and the remaining 7 with a second ON....GOTO.... line. BUT since this will start again with your variable at 1, you will need to subtract from it the number that you have already treated, in this case, 8. As in

```
30 ON NUM GOTO 100,200,300,400,500,600,700,800
32 ON NUM-8 GOTO 900,1000,1100,1200,1300,1400,1500
```

You may need to adjust your variable (for instance, add a small bit, multiply or divide it) to make sure that its value is such that the line that you want will be the one selected by the GOTO, but, as with most programming, if you work carefully you can achieve great things.

The other sort of sub-program comes into its own when a particular routine, or series of statements, is wanted in various different places in your programs. Let's say you're writing a game and you want to input angles as clock hours (eg. 12.0 means straight up). If you also want to use COS and SIN you'll have to convert all your angles to radians (pity about that). You could write a sub-program (often called a subroutine) to do the conversion, and then to use it you could have something like

```
:
100 INPUT"WARP DIRECTION (AS CLOCK HOUR)";CLOCK
110 GOSUB 1000
120 REM SUBROUTINE HAS CONVERTED TO RADIANS
:
:
200 INPUT"FIRE DIRECTION (AS CLOCK HOUR)";CLOCK
210 GOSUB 1000
220 REM ANGLE IS RAD
:
:
```

While the subroutine could be

```
1000 RAD=(3-CLOCK)*30*3.14159/180
1010 RETURN
```

The subroutine is called at lines 110 and 210 by the GOSUB statement, which is just like a GOTO, except that System 80 remembers where the GOSUB came from! Pretty neat, and very useful too!! When the subroutine has finished its work, it tells the computer with a RETURN statement. This RETURN line does exactly what the word says...it goes back to where it came from, or more precisely, to the line following the relevant GOSUB call. So after the GOSUB at line 110, the subroutine is executed, then the RETURN sends control back to line 120.

Exercise 20.
Write a small program to accept two options. One is debit, and the other credit. The sub-programs should input an amount and update a balance. Use a subroutine to do the amount input and to check that the amount is positive and less than $100. Request the input again if the limits are exceeded and show the balance after each transaction.

There is another statement that combines both the sub-program styles we have talked about, ie.

ON variable GOSUB line 1, line 2,....

Here the value of "variable" selects which subroutine will be called. Each of these subroutines must end with a RETURN statement, which will return the program to the statement following the ON...GOSUB...

This lets us change the earlier example into a really modular program. That is, fairly separate program modules can be written which are able to be considered by themselves. eg.

```
1Ø REM CONTROL MODULE
:
1ØØ INPUT "1 DR,2 CR,3 BAL,4 ADJ";OPT
11Ø ON OPT GOSUB 1ØØØ,2ØØØ,3ØØØ,4ØØØ
12Ø GOTO 1ØØ
:
:
1ØØØ REM DR MODULE
:
:
1999 RETURN
2ØØØ REM CR MODULE
:
:
2999 RETURN
:
:
```

(No, I'm still not filling in any details!)

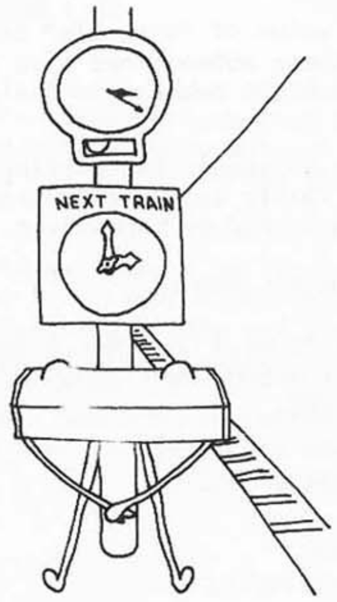
The RETURN statements in lines 1999, 2999 go back to line 12Ø which is just another statement telling the program where to go next. It sends System 80 back to line 1ØØ to get some more input.

Exercise 21.
Rewrite Exercise 2Ø as 4 modules; a header or control module, a DR module, a CR module and an input subroutine.

In Chapter 18

Commands	Statements	Other
	ON var. GOTO line 1.... GOSUB/RETURN ON var.GOSUB line 1../RETURN	modular programming

19. Fixed data



Sometimes a program needs to have a list of things that will stay the same for each run. These could be legal account numbers, star fleet base co-ordinates or areas under a normal curve(!) and so on.

The DATA statement provides this list. eg.

```
1Ø DATA 1,1,2,3,5,8,13
```

Then when you use the statement, READ variable , the next entry in the list is read (fancy that!) and assigned to the given variable. eg.

```
2Ø FOR C=1 TO 7
3Ø READ FIB
4Ø PRINT FIB
5Ø NEXT C
```

will list all the entries, one after another, from the above DATA statement. DATA statements may be anywhere in your program, even *after* the relevant READ statement. When you say READ, System 80 will look right through your program for the next DATA statements. So you could group them all at the end of your program (for easy addition of new DATA lines), or together at the beginning (for quick location in a listing) or throughout your program near the associated READ statements, and all the data will be read just the same. If, however, System 80 can't find a DATA statement or you've READ *all* the entries, you will get an ?OD ERROR message ie. out of data.

OK, you can get the DATA ... but only once! What if you want to use that same information again? Aha! You don't need to type out the same DATA statement again. You can get back to the start of your listed DATA by saying RESTORE. So if you add

```
6Ø RESTORE
7Ø GOTO 2Ø
```

this little program will run forever!!

Exercise 22.
Make a DATA statement of the following account numbers of overdrawn customers.
465, 591, 615, 988, 1ØØ5, 1213.
Then write a program to check that account numbers that are input are not in the red!.

Now for something really exciting (well, it turns me on anyhow). You can put text into DATA statements! You can have

```
10 DATA "ANDY","BRUCE","CHRIS","DAVE"
```

To READ these you need a *new* type of variable. This is called a *string variable*. The name of a string variable will look just like other variable names, except for a $ at the end. (No. Unfortunately it has nothing to do with money - $ is just the special marker for string variables.) So

```
        NAM$
        B4$
        OMD$
```

are legal string variables, while

```
        669$        (doesn't begin with A-Z)
        IF?$        (symbolic character, not A-Z or 0-9 in name)
        TOWN$       (reserved word TO in name)
```

are not legal string variables. Note too that NAM$ and NASTY$ will be taken as the same variable (because System 80 only notes the first two letters, remember?).

To print that last DATA statement's contents you can write

```
20 FOR N=1 TO 4
30 READ NAM$
40 PRINT NAM$
50 NEXT N
```

If you're beginning to think these string variables are similar to numeric variables, you're right. You can READ them, PRINT them and you can give them values. eg.

```
100 SURNAM$ ="JONES"
```

But note that we used a string variable (name$) and we are putting into it a string constant (characters enclosed by quotation marks) NOT a number. We *could* say

```
200 NUMB$="69"          (with quotation marks)
```

but NOT

```
200 NUMB$=69            (just a number)
```

If you try to equate strings and non-strings you will get a ?TM ERROR ie. type mismatch meaning that something is not a string. Anything you can type (even symbols) can be a string constant as long as it is contained within quotation marks. But, since these constants are text, not numeric values, we can't do arithmetic with them. However, using a "+" operator, we can stick them together, one after the other. This is called *concatenation*. eg.

```
110 FULLNAM$=NAM$ + SURNAM$
120 PRINT FULLNAM$
```

would give (assuming the previous program lines have just been done)

```
DAVEJONES          (no space)
```

If you don't like that, then

```
NAM$+" "+SURNAM$       is      DAVE JONES        or
SURNAM$+","+NAM$       is      JONES,DAVE
```

If you want a space printed, you need to include it yourself because System 80 doesn't automatically leave a space in front of text.

We can also input strings (and then they don't need to be enclosed by quotation marks). Try this game. (Remember to clean your slate first.)

```
1Ø REM CONVERSATION
2Ø PRINT"HI."
3Ø INPUT "WHAT'S YOUR NAME";NAM$
4Ø PRINT"HELLO ";NAM$;","
5Ø PRINT "I AM YOUR SYSTEM 80."
6Ø INPUT"HOW ARE YOU";HOW$
7Ø PRINT "WELL. I'M ";HOW$;" TOO!"
8Ø PRINT "BYE FOR NOW ";NAM$;"."
```

If you tried to be polite when you said how you were, and answered

```
VERY WELL, THANK YOU
```

System 80 would say to you

```
?EXTRA IGNORED
```

That's not an equally polite response is it? But this is only because System 80 would take the input as being two strings separated by a comma and so the second one (ie. THANK YOU) will be ignored. If the answer had been enclosed by a pair of quotation marks then it would have been accepted as one string. So you *can* use quotation marks in inputting strings but they are not usually needed and they are not included in the string. If you want to include quotation marks in a string, you can embed them in the text of an input string. eg.

```
I SAID "THANK YOU".
```

would be accepted in full.

If you have a very long name (and entered it all) and you were pretty talkative about your state of health, you may have got the message ?OS ERROR ie. out of string space. This will happen if you have used more than 5Ø characters (including spaces) in your total string inputs because System 80 only sets aside that much space for strings. However you can get over this limitation by allocating more string space yourself. eg.

```
CLEAR 1ØØ                    (a command)
```

will let you enter 1ØØ characters for this program. You can have as many as 255 characters in a string, though you will have needed to CLEAR that much space first, of course. If you use more than 255 characters (remember, that includes spaces and punctuation) you will get a ?LS ERROR ie. string too long.

We can also compare strings.

In the last chapter we said

```
100 INPUT"1 DR,2 CR,8 BAL,4 ADJ";OPT
```

Well, we can make that a bit fancier now! We can put the allowed (legal) options into a DATA statement, like

```
100 DATA "DR","CR","BAL","ADJ"
```

and then input the option in string form with

```
110 INPUT"OPTION";OPT$
```

Now we can check the input and deal with it using

```
120 RESTORE
130 FOR N=1 TO 4
140 READ RQ$
150 IF OPT$=RQ$ THEN ON N GOTO 1000,2000,3000,4000
160 NEXT N
170 PRINT"NOT A LEGAL OPTION"
180 GOTO 110
```

followed by the relevant subprograms.

Here, line 120 ensures that we start at the beginning of DATA,
line 130 starts a loop for the number of entries in DATA,
line 140 actually gets the next legal option (RQ$) out of DATA,
line 150 checks that the input option is legal. If so, then the loop counter, N, is the same as our old option input!
If the input is *not* in the list, then the loop ends after line 160 and lines 170 and 180 say that we're wrong and will try again.

Sure, this is longer, but it is pretty (as well as being more conducive to accuracy)!!

