Gateway to Computing

with the

Amstrad CPC464

Ian Stewart

SHIVA

BOOK 2

Gateway to Computing

with the Amstrad CPC464

Ian Stewart is Reader in Mathematics at the University of Warwick, having been Visiting Professor to a number of overseas universities. He has contributed to several computer magazines, including *Sinclair User, Oric Owner* and *Popular Computing Weekly*, and has written for *The Guardian, Nature, New Scientist* and *Scientific American*. Ian also writes occasional science fiction stories for *Analog* and *Omni*, and is a member of the British Science Fiction Association.

Now in his thirties, Ian has already written more than forty books, about half of these being computer books written jointly with Robin Jones including: *PEEK, POKE, BYTE & RAM!, Machine Code and Better BASIC* and *Easy Programming for the ZX Spectrum*. His books have been translated into ten languages. He is an amateur cartoonist under the pseudonym 'Cosgrove' and has published three cartoon books on advanced mathematics – in French – as well as *Computing: a Bug's Eye View*.

Ian lives in a small Warwickshire village with one wife, two sons, and two cats rejoicing in the names *Star* and *Stripes*. His hobbies include home computing, science fiction, playing the guitar, painting scenery and making wine.

Eleanor Ball's first encounter with computers happened by accident in 1966, whilst analysing the chemical properties of wheat and bread flour.

She soon abandoned the direction suggested by a BSc General Degree from London University in favour of computing. The next five years were spent as a Programmer with British Airways, establishing systems on various mainframe computers and terminal equipment at bases throughout Europe.

In 1973, Eleanor retired from city life and came to settle in Cheshire with her husband and young family. It was to be nine years before Shiva discovered her, in the early years of the company, and her interest in computers was rekindled. Eleanor's freelance editorial work for Shiva has turned into an almost full-time occupation, with even the family being caught up in the *Gateway* series, as her husband and children have all shared her involvement in its production and testing!

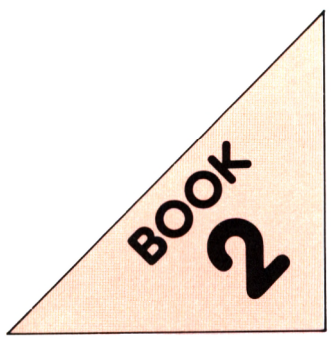
GATEWAY TO COMPUTING

with the Amstrad CPC464

Ian Stewart
Mathematics Institute, University of Warwick

Series Editor
Eleanor Ball

Shiva Publishing Limited



SHIVA PUBLISHING LIMITED
64 Welsh Row, Nantwich, Cheshire CW5 5ES, England

ISBN 1 85014 023 5

Typeset by MHL Typesetting Limited, Coventry
and printed by Devon Print Group, Exeter

Contents

Introduction

The *Gateway to Computing* series is designed to introduce young people (and those who are young at heart) to fundamental ideas of computer programming in an entertaining and comprehensible way. Each volume is available in several different versions for the main models of home computer currently popular. Book 1 deals mostly with the meanings of the main BASIC commands. Although this volume does introduce several new commands, the emphasis is on programming techniques and structure.

I sometimes worry that we're producing a world run by machines that hardly anyone understands. Being able to write programs is only one facet of what is fashionably called 'computer literacy', but it's an important one. It helps rob computers of their mystique. And there we're going to need all the help we can get.

The series is based on the belief that it is possible to be serious about something without being solemn. Learning can be fun. Difficult ideas can be easier to grasp if they are presented in a lighthearted way. Hence the cast of curious characters that romp through these pages: Squire Stoatthrostle; Sherlock Holmes and Dr Watson; Bernard and Ermintrude Gasquet; and Shifty Syd the Scurrilous Salesman – to name but a few.

With their invaluable assistance (and occasional obstruction) *Gateway to Computing Book 2* leads its readers gently but thoroughly through a variety of topics, including:

- Data storage
- Computer logic
- Debugging
- String building

- Multiple loops
- Data manipulation

As well as discussion of computing, there are puzzles, problems, and programs to help the reader practise new techniques. Answers are provided at the end of each chapter.

The numerous sample programs include:

- Analysis of weather data
- A prime number generator
- Pete's Phonebook
- Pickaxo's Plotter

The first chapter is a brief resume of the main topics covered in Book 1, and the rest of the material is carefully selected to be at a suitable level for anyone who has already mastered those elementary BASIC techniques.

The *Gateway to Computing* series has, I fondly hope, two main virtues:

- It's understandable
- Above all, it's *fun*

1 Holmes Recollects ...

As I climbed the long flight of stairs to the private ward in the Royal Nose, Throat and Private Eye Hospital in Bethnal Green, a single phrase kept pounding through my mind. *That fiend, Moriarty (may he rot for eternity)!* For it was Professor Moriarty, the Napolean of Crime, who had brought my illustrious colleague Mr Sherlock Holmes to these dire straits. I wondered how much damage a twenty-pound jade frog could do to an unprotected human skull. I must admit, I feared the worst.