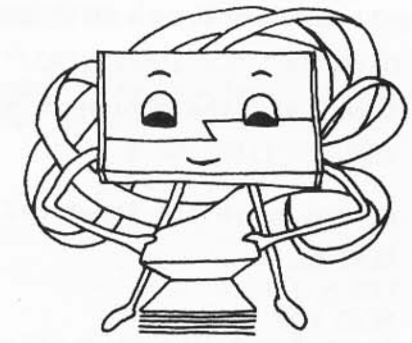
Exercise 23.
Create a little database (with DATA statements) with each entry as "surname","given name","address line 1","address line 2". You could use one DATA statement per entry. To mark the end use "surname" = "*". Write a program to input a surname and display either name and address or "not known".

In Chapter 19

Commands	Statements	Other
CLEAR string space	READ DATA RESTORE	string variables & ..$ string constants & ".." + as string concatenation

20.Compact style

You will have noticed that a lot of BASIC program lines are quite short. This means that even a fairly short program can end up as quite a few line numbers. Also, it doesn't take much program to go over one screen-full in a listing. Well, you *can* actually pack a program into a smaller package. (Oh good. I was beginning to wonder.)



You can write more than one statement on a line by putting a colon between them. For instance, try running

```
1Ø V=24Ø:PRINT V;"VOLTS"
```

You can put the whole delay loop on one line, ie.

```
2Ø FOR D=1 TO 345:NEXT D
```

Or, how about

```
3Ø FOR C=1 TO 2Ø:PRINT C,C[2:NEXT C
```

That's a whole table from one line!

It is certainly a good idea to put a group of related statements on a single line, especially if they are fairly simple - the delay loop is a good example. However if *everything* is just pushed together in long multi-statement lines it can get very difficult to follow what the program is trying to do! It will also make it hard to squeeze in additions (if you ever need them, of course).

There is another very valuable use for multiple statements - with an IF/THEN. Previously, if you had two things to do when an IF was satisfied, you had to GOTO somewhere, then GOTO back again - that's messy and sometimes difficult to follow in a listing. But NOW... If you write multiple statements after a THEN they will *all* be done if the test is true. If the test is *not* true then *none* of them will be done.

Try

```
1Ø INPUT A
2Ø IF A=1 THEN PRINT "AHA! A=";A: GOTO 1Ø
3Ø PRINT "THAT WASN'T 1.": GOTO 1Ø
```

Multiple statements can also be used in immediate mode! Try this line to get the total of numbers from 1 to 1Ø.

```
T=Ø:FOR N=1 TO 1Ø:T=T+N:NEXT N:?T
```

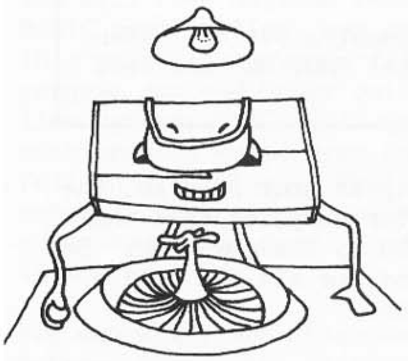
Exercise 24.
Write a program in as few lines as possible to input a string. Then, if your input is "YES" print "OK", otherwise print "NOT OK" and loop back for more.

Don't forget that when you include REM statements in your program, the letters REM at the beginning of the line tell your computer that what follows is for human use only, and can be ignored by the computer. So don't use REM to start any multiple program lines, or all the following instructions on that line won't be acted upon.

In Chapter 2Ø

Commands	Statements	Other
: and multiple statement commands	IF/THEN and :	: and multiple statement lines

21.The game generator



There is a function in System 80 specially designed for games (well...maybe some other things too). It allows you to get a random number, that is, a "surprise" number - one that doesn't depend on anything else! With this you could

- make arithmetic quizzes
- invent guessing games
- add randomness to games (like a cross-wind for golf?)

Whenever you use RND(X) you will get a new random whole number between 1 and X.

Try this little game.

```
1Ø REM NUMBER GUESSER
2Ø N=RND(1ØØ)
3Ø PRINT "I'VE GOT A NUMBER BETWEEN 1 AND 1ØØ."
4Ø INPUT "WHAT'S YOUR GUESS";G
5Ø IF G<N THEN PRINT "TOO LITTLE!": GOTO 4Ø
6Ø IF G>N THEN PRINT "TOO BIG!": GOTO 4Ø
7Ø PRINT "OH RATS, YOU GOT IT!"
```

(See how multiple statement lines can make your program neater when you use IF/THEN ?!!)

RND(X) will give you random numbers that are integral values but by saying RND(Ø) you can generate random numbers as fractions that are bigger than Ø and less than 1.

What, you might ask, do we get for negative arguments of RND? An ?FC ERROR!! Not so useful. You will also get an ?FC ERROR if the argument that you have chosen for RND is greater than 32768.

To be truthful about it, RND really generates "pseudo random" numbers. Whenever you turn System 80 on, if you use RND you will get the same sequence of "random" numbers. However you can get a *new* sequence by using the RANDOM statement.

So	RND(X)	returns an integer between 1 and X inclusive. X must be positive and >=1 but not more than 32768.
	RND(Ø)	returns a fraction between Ø and 1.
	RANDOM	sets up a new pseudo-random sequence.

Exercise 25.
Write a program to throw two dice.

Exercise 26.

Write a program to draw one playing card from a standard deck.

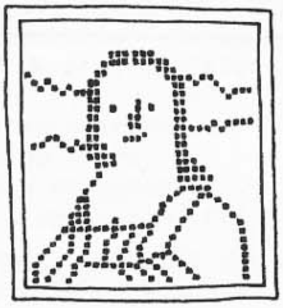
In Chapter 21

Commands	Statements	Other

	RANDOM	RND(Ø)
		RND(X)

22.Pictures

Yes, pictures! Well, sort of anyway! We've already done some graphical things with a sine curve back in Chapter 14 and our rockets in Chapter 15. Graphics, even quite simple ones, really make a program great! As the man said, "A picture is worth a thousand words" - even if it takes a thousand instructions to create it!



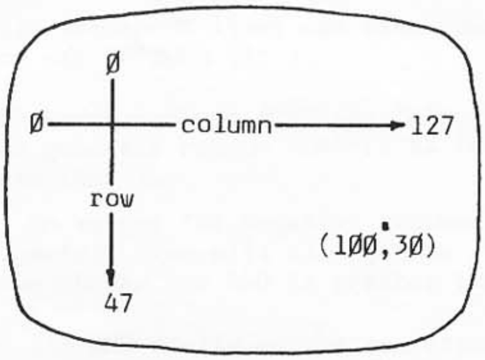
A slight note of discord? Well, yes. To make a picture the program must say where every dot of it goes. And yes, that can make quite long programs, and even rather tedious programming to get it all right.

But, ah,... the finished product!

About those "dots". System 80 has 6,144 of them!! To make it a bit easier they are set out like graph paper: 128 dots wide and 48 dots high. (That sounds a bit more manageable!) They are set out as

Ø to 127 from left to right and
Ø to 47 from top to bottom.

You can refer to any of these dot positions as (c,r) where c is the column number and r the row number as shown:



However, before we start making pictures, we should have a clear screen The statement CLS will do just that. Try using it as a command and type

```
CLS
```

Presto! (I suppose we *had* to get our ready display back.)
CLS is similar to the type-in NEW in that it empties the screen and puts the cursor at the top BUT the important difference is that CLS will *not* clear all your program and variables too.

OK, the instruction to "paint" a dot is SET(column,row).
Try this out.

```
1Ø REM GRAPHICS SAMPLER
2Ø INPUT"CHOOSE YOUR COLUMN,ROW";C,R
3Ø CLS
4Ø SET(C,R)
5Ø FOR X=1 TO 345:NEXT X
6Ø GOTO 2Ø
```

You'll be asked to input a column number and a row number, then the screen will be cleared and you'll have a second to see your point set on a clear screen *UNLESS* the numbers you choose are outside the range allowable for graphics. You'll get an ?FC ERROR if you choose a column number less than Ø or greater than 127 OR if you have a row number less than Ø or greater than 47.

You can see that the point (Ø,Ø) is right up in the top left-hand corner and the point (127,47) is at the extreme bottom right-hand corner.

Did you try using numbers with fractions? The numbers you choose don't *have* to be integral, but this function will only use the whole-number part.

The statement to clear a dot is RESET(column,row) with the same conditions on the column and row numbers. Add this to your program.

```
45 RESET(C,R)
47 FOR X=1 TO 1ØØ:NEXT X
```

Or you could try this one

```
1Ø REM ANOTHER GRAPHICS SAMPLER
2Ø CLS
3Ø SET(2Ø,2Ø)
4Ø FOR X=1 TO 1ØØ:NEXT X
5Ø RESET(2Ø,2Ø)
6Ø SET(3Ø,2Ø)
7Ø FOR X=1 TO 1ØØ:NEXT X
8Ø RESET(3Ø,2Ø)
9Ø GOTO 3Ø
```

and System 80 will blink at you.

Exercise 27.
Write a program to fill in the whole screen.
Add some more program to randomly clear points in this. (Termites!)

Did you know that computers were artistic? Well, try out this little doodler.

```
1Ø REM KALEIDOSCOPE
2Ø CLS
3Ø XI=RND(64)-1:YI=RND(24)-1
4Ø IF RND(1Ø)>1 THEN GOTO 8Ø
5Ø SET(XI,YI):SET(127-XI,YI)
6Ø SET(XI,47-YI):SET(127-XI,47-YI)
7Ø GOTO 3Ø
8Ø RESET(XI,YI):RESET(127-XI,YI)
```

```
90 RESET(XI,47-YI):RESET(127-XI,47-YI)
100 GOTO 30
```

Line 30 chooses the ($\frac{1}{4}$ screen) point location.
Line 40 decides whether to turn it on or not so that the screen does not eventually fill completely. This is a random decision (made by choosing a random number between 1 and 10) and changing the value "1" will change the "darkness" of the doodle by allowing a greater proportion of points to be turned on.

Exercise 28.
Snail Tracks! Start on the screen and move a snail track randomly about. Each move he can go -1, 0 or +1 in X or Y from where he is, but not outside his box.

There is another function that doesn't do *anything* to your picture. It just lets your program check what it has already done! POINT(column,row) returns a value of -1 if the dot is set, or 0 if it is reset (clear). eg.

```
... IF POINT(10,50)=-1 THEN PRINT "ON"
```

tests the dot at column 10 and row 50.

Exercise 29.
Change Exercise 28 so that your snail never crosses his own trail! How far can he get?