So, when the nurse bade me enter the dimly lit bedchamber, I was relieved to see Holmes sitting up in bed, hastily concealing a pipe under his pillow. But his next words sent a chill along my spine.

"Who the devil are you?" he said.

"Holmes! Do you not remember me?"

"No," he said quietly. "I don't even remember *me*. I have no idea who I am."

"Amnesia!" I cried.

"Delighted to meet you, Mr Amnesia," said Holmes at once. "I can see by the cut of your lapel buttonhole that you buy your suits at Popodopoulos's Tailors in Athens. Your haircut is obviously Greek, and your eyebrows have an ill-balanced droop that is generally acquired by one who spends much of his life on long sea-voyages. Are you by any chance Mr Stavros Z. Amnesia, the shipping magnate?"

"No, no," I interrupted him – though in truth it pleased me to see that he had lost none of his acuity of observation, nor his remarkable deductive talents. "My name is John H. Watson, MD, and you are the famous Sherlock Holmes. Whereas *amnesia* – "

"I don't remember anything," said Holmes.

"Absolutely right, old chap!" I was pleased that Holmes had not forgotten his medical terminology.

"For what am I famous?" asked Holmes.

"You are the world's most celebrated and most able detective," I told him. "And, on the side, one of the meanest debuggers in the computer business." (My readers will no doubt recall Holmes's decisive contributions to an excellent tome called *Floodgates to Computing Volume 1*, or something like that.)

"What is a computer, Dr Holmes?" he asked plaintively.

"No, *you're* Holmes. A computer, my dear Holmes, is an information-processing device." He looked at me blankly, and I tried again. "A machine for manipulating data. An artificial brain. A number-cruncher." Still blank. "A technological and educational breakthrough: no home should be without one." Nothing. "The greatest invention since the safety-pin." Still nothing. "A plastic box with lots of buttons that people use to play TV games."

He smiled, suddenly. "Oh, one of *those*," he said.

At last, a sign that his memory was returning! "But you, Holmes," I went on, "you did more than play games. You wrote *programs*."

His smile broadened. "Yes, yes, of course I did. How fascinating, Dr Whatsit." The smile faded and he broke off. "Just one thing bothers me, though."

I waited.

"I haven't the foggiest idea what a program is."

PROGRAMS

"A program, Holmes, is a list of instructions for the computer to carry out, in order to perform some particular task. It is written in one of a number of *languages* specially designed for the purpose. A very common and popular language is called BASIC. That stands for –"

"**B**ending **A**cronyms **S**o the **I**nitials look **C**ute."

"No, Holmes: it's **B**eginners' **A**ll-purpose **S**ymbolic **I**nstruction **C**ode."

"Quite," said Holmes drily.

A Transparent Device whereby the Author Reminds his Readers of what They should have Remembered from Book 1

"I shall remind you of some of the features of BASIC, Holmes," I said in my helpful manner. "Perhaps that will jog your memory." I searched through my pocket-book as I spoke. "Commands in BASIC are given using certain *keywords*: things like **RUN, LIST, STOP, NEW, FOR, NEXT, TO, IF, THEN, INPUT, PRINT, LET, GOTO.** Do those ring any bells?"

His brow furrowed in thought. "Are they perhaps types of traffic-signal for those newfangled horseless carriage things?"

At that moment I thought I smelled something – it reminded me of burnt feathers. But I was distracted by Holmes's reply. "Not *quite*, Holmes," I humoured the poor lunatic. "But you're getting warm."

"Deuced funny," he said. "I do believe you're right."

"Actually, Holmes, they are BASIC keywords." At last I found the card that I was looking for – it was in my trouser pocket. "This is a BASIC reference card, based on an admirable book which I *think* was called *Gatepost to Computing Book 74*, though my recollection may be at fault." And I showed Holmes the card.

Watson's Handy Reference Card

RUN Carry out a program.

LIST List the commands in a program.

STOP What do you *think* it means?

NEW Clear out an old program from memory.

FOR/NEXT/TO Used to form a *loop*. The loop starts with a command such as **FOR** N = 1 **TO** 1Ø and ends **NEXT** N. The commands in between are carried out 1Ø times, with N taking values 1, 2, . . . , 1Ø.

IF/THEN Branch commands. **IF** (condition) **THEN** (action) will cause the action to be taken if the condition is true. If it is false, the program goes on to the next line.

INPUT Allows the user to tell the computer a number, or a string. A *string* is any sequence of symbols, such as "I8NY" or "FRED". The quotes are *not* part of the string, but are used to show where it starts and stops.

PRINT Display a symbol on the TV screen.