In Chapter 22

Commands	Statements	Other

	CLS
	SET(column,row)
	reset(column,row)
		POINT(column,row)

23. Tables

Some time ago we found out how to display tables on the screen. That was useful,... and pretty. In fact, it was so useful that it would be good to keep those tables inside the computer to look at again. Let's do it.

How about a table of quantities? This could be numbers of boots sold, current levels of starship resources, or costs of cigars, etc. etc... Say we want up to 5Ø entries, System 80 must be told this so that space can be provided. This is done with a statement like

```
1Ø DIM QUANT(5Ø)
```

ie. the DIMension, or size, of our table called QUANT is 5Ø entries. The table QUANT is also called an array. QUANT is a lot like our usual variables, except that it comes in 5Ø flavours! (or at least, in this case it does) To pick the entry of QUANT that we want to refer to we need to specify a subscript. This is a number (or variable, or calculation) from Ø to 5Ø. So to refer to QUANT entry number 25 we use QUANT(25)

We can say things like

```
        QUANT(25)=365.25
        PRINT QUANT(25)        etc.
```

Try out this program.

```
1Ø DIM QUANT(5Ø)
2Ø INPUT"ENTRY";E
3Ø INPUT"VALUE";QUANT(E)
4Ø FOR E=Ø TO 5Ø:PRINT QUANT(E);:NEXT E:PRINT
5Ø GOTO 2Ø
```

You should have seen that the value you entered appeared in the place that you chose for it in the table. If you were to try "ENTRY" outside the Ø to 5Ø range you would get an error when System 80 tried to use the array. Either

```
?BS ERROR   -   bad subscript         or
?FC ERROR   -   really bad subscript!!
```

Exercise 3Ø.
Change the above example to only print non-zero entries. It should also print the entry number.

You'll notice that I've been lying (well, a bit misleading anyway). If we give an array a dimension of 5Ø, then we can actually have 51 entries because we can include the subscript Ø.

With an array it's easy to add up all the elements. Add something like

```
5Ø REM TOTAL QUANT
6Ø TQ=Ø
7Ø FOR E=Ø TO 5Ø:TQ=TQ+QUANT(E):NEXT E
8Ø PRINT"TOTAL";TQ
9Ø GOTO 2Ø
```

Line 5Ø replaces the old GOTO loop (which we put back in line 9Ø). Line 6Ø initialises the total so that in line 7Ø we can just add to it.

OK, that's one away. Let's try some more.

```
1Ø DIM NAM$(6), V(6), Q(6)
```

That's interesting! A *string* array! We can enter values with

```
2Ø FOR E=Ø TO 6
3Ø INPUT "NAME,UNIT COST,QUANTITY";NAM$(E),V(E),Q(E)
4Ø NEXT E
```

Next, we can display this with

```
5Ø SALES=Ø
6Ø PRINT "ITEM","PRICE"."QUANTITY","SALES"
7Ø FOR E=Ø TO 6
8Ø PRINT NAM$(E),V(E),Q(E),V(E)*Q(E)
9Ø SALES=SALES+V(E)*Q(E)
1ØØ NEXT E
11Ø PRINT
12Ø PRINT "TOTAL SALES",,,SALES
```

How's that? (Did you notice how the three commas in line 12Ø spaced the printout?) We have 3 arrays all running in parallel. The same element in each array describes a different characteristic of the same thing!

Let's say we want to keep sales figures for each item in each month. We could say

```
1Ø DIM SALES(6,12)
```

This lets us specify which item we want and also what month of the year we want too! Thus SALES(1,6) could be the June sales for item #1.

Another two dimensional array (sometimes called a matrix) could hold an object's position in space. eg.

```
... DIM XYZ(1Ø,3)
```

```
for 1Ø objects with XYZ(obj,1) the X co-ordinate,
                    XYZ(obj,2) the Y co-ordinate
               and  XYZ(obj,3) the Z co-ordinate
```

When you're debugging your programs you can get a display of (dump out) the contents of array NAM$ (for instance) with the command

```
FOR E=Ø TO 6:?NAM$(E):NEXT E
```

or XYZ with

```
FOR O=ØTO1Ø:FOR D=1TO3:?XYZ(O,D);:NEXT D:?:NEXT O
```

(Wow! That's quite a one-liner.)

Whenever you RUN a program, System 80 will clear all your variables (and arrays) to zero. You can also clean off your own slate with the statement

```
... CLEAR
```

At this time System 80 also sets aside the amount of space you can use for all your string arrays and variables. The space that is reserved for strings (ie. 5Ø characters - see Chapter 19) is not much if you're using arrays. To allocate more string space you can include at the start of your program

```
... CLEAR number of string spaces
```

If you try to use too much memory space, eg. with DIM BIG(1ØØØØ), you will get ?OM ERROR ie. out of memory. There are obviously limits to what you can do because everything uses up space in memory, program lines, variables, arrays both numeric and string ..etc... but you'll have to write a fair bit before you run out of space.

If you want to know how much memory you still have left you can use the command (always with PRINT)

```
PRINT MEM
```

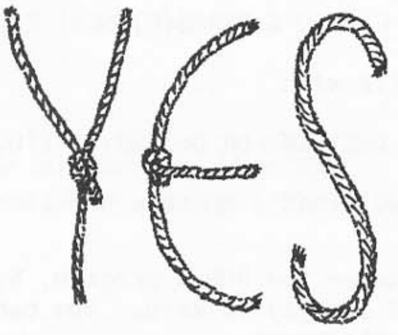
In Chapter 23

Commands	Statements	Other
PRINT MEM	DIM CLEAR CLEAR no. string spaces	arrays matrices subscripts

24. Did he say "YES"?

We saw in Chapter 19 how you can get string answers to questions. eg.

```
1Ø INPUT"QUESTION";ANS$
2Ø IF ANS$="YES" THEN PRINT "OK"
:
etc.
```



It would be nice to be able to recognise type-in of "YES" *or* "Y". Certainly we could make more than one IF, but that's messy. The function

```
    LEFT$(string,number)
```

gives a string of the leftmost given number of digits from the given string argument. So you could say

```
2Ø IF LEFT$(ANS$,1)="Y" THEN PRINT "OK"
```

and both "YES" and "Y" will work (and so will "YUCK" or "YNO" etc.). Similarly you could select the right-hand 3 characters with

```
    RIGHT$(ANS$,3)
```

Another thing you can do is to find out how long a string actually is. Use `LEN(string)`

> Exercise 31.
> If NAM$="SKYWALKER" can you give the results of ..?
> 1. LEN(NAM$)
> 2. LEFT$(NAM$,3)
> 3. RIGHT$(NAM$,6)
> 4. RIGHT$(NAM$,LEN(NAM$))
>
> Use System 80 to *check* your answers *only*.

You might have noticed that, as could have been expected, *functions* with names ending in "$" return strings, while those without "$" at the end return numbers.

There's another function here that acts a bit like a combination of LEFT$ and RIGHT$, and it is MID$.

```
    MID$(string,start,number)
```

returns a string "number" characters long, starting from the character "start" in "string". (The first character is 1 - not Ø.) So

```
    MID$("SKYWALKER",4,3)          is  "WAL"
```

If you ask for more characters than the string has, MID$ will give you as many as it can. You can leave out the third argument of MID$ and you will get all the rightmost characters of the string, starting with the character that you have specified. (Where have I seen that result before?) Try this little program to analyse typed input.

```
1Ø INPUT"STRING";S$
2Ø FOR C=1 TO LEN(S$)
3Ø PRINT"CHARACTER=";MID$(S$,C,1)
4Ø NEXT C
5Ø GOTO 1Ø
```

Now we've got strings.....and numbers..... and never the twain shall meet. But they do -

STR$(number)	allows the given number to be handled as a string and
VAL(string)	gives the numerical value of the given string. If you don't have a number in your string you'll just get zero, or if a number is followed by other characters you'll just get the number.

For instance, what would you expect to be printed if you said...?

```
PRINT VAL("23")+1,STR$(23)+"1"
```

Did you say 24   231   ?
Well, congratulations! Of course you did.

Exercise 32.
Write a calculator. It should accept;

```
ENTER number
ADD number
SUB number
```

(and others if you like).

(Hint - use a space to indicate the start of the number, and MID$ or RIGHT$ to get the number out for VAL to use!)

In Chapter 24

Commands	Statements	Other
		LEFT$(string,no.) RIGHT$(string,no.) MID$(string,pos,no.) LEN(string) VAL(string) STR$(number)

25. Conditions

Often the conditions you want to test for in your programs using an IF...THEN.. statement will involve more than a single check. You may want the computer to work out whether a number of things has occurred, all at the same time. For instance, if you want to go sailing you should;

```
        be able to swim
AND     be able to get a boat
AND     have a fine day.
```

You *don't* have to use more than one IF line to do this. Just as in English you can use the operator AND to make one statement do the job. So you could tell System 80

```
...IF SWIM$="YES"AND BOAT$="GOT"AND DAY$="FINE"THEN SAIL$="YES"
```

Here the string variable SWIM$ must have the value YES
AND BOAT$ must have the value GOT
AND DAY$ must have the value FINE
before SAIL$ is given the value YES.

If you might go boating in other circumstances (for instance if your plane ditched!) you could add

```
...IF SAIL$="YES" OR PLANE$="DITCH" THEN PRINT "GO SAILING"
```

Here the OR operator is being used, instead of AND, so that you may be
EITHER going sailing for fun
OR sailing of necessity.

The message GO SAILING will be printed if
EITHER the string SAIL$ has the value YES
OR the string PLANE$ has the value DITCH.

(System 80 won't insist that it's a fine day before you can save yourself from the plane.)

THUS in an IF statement
condition AND condition is true ONLY if both conditions are true
condition OR condition is true if EITHER condition is true.

You can even add to this by using parentheses to make "compound" conditions. Try this to get the feel of it.

```
10 INPUT "A,B,C";A,B,C
20 IF (A=1 AND B=2) OR C=0 THEN PRINT "YEP":GOTO 10
30 PRINT "NOPE":GOTO 10
```

So the AND and OR operators let you make *multi-barrelled* decisions with a *single* IF...THEN... statement which you can make more complex with parentheses.

As well as being able to do something under complex conditions, you can *instead* take different action if it is *not* true. You can do this just by adding ELSE... to your IF. eg. you could say

```
... IF A=1 THEN PRINT "YEP" ELSE PRINT "NOPE"
```

which is just the same as

```
... IF A=1 THEN PRINT "YEP"
... IF A<>1 THEN PRINT "NOPE"
```

only a lot prettier!