LET Assign a value to a variable. A *variable* is a named area of memory that can be used to store a number or a string that may change. **LET** K = 17 sets numeric variable K to the value 17. **LET** B$ = "FRED" sets string variable B$ to the value "FRED". Note the $ sign on names of string variables.

GOTO Causes the program to jump to a new line. Frowned upon in polite society. Still, what's wrong with a few **GOTO**s among friends?

CLS Clears the screen.

HOLMES'S INTEREST REKINDLED

"Well, Holmes?" I enquired anxiously. "Do you recall these BASIC commands, and how to use them?" I fervently hoped so. If he did not, then I feared we would not progress much beyond Chapter 1.

Holmes started to wriggle uncomfortably. "I *think* so,

Whatsit. They do seem somehow familiar. What should I do if I *can't* remember them?"

"I suggest you read a peerless primer that I have in my rooms at 221B Baker Street, Holmes. *Gatehouse to Computing Book 492*, or so I believe. Written by a chap called Iron Stewpot, for Sheba Publishing." (I have always prided myself on the excellence of my memory, and with good reason, as you see.)

"Yes!" cried Holmes. "I do believe my memory is returning! I have a strange picture in my head . . . it looks like . . . like a pair of Wellingtons with mice sitting inside them."

"That's the book, Holmes!"

"Aha. Which leaves only one blank area in my memory, Whatsit. How did I – wait a minute, Whatsit, is someone cooking a pheasant? I swear I can smell – oh, never mind, I prefer grouse anyway. What I was about to ask is: by what sequence of events did I come to be ensconced in this infirmary?"

"*That fiend, Moriarty (may he rot for eternity)!*" I briefly explained to Holmes how he had concealed himself among the rhododendrons at Grating Towers, stately home of the Duke of Westhamptonshire, having been told by an informant that Professor Moriarty was planning a daring theft of the Duke's priceless pearl cufflink-holder. Holmes had been about to apprehend the thief red-handed when an accomplice had swatted him on the cranium with a twenty-pound jade frog. (My personal suspicions lie with the butler, but it could have been the policeman!)

"Of course!" cried Holmes. "And then, no doubt, that fiend Moriarty (may he rot for eternity) and his accomplice absconded with the booty."

"No, they left the dear Duchess unmolested, Holmes."

He ignored my feeble jest. (Booty . . . beauty. Get it? Oh, never mind.)

"Pass me my trousers, Whatsit! We must track the villain down at once!"

"Of course, Holmes. While his trail is still hot!"

Holmes broke into a sweat. Fever? Or merely a burning desire to catch that fiend Moriarty (may he rot for eternity)?

"Whatsit?"

"Yes, Holmes?"

"*Whose* tail did you say was hot?"

"No, Holmes. *Trail.*"

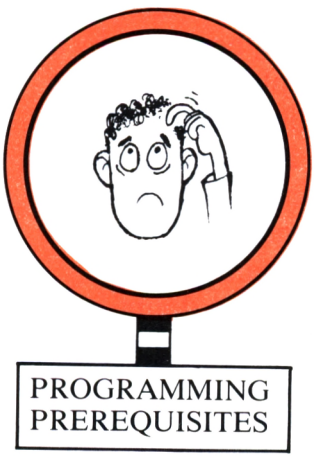
"Oh. I thought . . . You know, I *do* feel very peculiar. I think I may be running a temperature. It's very warm in here, don't you think?"

"Oh, I don't know, Holmes. It's snowing a blizzard outside and Matron has left the windows wide open as usual. I'll just –"

At that moment the pillow burst into flames, and Holmes leaped with great agility from the bed, trailing smoke from the seat of his flannel pyjamas and clasping his hands to the affected area. It would seem that he had been in such a hurry to conceal his pipe from nurse's prying eyes that he had omitted to put it out before stuffing it under his pillow.

Well, speaking as a medical man, I was pleased to see such incontrovertible evidence that, as they say, smoking *can* damage your health.

COMMERCIAL BREAK



Watson and Holmes's little episode is intended to make sure that you remember the main ideas from *Gateway to Computing Book 1*. You don't have to have read that book, but you should recognize the BASIC keywords in Watson's Handy Reference Card, and know how to use them. You should also be able to input a program, edit it, and run it. And fix it up if there's something wrong, using simple debugging techniques.

Here are a few problems to test your knowledge. Make sure you understand the answers before reading the next chapter. All program listings in this book use upper case only. Pressing CAPS LOCK *before* entering any program will automatically generate upper case letters on the screen. (Of course, you can always use lower case if you wish.)

TIM'S TOTALIZATOR

1. Torpid Tim, the Tranquil Trainee, has been told to write a program to add up all the numbers from 1 to 1000. While he has a quiet snooze behind the coffee-machine, see if you can help him out.

CASHSNITCHER'S CREDIT CARD

Carlton Q. Cashsnitcher, Manager of the Lower Standards branch of Gnatwest Bank plc, has decided to send all of his customers an Armenian Excess credit card (don't go home without it) . . . *provided* they have a bank balance of at least £500. He has hired you as a consultant (at an exorbitant fee because he doesn't know any better) to write a program that will accept as input the customer's name and bank balance, and tell his Chief Cashier whether or not to send them a credit card. You must also make the program loop so that it is ready to input the next name; and if you input the name "MONGOOSE" the program should stop. (No, don't ask me *why* " MONGOOSE", OK?)

Can you earn your exorbitant fee?

VOTING MACHINE

The next program is part of a software package produced by Apfelsoft Inc. for the Stork-37 micro, for computerized voting in elections. The users input their choices, and the program counts the votes and says who wins. Like most Stork-37 products, it doesn't work.

Fix it, using, for example, the dry-run technique.

```
1Ø  PRINT "COMPUTAVOTE
    AUTODEMOCRACY PACK V.7B"
2Ø  PRINT "NAME OF CANDIDATE 1?"
3Ø  INPUT N$
4Ø  PRINT "PARTY OF CANDIDATE 1?"
5Ø  INPUT P$
6Ø  PRINT "NAME OF CANDIDATE 2?"
7Ø  INPUT N$
8Ø  PRINT "PARTY OF CANDIDATE 2?"
9Ø  INPUT P$
1ØØ PRINT "GOOD MORNING, VOTERS!"
11Ø PRINT "THIS IS YOUR CHANCE TO"
```

```
120 PRINT "EXERCISE YOUR DEMOCRATIC
    RIGHTS."
130 PRINT "YOUR CHOICE IS BETWEEN"
140 PRINT "1: [SPACE] ";N$;" [SPACE] OF THE
    [SPACE] ";P$;" [SPACE] PARTY"
150 PRINT "AND"
160 PRINT "2: [SPACE] ";N$;" [SPACE] OF THE
    [SPACE] ";P$;" [SPACE] PARTY"
170 PRINT "PLEASE INPUT YOUR CHOICE: 1
    OR 2"
180 LET T1 = 0
190 LET T2 = 0
200 INPUT C
210 IF C = 1 THEN LET T1 = T1 + 1
220 IF C = 2 THEN LET T2 = T2 + 1
230 PRINT "HAS THE POLL CLOSED?"
240 INPUT A$
250 IF A$ = "NO" THEN GOTO 100
260 IF T1 > T2 THEN PRINT "CANDIDATE 1
    WINS FOR THE [SPACE] "; P$;" [SPACE]
    PARTY"
270 IF T1 < T2 THEN PRINT "CANDIDATE 2
    WINS FOR THE [SPACE] "; P$;" [SPACE]
    PARTY"
```

ANSWERS

Tim's Totalizator

```
10 LET SUM = 0
20 FOR N = 1 TO 1000
30 LET SUM = SUM + N
40 NEXT N
50 PRINT "TOTAL IS:";SUM
```

Cashsnitcher's Credit Card

```
10  PRINT "ARMENIAN EXCESS CREDIT
    RATING"
20  PRINT "CUSTOMER'S NAME?"
30  INPUT NAME$
40  IF NAME$ = "MONGOOSE" THEN GOTO
    100
50  PRINT "CURRENT BALANCE?"
60  INPUT BALANCE
70  IF BALANCE < 500 THEN PRINT "DO NOT
    SEND CARD TO [SPACE] ";NAME$
80  IF BALANCE > = 500 THEN PRINT "SEND
    CARD TO [SPACE] ";NAME$
90  GOTO 20
100 PRINT "THIS PROGRAM WAS SPONSORED
    BY"
110 PRINT "ARMENIAN EXCESS
    INCORPORATED"
120 PRINT "(DON'T GO HOME WITHOUT IT)"
```

Voting Machine

The program splits into three main parts. Lines 10–160 set things up ready to run; 170–250 form the main part, where the votes are taken and counted; and 260–270 say which candidate won. The program can be debugged section by section.

The first step is to run the program, input test values, and see what happens. For instance we could take:

```
Name of candidate 1:  M MOUSEBENDER
Party of candidate 1:  SOCIALIST DEMAGOGUE
Name of candidate 2:  J P GROTTY
Party of candidate 2:  SELFSERVATIVE
```

Straight away something goes wrong, because the computer then tells us our choice is between:

1: J P GROTTY OF THE SELFSERVATIVE PARTY
AND
2: J P GROTTY OF THE SELFSERVATIVE PARTY

which seems a little unfair to poor old Mousebender. However, undaunted, we enter one vote for our friend Mousebender (candidate 1) and then tell the computer the poll has closed. Now we're told:

CANDIDATE 1 WINS FOR THE SELFSERVATIVE PARTY

which has the right candidate but the wrong party.
Hmmm.
Here's what my debugging session led to.