And.. you can still put multiple statements in! So you could have

```
1Ø INPUT A
2Ø IF A=1 THEN PRINT"YEP":GOTO 1Ø ELSE PRINT"NOPE":GOTO 1Ø
3Ø STOP
```

which will *never* get to the STOP!

Try replacing line 2Ø above with

```
2Ø IF NOT A=1 THEN PRINT"YEP":GOTO 1Ø ELSE PRINT"NOPE":GOTO 1Ø
```

This completely turns our test around!

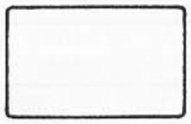
You can also use NOT to test for the untrue value of an extended set of conditions in parentheses. For instance, if we went back to our earlier example and said

```
2Ø IF NOT (A=1 AND B=1 AND C=1) THEN PRINT"YEP":GOTO 1Ø
```

then all three values would need to be NOT equal to 1 for "YEP" to be printed.

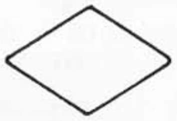
So you can certainly cover all contingencies and still with *only one* IF...THEN... statement! This is starting to make things complicated! (starting?) There are some program design methods to help the poor old programmer exercise his art. One of these is a graphical method called flowcharting. It is just some specially shaped boxes to break up a program into manageable pieces, and diagramatically show its "shape".

Flowcharting uses



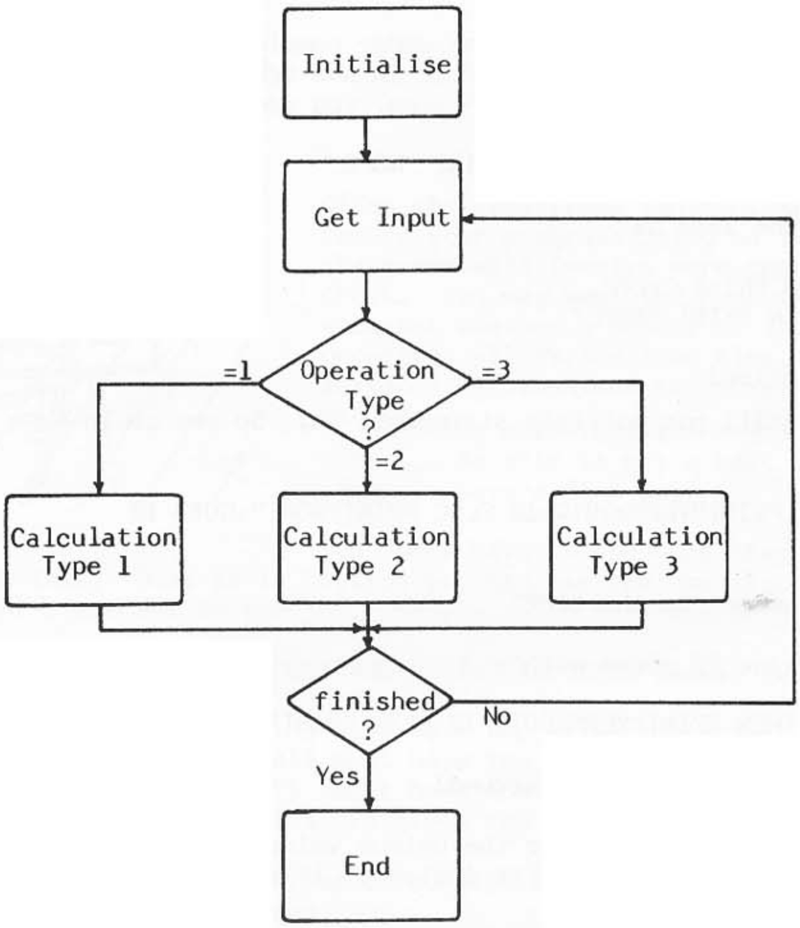
for calculations

and



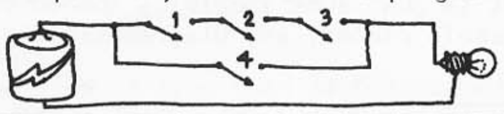
for decisions

and lines to indicate program flow. (Some people use more symbols, but two is enough!) For instance, a typical program could look like



Flowcharts can be particularly useful when working out what you want to do with a program that will be using the IF...THEN...statement or IF...THEN...ELSE... with the AND,OR and NOT operators to give various different treatments.

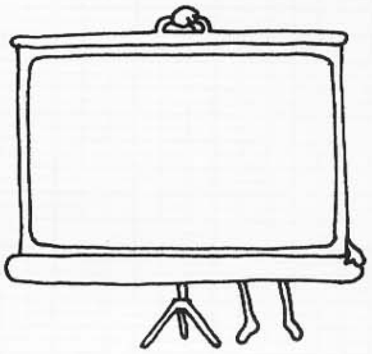
Exercise 33.
We want to simulate the behaviour of this light bulb with a program. It must input the states of the switches, then display what happens. First flowchart the problem, then write a program from your flowchart.



In Chapter 25

Commands	Statements	Other
	IF..THEN..ELSE..	AND OR in IF statements NOT logical parentheses flowcharting

26. Screen control

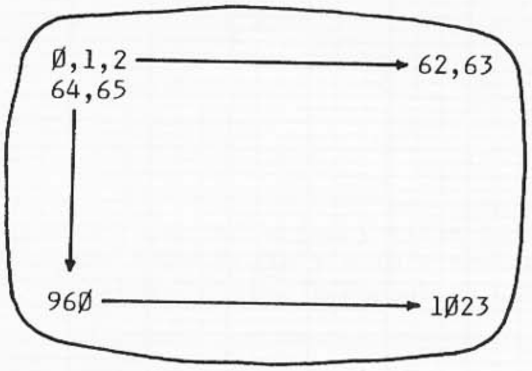


You can display numbers and things on the screen and make simple pictures, but imagine putting them together! Actually, as you'll see it's not very hard, but it *always* needs careful planning.

A print can be made to start *anywhere* on the screen by saying

```
... PRINT @n,things to print
```

where "n" is the character position set out as



Try

```
PRINT @ Ø,"HI"
```

You'll see HI at the top left, with the ready display on the next line (probably over the top of old screen contents). It might look a bit confusing, but you can put that message wherever you want it!

Next comes the really exciting bit - combining graphics with text. First you must know how System 80 makes its graphics. It divides each character position into 6 dots like ▦

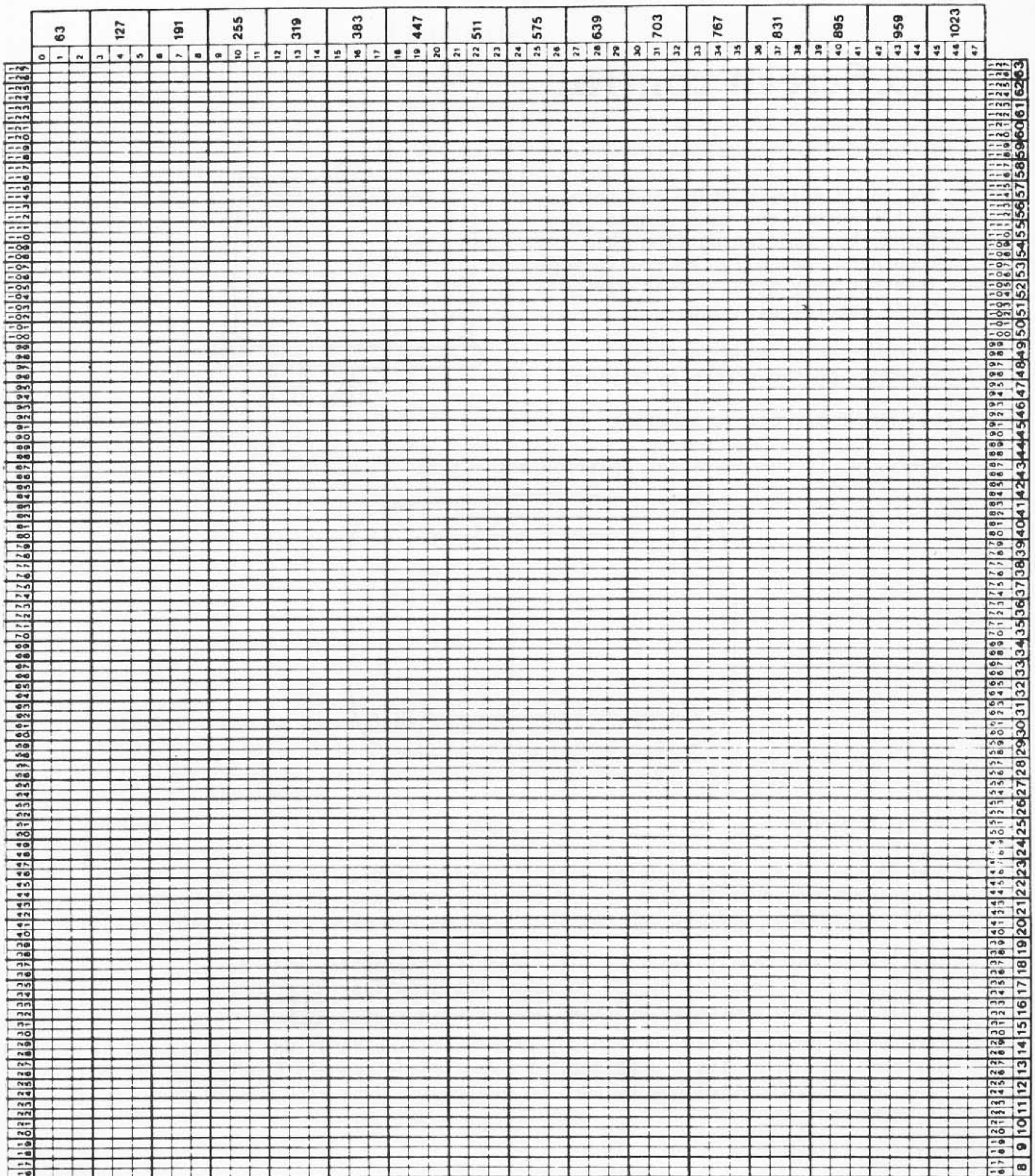
Each 6-dot position may contain any character(like "A") OR have any combination of its 6 dots set. We've already seen in Chapter 22 (the GRAPHICS Chapter) how to turn on one of the dots with SET(col,row) If we said

```
SET(Ø,Ø)
```

we would turn on the top left dot of character position Ø. The column and row numbers used in the instructions SET, RESET and POINT make up the dots of character position Ø like this

(Ø,Ø)	(1,Ø)
(Ø,1)	(1,1)
(Ø,2)	(1,2)

Video Display Map

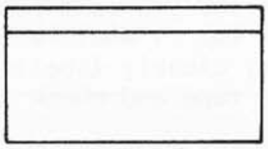


The accompanying VIDEO DISPLAY MAP gives the dot/character positions for *all* the screen.