1. In lines 30, 50, 70 and 90 the variables N$ and P$ are used twice with different meanings. A dry run shows that these variables always end up containing candidate 2 and party 2. To fix this, we have to separate the variables, say using M$ and P$ for candidate and party 1, and N$ and Q$ for candidate and party 2. Which means we must:

   Change N$ to M$ in line 30
   Change P$ to Q$ in line 90

   There are some consequent changes in the first section of the program:

   Change N$ to M$ in line 140
   Change P$ to Q$ in line 160

   The first part of the program now checks out, but the whole thing is still wrong.

2. A dry run of this section shows that lines 180 and 190 reset T1 and T2 to zero *each time round the loop*. They should come much earlier, *outside* the loop:

   Move line 180 to line 92
   Move line 190 to line 94

3. When you input 'NO' as answer to line 25Ø you get a whole mass of unwanted printout. The jump is to the wrong line.

   In line 25Ø change **GOTO** 1ØØ to **GOTO** 17Ø

4. All now seems well in the middle section of the program, but the final section is all haywire. Whatever else happens, it always tells us that the winner is from the (Party 1) party. Clearly we must:

   Change P$ to Q$ in line 27Ø

5. It now seems to be working pretty well, but that doesn't necessarily mean it's bug-free. Try giving both candidates the same total vote (1 each is easy). Nothing gets printed out. You need a final line:

```
28Ø IF T1 = T2 THEN PRINT "VOTES EQUAL:
    RUN-OFF REQUIRED"
```

2 Loopier and Loopier

How loopy can you get? Quite a bit! Now that you know how to use the simple **FOR** . . . **NEXT** loop, you can go for some fancy loops too. In particular:

(a) Loops that don't go up in steps of 1.
(b) Combinations of several loops.

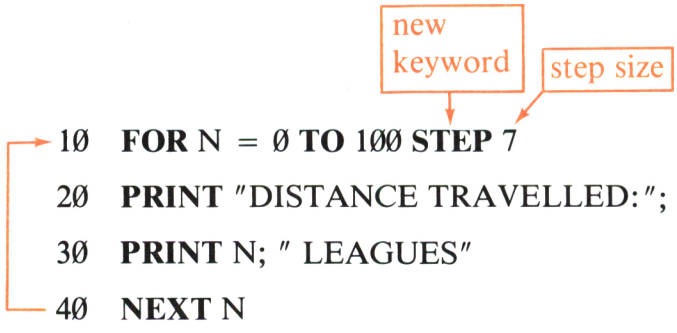
Let's start with (a).

SEVEN LEAGUE BOOTS

There is an old story of a man who had a pair of seven-league boots—boots that took him seven leagues at every step. (A *league* is about 5 kilometres.) In BASIC there is a command:

STEP

that can be used with a **FOR** . . . **NEXT** loop to move the loop counter up (or down) in step sizes different from 1. For instance, here's a print-out of distances that can be travelled in seven-league boots.



This works just like the standard loop, except that N goes up in sevens:

Ø, 7, 14, 21, . . . , 98

It stops there because the next value, 1Ø5, would be greater than 1ØØ, the finish number of the loop. (Seven-league boots are a nuisance if you only want to go half a league to the chip shop!) Similarly:

FOR N = 25 **TO** 35 **STEP** 2

would give the numbers:

25, 27, 29, 31, 33, 35

(the odd numbers between the start and finish values occur because the start is odd and the count goes up in twos), and:

FOR N = 1 **TO** 1ØØØ **STEP** 1ØØ

would go:

1, 1Ø1, 2Ø1, 3Ø1, 4Ø1, 5Ø1, 6Ø1, 7Ø1, 8Ø1, 9Ø1

MOUSEBENDER'S INTELLIGENCE TEST

Marmaduke Mousebender the Mysterious Mathematician is trying to find **FOR** . . . **NEXT** commands to produce the following series of numbers:

(a) 2, 4, 6, 8, 1Ø, 12, 14, 16
(b) 1Ø, 13, 16, 19, 22, 25, 28, 31, 34
(c) 5Ø, 6Ø, 7Ø, 8Ø, 9Ø, 1ØØ
(d) 514, 537, 56Ø, 583, 6Ø6, 629

Can you help him?

COUNTDOWN

You can even use a negative **STEP** size, to count downwards:

count *down* in 1s

```
10 FOR N = 10 TO 0 STEP -1
20 PRINT N
30 NEXT N
40 PRINT "OH WELL, BACK TO THE
   DRAWING-BOARD"
```

MULTIPLE LOOPS

Suppose you wanted to print out a *complete* set of multiplication tables—2 times, 3 times, . . . all the way to 12 times. Go on, suppose. Do it for *me*.