Now for some action! This example plots a box then puts a message inside it. Pretty hot stuff!

```
10 REM MAKE A BOX
20 CLS
30 FOR X=20 TO 80
40 SET(X,10):SET(X,30)
50 NEXT X
60 FOR Y=10 TO 30
70 SET(20,Y):SET(80,Y)
80 NEXT Y
90 PRINT @ 384+18,"THIS IS A BOX";
100 REM LINE 110 STOPS PROMPT APPEARING
110 GOTO 110
```

Exercise 34.
Write a program to lay out a box like



the size of the screen. Next put a fancy title in the box at the top. (You'll have to end your PRINT @ lines with a semi-colon to keep the right of your box intact!) (Don't forget to CLS first.)

Exercise 35.
Write a program to draw an ace of hearts.

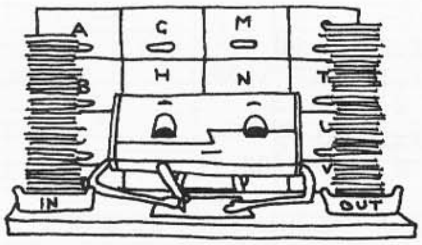
Exercise 36.
Input 5 product names and sales quantities and make a bar chart of their sales.

In Chapter 26

Commands	Statements	Other
	PRINT @ n,....	display control

27.Changing data

It's all very well to have DATA statements to store information in a program, but what if that DATA changes frequently?? Do we have to keep changing our DATA statements every time there is a change? No, of course not. It is possible to keep your data on cassette and up date it when needed.



But before we go on, a word of warning.
So that you're not completely wiped out if your machine hiccups, as may happen, you should at least keep multiple copies of cassette data. If you're updating its contents don't write out to the *same* tape. Use a different one and keep the one you used for input so that you can regenerate files if you need to. I would strongly suggest keeping each file on a separate tape, clearly labelled!! It might well be a good idea too, to re-read a tape and check it against what you expected to have written.

So *BE CAREFUL!* and test out your work *thoroughly* before filing all your company records in the rubbish bin.

Well, now that we're ready, I guess I had better tell you how to do it! You need to first of all position your tape as in the CSAVE operation. Make sure the tape is past its leader and set the recorder to record (with F1 up).

Now for the transfer of data! This is done with a PRINT statement, but since you don't want your data displayed on the screen but printed onto the tape instead the instruction is modified thus;

```
... PRINT#-1,list of things to be recorded
```

To try out the procedure you could say

```
1Ø REM CASSETTE DATA SAVE
2Ø INPUT"PREPARE CASSETTE FOR RECORDING & PRESS NEW LINE";GO$
3Ø PRINT#-1,"I'M SYSTEM",80
4Ø PRINT"OK,YOU CAN STOP THE CASSETTE NOW!"
```

That GO$ in line 2Ø is just a dummy to make INPUT wait till you are ready to go to the next line.

Don't make your PRINT#-1 lines too long because anything after 255 characters will be lost!!

Sooner or later you'll probably want to read your cassette data back in (well, I hope so anyway!)

Set the cassette deck to play (F1 up) and this time the statement is like INPUT (reasonable, that) except that, because the source of the data is a cassette, not you, the statement is again modified like this

```
... INPUT#-1,list of variables for data to be read into
```

A bit of care is needed here though. The list of variables in INPUT#-1 *must* match in type and number what is on the tape. If they *don't*, or there is an input error, you'll get an ?FD ERROR - file data error. However the *names* of the variables need not match as long as numeric variables are *still* numeric and string variables *remain* strings.

To get back the data that we just recorded we could write

```
10 REM CASSETTE DATA INPUT
20 INPUT"PREPARE CASSETTE FOR INPUT AND PRESS NEW LINE";GO$
30  INPUT#-1,NAM$,NUMB
40 PRINT"I FOUND ";NAM$;" & ";NUMB
50 PRINT"YOU CAN STOP THE CASSETTE NOW."
```

The tape should stop when the data has been read in (but if it doesn't, just press RESET, at the left back, and try again).

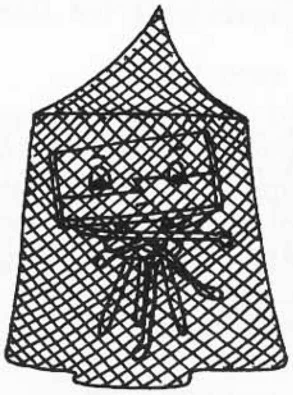
Exercise 37.
Request that the user type his name and address into a string array. Save this on cassette and display it.
(Don't forget to tell yourself what to do with the recorder!)

These operations could be used for mailing lists, library catalogues, kitchen recipes, family genealogy or a million other things. It can be very useful in playing games on the computer to be able to record things like the current status of Space Affairs for later use in a long game of Space Trader. BUT please be careful. It could be very easy to lose your valuable lists!

In Chapter 27

Commands	Statements	Other
	PRINT#-1, list INPUT#-1, list	

28. Antibugging



This is more a frame of mind than a thorough methodology. It is an approach to program design that assumes that you are likely to make some mistakes or omissions and so will take appropriate precautions. It is something like putting ant-caps on your house's foundations - it reduces the number of places that bugs can get in.

The first thing to do is to actually sit down and design your program. It's fine to type a straightforward program straight from your imagination into System 80's memory, but there comes a stage when it's hard to hold all the facets of a program in your consciousness at one time.

We've already talked a bit about program design with flowcharts. They are certainly good for nutting out complex logical situations, however some people find them less well suited for overall design.

One good approach to total program design is called "top-down" design. Essentially this means to program (maybe even in plain old ordinary English) the most broad outline first. eg.

```
           ┌ Set-up
           │
Program   <  Handle requests
           │
           └ End
```

This is the top level. Next, each of these steps is refined. For instance, if this program were a simulation of a space battle,

```
           ┌ Set-up                  { clear display
           │                         { generate positions
           │
           │  Handle requests        ┌ get request
Program   <  (repeat while enemy    <  process request
           │    still exists)        └ end request
           │
           │
           └ End
```

You can go on refining until the required program instructions are obvious. If you run out of room, just start a new page for the sub-classes eg.

```
                  ⎧ if "map"     { display positions
                  ⎪
                  ⎪ if "warp"    { request details
                  ⎪                move ship
Process request  ⎨
                  ⎪ if "fire"    { request details
                  ⎪                track weapon
                  ⎪
                  ⎩ if "status"  { list status
```

This gives you an overall understanding of your program, and you can see at what level it is most appropriate to break into separate blocks of instructions. (A rough guide is that a block, or "module", should fit into a single screen full.)

Another level of "anti-bugging" comes into the picture when you are actually writing the program. If a bit of program is especially complicated, or contains lots of calculations, it is a good idea to spread it out a bit. Perhaps you could even show some intermediate values. If a program is not squashed up, using all available line numbers for that section, then there will be space for extra debugging PRINTs to be put in if all is not well. It also allows some extra space if it is necessary to add further operations.

Again, if the program is getting particularly tortuous, it might be time to sit back and see if there is a different way to do it all more simply. The simplest solution is *always* the best especially if the program ever has to be changed!

Well, what a boring old load of pontifical *censored* that was! Anyhow, it's just common sense. I guess you've got to have a low point somewhere.

29. END

Here are a few extra bits of fairly esoteric information that might be of interest.

There are some statements that you can use as commands and some commands that will work as statements in a program.

RUN and LIST can both be acted on in a program (though they're not very useful there since RUN will cause a program to go forever and LIST will prevent it from being run at all by just giving a listing).

GOTO line number
can be used as a command to cause a program to be run from the given line number *without* your variables being cleared first (as would happen with the command RUN).

However, there are also some statements that *cannot* be used as commands, such as INPUT, and when they are tried you will get ?ID ERROR ie. illegal in direct mode or this can't be done by itself.

If you want to use BASIC programs from other computers (or want to shorten your own work) you might like to know that System 80 accepts two other forms of IF condition THEN GOTO line number

They are IF condition GOTO line number

and IF condition THEN line number

That's all for basic BASIC! However, as you'll see when you look at the System 80 Basic Manual there are *lots* more goodies (though you might have to look a long way before finding a use for some of them!)

Good programming!

Glossary

argument; either a heated discussion or more hopefully just a value, numeric or string, given for a function to work with.

array; a collection of related items grouped under one variable name.

ASCII; American Standard Code for Information Interchange. This is a code for computing characters.

BASIC; B..... Antagonising Sometimes but Ingenious Code. Actually, BASIC is the computing language that System 80 uses and stands for Beginners' All-purpose Symbolic Instruction Code.

binary; system of representing numbers with only two different digits. eg. ON or OFF for a computer's electronics.

bit; binary digit - smallest unit of computer's Memory(see memory)

byte; group of 8 bits - smallest program accessible unit of the computer's Memory. Each byte may contain part of an instruction, part of a number to be used or the code for a character (see memory)

bug; program error

command; instruction which can be immediately acted on.

computer; ingenious electronic device for consuming time.

constant; fixed value, numeric or string, used in a program.

cursor; not a user of strong language but a mark (underline in System 80) indicating the position the next character will take.

dump; display of information, some of which might even be useful!

E; exponentiation factor followed by a power of 1Ø.

execute; carry out instructions

field; subdivision of a section of display, records etc.

function; operation on an argument to give a new result.

hardware; nuts, bolts, transistors and integrated circuits in computer.

hexadecimal; non-decimal system of number representation - base 16 Digits are Ø to 9 and A to F OR Ø to 15

home; starting position on computer's display, top left-hand corner.

initialise; set to starting value

K; represents 1ØØØ in normal parlance, but for computing it usually represents $2^{1Ø}$ ie. 1Ø24

line numbers; necessary beginning of program line.

machine language; computer's own representation of instructions for hardware to execute.

matrix; two dimensional array

memory; this is where the computer stores all its instructions, values etc. It has two parts, a user-accessible section where things like the program lines and input are stored (Random Access Memory or RAM) and a place where all the permanent information for the computer, like how to decipher BASIC, is stored in a form not generally accessible to the user (Read Only Memory or ROM). Memory is made up of lots of bits. Each bit (or binary digit) is either on or off ('cause that's easy to do with electronics). These bits are organised into lots of groups of 8 called bytes, and this is the smallest unit of memory that the computer can do programming things with. An 8K computer has about 8ØØØ bytes of memory to program in (actually, in computing "K" often means an equivalent in binary terms ie. $2^{1Ø}$ or 1Ø24, so 8K is 8192 bytes!)

nested loops; FOR/NEXT loop containing more FOR/NEXT loops. The innermost one will be completed for each pass through any loop containing it.

peripheral; hardware allowing the computer to communicate with the external world.

program documentation; vital to programmers. Explanation of purpose of program as in REM.

prompt; character(s) on computer's screen indicating its readiness to accept input - usually precedes cursor on the screen.

RAM; Random Access Memory, usually implies READ/WRITE storage space.

reserved words; letter groups set aside in BASIC for use only in the defined instructions. Their appearance elsewhere will wreak havoc.

ROM; Read Only Memory, permanent information in System 80.

scientific notation; representation of numbers using a power of 1Ø.

software; programs provided to make System 80 work.

statement; component of program. Line of instruction to be executed when program is run.

variable; name of an entity which can hold different values.

zone; fixed field eg. System 80 has 4 zones across the display screen.

Appendix A. Summary of BASIC

Here is a summary of System 80's Level II BASIC:

1. COMMANDS

AUTO
: automatic line numbering by steps of 1Ø from line 1Ø

AUTO line number, increment
: automatic line numbering by chosen steps from given line number

CLEAR
: set program variables to zero

CLEAR n
: clear variables and "n" string-space bytes

CLOAD "name"
: load cassette file "name" into memory

CLOAD? "name"
: cassette program check

CLS
: clear screen

CONT
: after $\boxed{\text{BREAK}}$ or STOP to continue

CSAVE "name"
: save program on cassette as file "name"

EDIT line number
: select line to edit, then use
: space
: backspace
: $\boxed{\text{S}}$ c — search for character "c"
: $\boxed{\text{D}}$ — delete next character
: $\boxed{\text{I}}$text $\boxed{\text{SHIFT}}\boxed{\text{ESC}}$ — insert text
: $\boxed{\text{H}}$text — delete remainder and insert text
: $\boxed{\text{C}}$ c — change next character to "c"
: $\boxed{\text{NEW LINE}}$ — finish edit

LIST
: program list

LIST line number-line number
: list part of program

NEW
: remove current program from memory

PRINT MEM
: display amount of free memory

RUN
: execute program

RUN line number
: execute from given line number

TRON
TROFF
: line number trace display on or off

2. PROGRAM ELEMENTS

Each program consists of a series of lines. Each line is

line number statement

OR

line number statement:statement:....

up to 255 characters (including spaces). Spaces may be left out, so care must be taken to avoid reserved words as these will result in unexpected errors.

MAX. No OF LINE Nos ∅ - 65,529

Variable names;
must start with a letter and the rest can only be made up of letters and numbers. System 80 will only take note of the first two characters.

String variable names;
must also have a "$" at the end.

Numeric constants;
will be ±number
±number E±number
up to 1E38 (normal precision correct to 6 digits)

String constants;
must normally be enclosed by quotes ie"string"
(but quotes are optional for string input only)
up to 255 characters long (including spaces)

Numeric operators;
+ addition
- subtraction or unary minus
* multiplication
/ division
[exponentiation

String operator;
+ concatenation

Order of precedence;

```
first..() parentheses
       [  exponentiation
       -  negation (unary minus)
       */ multiplication and division, left to right
last.. +- addition and subtraction, left to right
```

3. STATEMENTS

CLEAR n
clear variables and "n" string-space bytes

CLS
clear screen

DATA list
data for READ statement

DIM name (dimension list)
allocate array space

END
    terminate program execution

FOR var=start TO end STEP change
    set "var" to "start" and execute till NEXT statement

GOSUB line number
    unconditional branch to subroutine - RETURN at subroutine end

GOTO line number
    unconditional branch to line number

IF condition THEN line number
IF condition GOTO line number
IF condition THEN GOTO line number
    conditional branch to line number

IF condition THEN statement
    conditional execution of "statement"
IF condition THEN statement 1 ELSE statement 2
    execute statement 1 if condition is true
    execute statement 2 if condition is false
        condition is; expression relation expression
        relation is; <, <=, =, >=, >, <>
        operate on conditions with AND, OR, NOT, ()

INPUT "text";variable list
    display "text" and accept type-in to variables

INPUT#-1, variable list
    transfer data from cassette to the variables listed

LET variable=expression
    assign value of "expression" to "variable"
    "LET" is optional

NEXT variable
    end of FOR/NEXT loop

ON variable GOSUB line number list
    "variable" selects a "line number" (from list) of subroutines
    to go to

ON variable GOTO line number list
    "variable" selects a "line number" to GOTO

PRINT expression list
    to display
    Use semi-colon to compact items or comma for zones.
    Use TAB(col) to lay out print.
    PRINT with no following text spaces to a new line.
    ? can be used as shorthand for PRINT.

PRINT @ number, expression list
    display at character position given by "number"

PRINT#-1, expression list
    send data to cassette

RANDOM
    start new random number sequence

READ variable list
    read next entry out of DATA statement list

REM remark
    comment or explanation - this is not executed

RESET(column,row)
    clear dot on display screen

RESTORE
    start DATA list again

RETURN
    finish subroutine and go back to statement after GOSUB

SET(column,row)
    set dot on display screen

STOP
    stop program execution - may go on with CONT

4. FUNCTIONS

```
ABS(arg)        absolute value
ATN(arg)        reverse of TAN - result in radians
COS(radians)    trig cosine (radians=3.14159/18Ø*degrees)
EXP(arg)        e[argument
INT(arg)        returns integer  = argument
LEFT$(string,number)
                leftmost "number" characters from "string"
LEN(string)     number of characters in "string"
LOG(arg)        reverse of EXP - e[result=argument
MID$(string,position,number)
                return string as "number" characters from "position"
                in "string"
POINT(column,row)
                check display dot position
               -1 if set
                Ø if reset
POS(dummy arg)returns present display position (Ø to 63)
RIGHT$(string,number)
                returns "number" characters from right of "string"
RND(arg)        arg >Ø gives random integer between 1 and arg.
                arg =Ø gives random fraction between Ø and 1
SGN(arg)        returns -1 for arg <Ø
                         Ø for arg =Ø
                         1 for arg >Ø
SIN(radians)    trig sine
SQR(arg)        square root of argument ie. arg=resultL2
STR$(arg)       returns string representation of argument
TAB(column)     spaces to "column" (Ø to 63) in PRINT
TAN(radians)    trig tangent
VAL(string)     returns "string" as numeric value
```

5. RESERVED WORDS

Must not be used in variable names

ABS	EDIT	LIST	REM
AND	ELSE	LOAD	RESET
ASC	END	LOC	RESTORE
ATN	ERL	LOF	RESUME
CDBL	ERR	LOG	RETURN
CHR$	ERROR	MEM	RIGHT$
CINT	EXP	MERGE	RND
CLEAR	FIELD	MID$	SAVE
CLOSE	FIX	MKD$	SET
CLS	FOR	MKI$	SGN
CMD	FRE	MKS$	SIN
CONT	GET	NAME	SQR
COS	GOSUB	NEW	STEP
CSNG	GOTO	NEXT	STOP
CVD	IF	NOT	STRING$
CVI	INKEY$	ON	STR$
CVS	INP	OPEN	TAB
DATA	INPUT	OUT	TAN
DEFDBL	INSTR	PEEK	THEN
DEFFN	INT	POINT	TIME$
DEFINT	KILL	POKE	TO
DEFSNG	LEFT$	POS	TROFF
DEFSTR	LET	PRINT	TRON
DEFUSR	LSET	PUT	USING
DELETE	LEN	RANDOM	USR
DIM	LINE	READ	VAL
			VARPTR

NOTES ON USING THE SYSTEM 80 COMPUTER

You probably won't need to worry much about these points while you're working through this book. However if you do go on to delve deeper into programming your System 80, you'll find them well worth bearing in mind.

1. The reason why System 80 gives two READY? messages when you first turn on the power is to give you the opportunity to reserve part of its memory space for machine language or SYSTEM programs. If you just press NEW LINE after the first READY appears, no such space is reserved; but if you type in a number first, System 80 will take that number as the decimal address of the "top" of its memory space available for BASIC programs. Whatever actual memory space is present above that address will be left free for machine language programs.

2. System 80's inbuilt cassette tape deck is fine most of the time, but there are times when you do need an external recorder -- like when you want to use one of the low-cost light pens that need to plug into the recorder's MIC input. System 80 has provision for an external recorder, but don't forget that your programs have to know how to talk to the external recorder instead of the internal deck.
With BASIC programs this can be done very easily, without even modifying the programs themselves. After loading them into the computer, and before typing RUN, just type in:

```
OUT254,255
```

followed by NEW LINE. System 80 will then automatically "connect" your programs to the external recorder whenever they try to communicate with the internal deck.

3. For people who are already familiar with the Tandy TRS-80 computer, System 80's keyboard differs from that on the former in a few minor ways:

(a) The NEW LINE key replaces the TRS-80's ENTER key

(b) The BACKSPACE key replaces the "←" key

(c) The CTRL key replaces the "↓" key

(d) The ESC key replaces the "↑" key

(e) There are no CLEAR or "→" keys. This is not terribly important, as these keys are rarely used. However if the "→" key is needed,it can be added fairly easily. Details are available in DSE Technical Bulletin No.12.

Appendix B. Errors

Error messages are given as

?code ERROR IN line number

for programs, and

?code ERROR

in immediate mode.

The codes are

BS	Bad Subscript. You tried to refer to an array element outside the DIMensioned range - either too big (eg. the 5Ø th element in a 1Ø element array) or wrong dimensions (eg. ELEMENT(1,2) in a one-dimensional array).
CN	Can't Continue. The computer can't continue if it hasn't stopped - BREAK or STOP - if there have been modifications to the program since stopping or if there is no more program to run.
DD	Re-dimensioned array. A subsequent DIM has been found after one has already been assigned either by a DIM statement or by default (when DIM size is 1Ø). A default size is allotted if the array has been referred to before its DIM has been encountered and for this reason it is a good idea to put all DIM statements at the head of the program.
FC	Function Call error. This means that you have given an argument that cannot be used with the function you have called. There are many causes including argument too big (eg. more than 255 with TAB), negative argument with LOG or SQR, array subscript negative or greater than 32767 or non-integral exponent.
FD	File Data bad. While doing an INPUT#-1 System 80 detected an error. There was either a mis-read from the cassette or the variable list in the instruction didn't match the actual data on cassette.
ID	Illegal in Direct mode. An attempt has been made to use instructions in direct (command) mode that can only be used in programs eg. INPUT
MO	Missing Operand. A required argument was not provided eg. * without anything to multiply by, or CSAVE without a name.
NF	NEXT without FOR. Every NEXT statement (end of a FOR/NEXT loop) must be preceded by a FOR (beginning of that loop) using the same variable. This error can be shown if you have simply omitted the FOR line, given a different variable name in NEXT or incorrectly nested loops so that the computer can't get at the matching FOR.