You *could* write 11 separate programs, starting with:

```
10  FOR N = 1 TO 12
20  PRINT N;"*";2;" =";N * 2
30  NEXT N
```

and change the 2 in line 20 to 3, 4, 5, . . ., 12 in turn. But that would take quite a lot of effort. And the programs would all look *almost* the same. Which suggests using a second loop, whose counter K runs from 2 to 12. The main outline of the program would go like this:

```
10  FOR K = 2 TO 12
...
....                    <- program to produce K times table
...
60  NEXT K              <- or whatever fits best
```

Now it's easy enough to modify the 2 times table program above to make a general K times table:

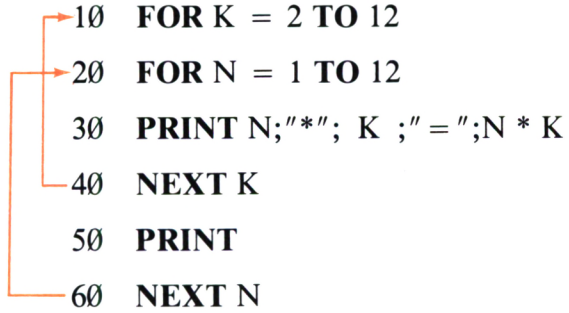
```
20  FOR N = 1 TO 12
30  PRINT N;"*";K;" =";N * K    <- note use of semicolons
40  NEXT N
```

So, fitting this into the outline and tidying a little, we get:

```
10  FOR K = 2 TO 12          <- outer loop
20  FOR N = 1 TO 12          <- inner loop
30  PRINT N; "*"; K; " ="; N * K
40  NEXT N
50  PRINT                    <- puts in a blank line between tables
60  NEXT K
```

Well, that's fascinating! We've ended up with two loops, *one inside the other.*

You might like to experiment with what happens if you get the two **NEXT**s in the wrong order, like this:



```
10 FOR K = 2 TO 12
20 FOR N = 1 TO 12
30 PRINT N;"*"; K ;" =";N * K
40 NEXT K
50 PRINT
60 NEXT N
```

The CPC464 discovers the error, and won't even *start* to run.

> When one loop lives inside another:
>
> (a) Make sure it is completely inside, with the **NEXT**s in the right order.
>
> (b) Use different names for the two loop counters.

Loop counter names can be any valid variable name.

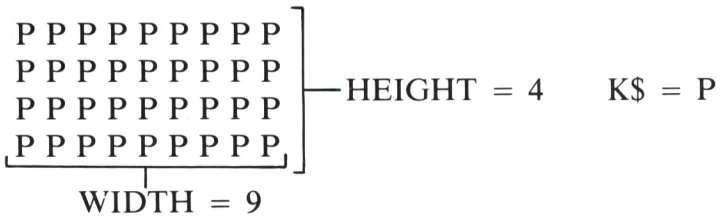
PRESTI-DIGIT-ATION

According to Marmaduke Mousebender, the Magic Mathematician, there is exactly one two-digit number that is twice the product of its digits. Write a program to find it. (Two-digit numbers are those between 10 and 99. The product is what you get by multiplying. For example, you could try 72. Twice the product of the digits is twice 7 * 2, which is twice 14, or 28. This is *not* equal to 72, so 72 doesn't work. Now you've only got 89 more to try . . .)

Hint: use a multiple loop, one loop for each digit.

PICKAXO'S PLOTTER

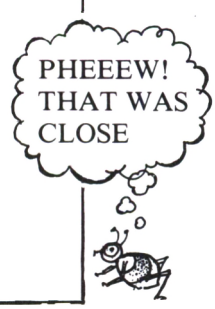
Pablo Pickaxo the Pre-Raphaelite Painter has written a program to produce rectangles on the screen, made up by repeating a single character, like this:



He uses three variables: HEIGHT and WIDTH to set up the shape, and K$ to choose the symbol being plotted.

Unfortunately, a nestful of termites wandered over Pickaxo's program. Pablo debugged it successfully, but in doing so he made a bit of a mess of the program. His mallet got a bit mucky too. Can you fill in the parts that have been obliterated?

```
10  PRINT  RECTANGLE PRINTER "
20    " WHAT WIDTH ?
30  INPUT WIDTH
40  PRINT " WHAT   ?"
50    HEIGHT
      WHICH CHARACTER ? "
70    K
80  CL
90  FOR L   1 TO 5
    PRINT
110 NEXT
120   R$ = " "
130   X = 1 TO
140 LET R$  R  + K$
150   X
160 FOR   1 TO 4
    LET R  = "  SPACE  "  R$
180   Y
    FOR P = 1 TO HEIGHT
200   R$
210
```