LS String too Long. An attempt has been made to create a string longer than 255 characters.

OD Out of Data. A READ statement has been encountered when there is no DATA for it. DATA may have been omitted or already read. (Include a RESTORE statement to allow DATA to be re-read.) Another possible source is not enough data items on a tape for all the variables in an INPUT#-1 statement (the last bits of data may have been lost from the corresponding PRINT#-1 if they exceeded 255 characters).

OM Out of Memory. Your program has taken up all available memory in System 80. Maybe you have a very large array or your program is reserving too much area for seldom-used branches that can be written more economically.

OS Out of String Space. System 80 allocates space for 5Ø string characters but if you want to use more string space you need to use CLEAR to set it aside first.

OV Overflow. INPUT or result of a calculation is too large for System 80 to handle (in excess of 1E38). Note that underflow becomes zero ie. when numbers are too small for System 80 to deal with accurately they are taken to be zero.

RG RETURN without GOSUB. A RETURN statement signifies the end of a subroutine which must have been entered with GOSUB (not GOTO). RETURN tells the computer to go back to the line after GOSUB which can't be done if there is no such statement.

SN Syntax error. Something is wrong in the characters that you are using - spelling error, incorrect punctuation, missing parenthesis, illegal character, reserved word in variable name are some possibilities.

ST String expression too complex. System 80 couldn't handle the string operation as it was given. Can it be made simpler or could you split it up into smaller steps?

TM Type Mismatch. One side of assignment numeric and the other side string.

UL Undefined Line number. You tried to refer or branch to a non-existent line.

/Ø Division by Zero. That's illegal and illogical! Perhaps your number was so small that it was taken as zero.

These are the error codes that are mentioned in this book. There are some more that System 80 uses and for these you can refer to the System 80 Basic Manual.

Appendix C. Answers to Exercises

```
10 REM CHAPTER 4, EXERCISE 1
20 REM SPEED IN KM/H & TIME IN HOURS, FIND DIST IN KM
30 SPEED=400
40 TIME=5
50 DIST=SPEED * TIME
60 PRINT DIST;"KM FROM THE ALICE TO PERTH."
```

```
10 REM CHAPTER 4, EXERCISE 2
20 REM RATE IN $/MONTH, TIME IN YEARS, FOR REPAY IN $
30 RATE=280
40 TIME=25
50 REPAY=RATE * TIME * 12
60 PRINT"TOTAL REPAYMENT IS $";REPAY
```

```
10 REM CHAPTER 4, EXERCISE 3
20 REM PART 1
30 REM V VELOCITY, U INITIAL V, A ACCELERATION, T TIME
40 U=50000
50 A=2000
60 T=5
70 V=U + A * T
80 PRINT V;"KM/SEC AFTER";T;"SECONDS"
100 REM PART 2
110 REM S DISTANCE
120 S=U*T + 1/2 * A * T*T
130 PRINT"AT A DISTANCE OF";S;"KM"
```

```
10 REM CHAPTER 5, EXERCISE 4
20 REM V VELOCITY, T TIME, S DISTANCE
30 V=3E5
40 T=5E2
50 S=V * T
60 PRINT S;"KM FROM EARTH TO THE SUN"
```

```
10 REM CHAPTER 5, EXERCISE 5
20 REM SEE EX 3
30 U=5E4
40 A=2E3
50 T=25E0
60 V=U + A*T
70 S=U*T + 1/2*A*T*T
80 PRINT V;"KM/SEC AT";S;"KM"
```

```
10 REM CHAPTER 6, EXERCISE 6
20 REM 1
30 N=1+3*4
40 PRINT N
50 REM 2
60 N=(2+4)*2
70 PRINT N
80 REM 3
90 N=2[3-1
100 PRINT N
110 REM 4
120 N=-3[2
130 PRINT N
```

```
10 REM CHAPTER 6, EXERCISE 7
20 L= TAN( 55 * 3.14159/180 ) * 6 * 2
30 REM IE 55 DEGREES TO RADIANS, THEN TAN GIVES HEIGHT TO BASE
40 REM RATIO, SO * 6 GIVES THE HEIGHT. THEN *2 FOR CORD REQUIRED.
50 PRINT L;"M OF CORD REQUIRED"
```

```
1 REM CHAPTER 7, EXERCISE 8
5 COUNT=0
10 PRINT "DECISIONS!"
11 COUNT=COUNT+1
12 REM THIS IS THE NEW LINE!
13 IF COUNT=5 THEN PRINT "MORE"
14 IF COUNT=7 THEN PRINT "I'M NOT FINISHED YET"
15 IF COUNT=10 THEN END
20 GOTO 10
```

```
10 REM CHAPTER 8, EXERCISE 9
20 REM N IS NUMBER OF TIMES, C IS COUNT
30 INPUT"HOW MANY TIMES";N
40 C=0
50 PRINT"QUESTIONS!"
60 C=C+1
70 IF C=N THEN END
80 GOTO 50
```

```
10 REM CHAPTER 8, EXERCISE 10
20 REM YOUR CHOICE HERE!
30 REM TO MAKE EX 9 GO BACK AND ASK AGAIN,
40 REM REPLACE LINE 70 WITH
50 REM 70 IF C=N THEN GOTO 30
```

```
?"CHAPTER 10, EXERCISE 11"
?30*365*24;"HOURS OLD (MINUS A FEW MONTHS!)"
```

```
10 REM CHAPTER 12, EXERCISE 12
20 FOR X=1 TO 10
30 PRINT X
40 NEXT X
```

```
 10 REM CHAPTER 12, EXERCISE 13
 20 FOR X=1 TO 10
 30 PRINT X,X[2
 40 NEXT X

 10 REM CHAPTER 12, EXERCISE 14
 20 FOR X=0 TO 20
 30 PRINT X/10,SIN(X/10)
 40 NEXT X

 10 REM CHAPTER 13, EXERCISE 15
 20 INPUT"INTERVAL IN SECONDS";S
 30 FOR C=1 TO 345*S
 40 NEXT C
 50 PRINT"THAT'S ALL FOLKS"

 10 REM CHAPTER 14, EXERCISE 16
 20 REM PLOT X[2
 30 FOR X=1 TO 7 STEP 0.5
 40 PRINT TAB(X[2);"*"
 50 NEXT X

 10 REM CHAPTER 14, EXERCISE 17
 20 REM PLOT SIN(X)/X
 30 FOR X=1 TO 14
 40 PRINT X;TAB(SIN(X)/X*30+30);"*"
 50 NEXT X

 10 REM CHAPTER 15, EXERCISE 18
 20 M MINUTES COUNTER, S SECONDS, D DELAY
 30 FOR M=5 TO 1 STEP -1
 40 PRINT M;"MINUTES TO BLAST OFF"
 50 FOR S=1 TO 50
 60 FOR D=1 TO 345
 70 NEXT D
 80 NEXT S
 90 FOR S=10 TO 1 STEP -1
100 PRINT S
110 FOR D=1 TO 345
120 NEXT D
130 NEXT S
140 NEXT M
150 PRINT"F I R E  !"
```

```
160 REM CHAPTER 15, EXERCISE 19
170 REM PART 4 - 10 ROCKETS
180 FOR R=1 TO 10
200 REM PART 1 - A ROCKET
210 PRINT" A"
220 PRINT" H"
230 PRINT"<A>"
300 REM PART 2 - EXHAUST
310 FOR E=1 TO 8
320 PRINT" *"
330 FOR D=1 TO 100
340 NEXT D
350 NEXT E
400 REM PART 3 - SPACE
410 FOR S=1 TO 8
420 PRINT
430 FOR D=1 TO 100
440 NEXT D
450 NEXT S
500 REM OTHER BIT OF PART 4
510 NEXT R

10 REM CHAPTER 18, EXERCISE 20
20 BAL=0
30 INPUT"1 FOR DR OR 2 FOR CR";OPT
40 IF OPT=1 THEN PRINT"DR AMOUNT";
50 IF OPT=2 THEN PRINT"CR AMOUNT";
60 GOSUB 200
70 IF OPT=1 THEN BAL=BAL-AMT
80 IF OPT=2 THEN BAL=BAL+AMT
90 PRINT BAL
100 GOTO 30
200 REM SUBPROGRAM
210 INPUT AMT
220 IF AMT<0 THEN GOTO 210
230 IF AMT>100 THEN GOTO 210
240 RETURN
```

```
10 REM CHAPTER 18, EXERCISE 21
20 BAL=0
30 REM CONTROL MODULE
40 INPUT"1=DR,2=CR";OPT
50 ON OPT GOSUB 1000,2000
60 PRINT BAL
70 GOTO 40
1000 REM DR MODULE
1010 PRINT"DR AMOUNT";
1020 GOSUB 9000
1030 BAL=BAL-AMT
1040 RETURN
2000 REM CR MODULE
2010 PRINT"CR AMOUNT";
2020 GOSUB 9000
2030 BAL=BAL+AMT
2040 RETURN
9000 REM INPUT MODULE
9010 INPUT AMT
9020 IF AMT<0 THEN GOTO 9010
9030 IF AMT>100 THEN GOTO 9010
9040 RETURN
```

```
10 REM CHAPTER 19, EXERCISE 22
20 DATA 465,591,615,988,1005,1213
30 INPUT"CUSTOMER NUMBER";C
40 RESTORE
50 FOR N=1 TO 6
60 READ BAD
70 IF C=BAD THEN GOTO 200
80 NEXT N
90 REM NOT FOUND, SO OK
100 PRINT"CREDIT OK!"
120 GOTO 30
200 REM WOOPS, FOUND.
210 PRINT"HE'S A NO-NO"
220 GOTO 30
```

```
10 REM CHAPTER 19, EXERCISE 23
20 DATA"HONG","MAGGIE","80 THE ROAD","SYDNEY 2000"
30 DATA"RALPH","INGRID","1 COMPUTER WAY","HOBART 7000"
100 DATA"*"
120 INPUT"SURNAME";SN$
130 RESTORE
140 READ DS$
150 IF DS$="*" THEN GOTO 300
160 IF DS$=SN$ THEN GOTO 200
170 REM PASS EXTRA ENTRIES!
180 READ DG$,D1$,D2$
190 GOTO 140
200 PRINT "FOUND"
210 READ DG$,D1$,D2$
220 PRINT DG$;" ";DS$
230 PRINT D1$
240 PRINT D2$
250 GOTO 120
300 PRINT"NOT KNOWN!"
310 GOTO 120

10 REM CHAPTER 20, EXERCISE 24
20 INPUT"YES";ANS$:IF ANS$="YES" THEN PRINT"OK":GOTO 20
30 PRINT"NOT OK":GOTO 20

10 REM CHAPTER 21, EXERCISE 25
20 D1=RND(6)
30 D2=RND(6)
40 PRINT YOUR THROW IS";D1;"AND";D2

10 REM CHAPTER 21, EXERCISE 26
20 DATA"JACK","QUEEN","KING","ACE","SPADES","CLUBS"
21 DATA"DIAMONDS","HEARTS"
30 N=RND(13) : REM 13 CARDS IN A SUIT
40 S=RND(4) : REM 4 SUITS
50 PRINT"YOUR DRAW IS THE ";
60 IF N<10 THEN PRINT N+1; : GOTO 200
70 RESTORE : FOR C=1 TO N-9 : READ CD$ : NEXT C
80 PRINT CD$;
200 RESTORE : FOR C=1 TO S+4 : READ ST$ : NEXT C
210 PRINT" OF ";ST$

10 REM CHAPTER 22, EXERCISE 27
20 FOR R=0 TO 47 : REM BLOCK
30 FOR C=0 TO 127
40 SET(C,R)
50 NEXT C : NEXT R
60 RESET(RND(128)-1,RND(48)-1) : REM TERMITES!
70 GOTO 60
```

```
10 REM CHAPTER 22, EXERCISE 28
20 CLS
30 X=RND(127) : Y=RND(47)
40 SET(X,Y) : REM MAIN SNAIL LOOP
50 X=X+RND(3)-2
60 IF X<0 THEN X=0
70 IF X>127 THEN X=127
80 Y=Y+RND(3)-2
90 IF Y<0 THEN Y=0
100 IF Y>47 THEN Y=47
110 GOTO 40
```

```
10 REM CHAPTER 23, EXERCISE 29
11 REM ADD LINES 45 & 105 TO EXERCISE 28
20 CLS
30 X=RND(127) : Y=RND(47)
40 SET(X,Y) : MAIN SNAIL LOOP
45 XS=X : YS=Y : REM SAVE POSITION
50 X=X+RND(3)-2
60 IF X<0 THEN X=0
70 IF X>127 THEN X=127
80 Y=Y+RND(3)-2
90 IF Y<0 THEN Y=0
100 IF Y>47 THEN Y=47
105 IF POINT(X,Y)=-1 THEN X=XS : Y=YS : GOTO 50
110 GOTO 40
```

```
1 REM CHAPTER 23, EXERCISE 30
10 DIM QUANT(50)
20 INPUT"ENTRY";E
30 INPUT"VALUE";QUANT(E)
40 FOR E=0 TO 50
50 IF QUANT(E)<>0 THEN PRINT E,QUANT(E)
60 NEXT E
70 GOTO 20
```

```
?"CHAPTER 24, EXERCISE 31"
NAM$="SKYWALKER"
?1,LEN(NAM$)
?2,LEFT$(NAM$,3)
?3,RIGHT$(NAM$,6)
?4,RIGHT$(NAM$,LEN(NAM$))
```

```
10 REM CHAPTER 24, EXERCISE 32
20 X=0
30 INPUT RQ$ : REM CALCULATOR LOOP
40 FOR I=1 TO LEN(RQ$)
50 IF MID$(RQ$,I,1)=" " THEN GOTO 70
60 NEXT I : GOTO 200
70 NMBR=VAL(RIGHT$(RQ$,LEN(RQ$)-I))
80 IF LEFT$(RQ$,I-1)="ENTER" THEN X=NMBR
90 IF LEFT$(RQ$,I-1)="ADD" THEN X=X+NMBR
100 IF LEFT$(RQ$,I-1)="SUB" THEN X=X-NMBR
200 PRINT X : GOTO 30
```

```
10 REM CHAPTER 25, EXERCISE 33
20 PRINT"ENTER SWITCH STATES AS ON OR OFF"
30 INPUT"SW 1";S1$
40 INPUT"SW 2";S2$
50 INPUT"SW 3";S3$
60 INPUT"SW 4";S4$
70 IF S1$="ON" AND S2$="ON" AND S3$="ON" THEN GOTO 200
80 IF S4$="ON" THEN GOTO 200
90 PRINT"DARKNESS PREVAILS!"
100 GOTO 10
200 PRINT"THE FIRMAMENT IS INDEED ILLUMINATED!"
210 GOTO 10
```

```
10 REM CHAPTER 26, EXERCISE 34
20 CLS : REM BUSINESS FORM!
30 FOR X=0 TO 127
40 SET(X,0) : SET(X,6) : SET(X,47)
50 NEXT X
60 FOR Y=0 TO 47
70 SET(0,Y) : SET(127,Y)
80 NEXT Y
90 PRINT@84,"SYSTEM 80 FANCY TITLE";
100 GOTO 100 : REM STOP PROMPT
```

```
10 REM CHAPTER 26, EXERCISE 35
20 CLS : REM ACE OF HEARTS
30 FOR X=1 TO 20
40 SET(X,0) : SET(X,21)
50 NEXT X
60 FOR Y=1 TO 20
70 SET(0,Y) : SET(21,Y)
80 NEXT Y
90 PRINT@65,"A"; : PRINT@393,"A";
100 FOR Y=0 TO 3
110 SET(8,Y+8) : SET(9,Y+9) : SET(10,Y+10)
120 SET(12,Y+8): SET(11,Y+9)
130 NEXT Y
140 SET(7,9) : SET(7,10) : SET(13,9) : SET(13,10)
150 PRINT@512, : REM JUST POSITION PROMPT!!
```

```
10 REM CHAPTER 26, EXERCISE 36
20 DIM NAM$(5),Q(5)
30 CLEAR 200
40 FOR P=1 TO 5
50 INPUT"NAME, QUANTITY";NAM$(P),Q(P)
60 NEXT P
70 BIG=0 : REM FIND BIGGEST FOR SCALING
80 FOR P=1 TO 5
90 IF Q(P)>BIG THEN BIG=Q(P)
100 NEXT P
110 CLS
120 PRINT"PRODUCT","% SALES"
130 FOR P=1 TO 5
140 PRINT LEFT$(NAM$(P),16)
150 FOR X=0 TO Q(P)/BIG *90 : REM SCALE BAR CHART
160 SET(X+32,P*3+1)
170 NEXT X : NEXT P
```

```
10 REM CHAPTER 27, EXERCISE 37
20 DIM NAD$(5)
30 INPUT"NAME";NAD$(1)
40 FOR I=1 TO 4
50 INPUT"ADDRESS LINE";NAD$(I+1) : NEXT I
60 PRINT"PREPARE CASSETTE TO RECORD PLEASE"
70 INPUT"PRESS 'NEW LINE' WHEN READY";ANS$
80 FOR I=1 TO 5 : PRINT#-1,NAD$(I) : NEXT I
90 PRINT"OK, STOP THE TAPE PLEASE"
100 PRINT:PRINT:PRINT
110 PRINT"NOW WE'LL RELOAD IT"
120 PRINT"REWIND TAPE PLEASE"
130 INPUT"PRESS 'NEW LINE' WHEN READY";ANS$
140 FOR I=1 TO 5 : INPUT#-1,NAD$(I) : NEXT I
150 PRINT"STOP THE TAPE PLEASE"
160 PRINT:PRINT:PRINT
170 PRINT"THIS IS WHAT WE GOT;":PRINT
180 FOR I=1 TO 5
190 PRINT NAD$(I)
200 NEXT I
```