ANSWERS

Mousebender's Intelligence Test

(a) **FOR** N = 2 **TO** 16 STEP 2

(b) **FOR** N = 1Ø **TO** 34 STEP 3

(c) **FOR** N = 5Ø **TO** 1ØØ STEP 1Ø

(d) **FOR** N = 514 **TO** 629 STEP 23

Presti-digit-ation

```
1Ø  FOR A = 1 TO 9
2Ø  FOR B = Ø TO 9
3Ø  LET X = 1Ø * A + B
4Ø  IF X = 2 * A * B THEN PRINT "GOT IT!
    SPACE ";X
5Ø  NEXT B
6Ø  NEXT A
```

Pickaxo's Plotter

```
1Ø  PRINT "RECTANGLE PRINTER"
2Ø  PRINT "WHAT WIDTH?"
3Ø  INPUT WIDTH
4Ø  PRINT "WHAT HEIGHT?"
5Ø  INPUT HEIGHT
6Ø  PRINT "WHICH CHARACTER?"
7Ø  INPUT K$
8Ø  CLS
9Ø  FOR L = 1 TO 5
1ØØ  PRINT
11Ø  NEXT L
12Ø  LET R$ = " "
13Ø  FOR X = 1 TO WIDTH
```

```
140  LET R$ = R$ + K$
150  NEXT X
160  FOR Y = 1 TO 4
170  LET R$ = "  SPACE  " + R$
180  NEXT Y
190  FOR P = 1 TO HEIGHT
200  PRINT R$
210  NEXT P
```

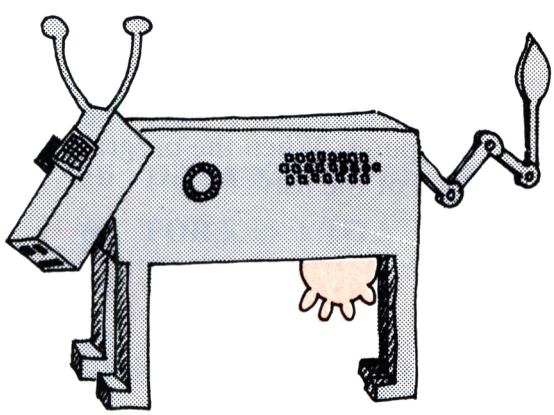
3 She was only a Farmer's DATA

(Pronounce that 'darter' to get the joke.) (What joke? Ed.)

Millicent MacHaddock the Misguided Milkmaid, famous as the inventor of the Millie-Litre, has been asked by her father Hamish MacHaddock to computerize the farm records. Millie decides to start by listing all the cows in her flock. (What? *Herd* of cows? All together now... "Of *course* I've heard of cows!") Each cow has a serial number marked on its ear. (Each cereal has a cow number marked on *its* ear, too, but that's not relevant here.) The full list is too long to record here, but it starts like this:

CORNFLAKE	22443
JUNKET	7145Ø
COWSLIP	22222
FARRAH	93665
XANTIPPE	69888

The problem is: how to put this information into a program... and how to get it out again when needed.



Millicent has a *data-storage* problem. It is solved by the BASIC keywords:

DATA **READ** **RESTORE**

which let you incorporate lists of data into a program, and retrieve items from the list.

COMPUTERIZED COWS

Data can be numbers, or strings, as you wish. Items in a **DATA** list must be separated by commas. Strings do *not* need quotes round them. Millicent's problem is solved, in part, by the program lines:

```
10 DATA CORNFLAKE, 22443
20 DATA JUNKET, 71450
30 DATA COWSLIP, 22222
40 DATA FARRAH, 93665
50 DATA XANTIPPE, 69888
```

and so on.

To *use* this list, Millicent must be able to extract an item from it. This is done by the command:

READ

used in the form:

READ variable

This gives to the stated variable the value of the 'next' item in the **DATA** list.

What do I mean by 'next'? It's like this.

Think of the list as having a *pointer* attached by the computer. When the program is first run, this points to the start of the list:

pointer

```
10 DATA CORNFLAKE, 22443
...
```

Every time a **READ** is performed, the computer moves the pointer one place along the list. So after the first **READ**, we have:


pointer

```
10  DATA CORNFLAKE, 22443
```

Next **READ**, it moves on to point to JUNKET, then to 71450, and so on.

> The variable name used in a **READ** command must have the correct type (numeric or string) for the **DATA** item that it refers to.

That is, the first **READ** would refer to CORNFLAKE, which is a string, so you'd need to use:

```
READ N$
```
dollar sign for a string

but the second would refer to 22443, a number, which requires something like:

```
READ NUMBER
```
no dollar sign on variable name

To start with an easy one, we'll write a program to print out the **DATA** list. Here it is:

```
10  DATA CORNFLAKE, 22443
20  DATA JUNKET, 71450
30  DATA COWSLIP, 22222
40  DATA FARRAH, 93665
50  DATA XANTIPPE, 69888
60  FOR K = 1 TO 5
70  READ N$
80  READ NUMBER
90  PRINT N$,NUMBER
100 NEXT K
```
original **DATA** list

Let's just see how it works, as we run through the loop. The arrows show the effects of the **READ**s.