Appendix D. Getting started before you read any books !

If you're like most people, the first thing you'll want to do when you get your new System 80 computer home is plug it in and get a program running - long before you start reading through any books at all!

This is natural enough; in fact you'll find that there's a demonstration cassette with your System 80 designed for this very purpose. Or you can buy any of the pre-recorded program cassettes for the System 80 or the Tandy TRS-80 Level II.

To get such a pre-recorded program running, you only need to read chapter 1 of this book. This tells you how to connect up your System 80 and get it ready for business. Then you work through the following short sequence of steps:

1. Press the $\boxed{\text{F1}}$ button above the keyboard on the System 80 (the adjacent red lamp should come on). Now press the STOP-EJECT key on the tape deck. This will open the deck lid, so that you can insert the cassette - make sure that the side marked "DEMONSTRATION" (or the wanted program with another program cassette) is uppermost.

2. Press the REW key, to rewind the cassette to the start. Now press the STOP-EJECT key, and remove the cassette. Then with a pencil or your finger, wind the tape inside the cassette until the brown magnetic part of the tape reaches the front gaps. Now re-insert the cassette, same side up as before.

3. Press the $\boxed{\text{F1}}$ button again (the lamp should now go out), and press the PLAY key on the tape deck until it latches down.

4. Now type in the command NEW, followed by $\boxed{\text{NEW LINE}}$, and follow this with the second command CLOAD (again followed by the $\boxed{\text{NEW LINE}}$ key). The tape should now move, and the first program on the tape will be loaded.

5. When the word READY re-appears on the screen, press the STOP key on the cassette deck to make sure that the tape stops. Then type in the command RUN, again followed by $\boxed{\text{NEW LINE}}$. The program loaded from tape should now run.

6. To stop a program running, press the $\boxed{\text{BREAK}}$ key.

7. To load in a new program, type NEW and $\boxed{\text{NEW LINE}}$ to erase the existing program. Then press the PLAY key of the cassette deck, and type in CLOAD again followed by $\boxed{\text{NEW LINE}}$.

8. Note that the System 80 demonstration cassette has five different demonstration programs. These can be specified when you give the CLOAD command, if you wish, by following the word CLOAD with the first letter of the program name, in quotes. So to specify the first program, called GRAPHICS, you would type CLOAD"G". Similarly to specify the last program, called STAR WAR, you would type CLOAD"S". If you do give the first letter of a program name in this way, the System 80 will search along the tape until it finds a program whose name starts with the specified letter. (If you just type CLOAD, it will simply load in the next program it finds) Don't forget to always press the $\boxed{\text{NEW LINE}}$ key after typing in a command such as NEW, CLOAD, CLOAD"G" or RUN.

The above "short instructions" should let you get your System 80 running very quickly on pre-recorded programs. But in order to write your own programs you'll still have to read the rest of this book, and preferably the manuals which come with the System 80 itself...