Value of K	N$ becomes	NUMBER becomes	Pointer points to
(before start)	–	–	CORNFLAKE
1	CORNFLAKE	–	22443
	CORNFLAKE	22443	JUNKET
2	JUNKET	22443	71450
	JUNKET	71450	COWSLIP
3	COWSLIP	71450	22222
	COWSLIP	22222	FARRAH
4	FARRAH	22222	93665
	FARRAH	93665	XANTIPPE
5	XANTIPPE	93665	69888
	XANTIPPE	69888	falls off end: no more **READ**s allowed

See how the pointer just ticks along the list? See how the **READ**s work?

PETE'S PHONEBOOK

Protocol Pete, the Phonophiliac Programmer, wants to store his personal telephone directory as a **DATA** list, and print it out. Here's the directory:

Name	Phone number
Fred Subtlebug	667142
Amanda Bander-Gander	1234567
Lolita Nabokova	9999
Ytzak ben Nevis	434772
Igor Biva	1001001

What does the program look like?

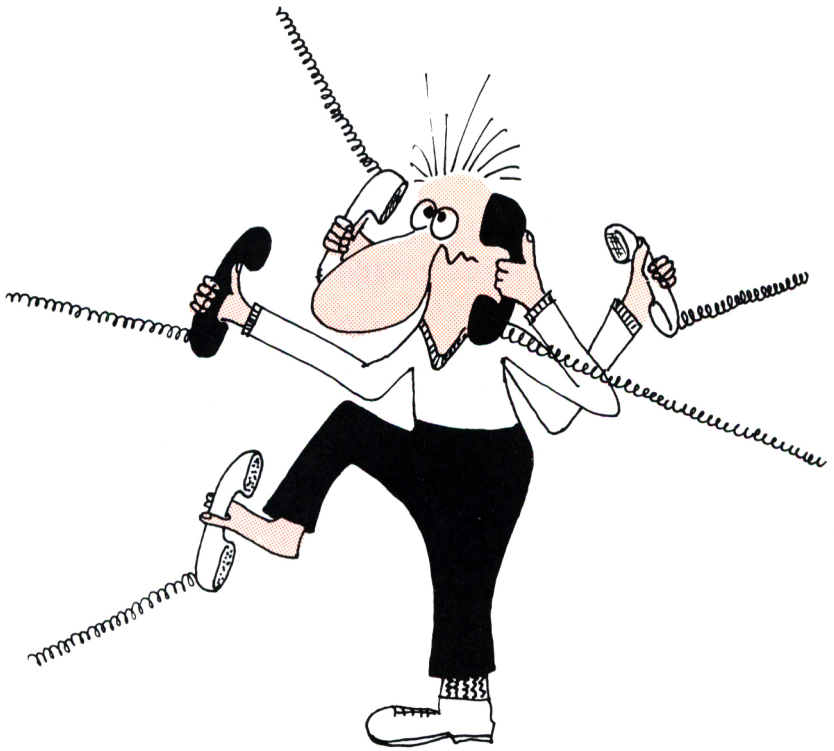


TABLE LOOK-UP

Printing the list out isn't too tricky. Another useful program would be one that *searches* the list for a name, and prints it out together with its number. For Millicent, this would answer questions like 'I wonder what Cowslip's number is?'; and for Pete, 'I must call Lolita at once, but what's her number, drat it?'

The idea is to **READ** through the list, using an **IF** command to take action once you find what you want. Like this:

```
10 DATA FRED SUBTLEBUG, 667142
20 DATA AMANDA BANDER-GANDER,
   1234567
30 DATA LOLITA NABOKOVA, 9999
40 DATA YTZAK BEN NEVIS, 434772
50 DATA IGOR BIVA, 1001001
```

```
60  PRINT "NAME FOR SEARCH?"
70  INPUT SNAME$
80  PRINT "SEARCHING FOR
    SPACE ";SNAME$
90  FOR K = 1 TO 5
100 READ NAME$
110 READ NUMBER
120 IF NAME$ = SNAME$ THEN PRINT
    "FOUND SPACE "; SNAME$,, "
    THE NUMBER IS ";NUMBER
130 NEXT K
```

(a) You can string **DATA** items together in longer lines, and scatter the **DATA** statements throughout the program. Only the overall order counts. The computer combines all **DATA** commands into a single list.

(b) You can **READ** several items in one command, for example:

READ NAME$, NUMBER

The pointer jumps on by the number of items read.

QUICKIE

Use short-cut (b) to simplify the programs in this chapter so far.

RESTORE

I said above that when the pointer gets to the end of the **DATA** list, no more **READ**ing is allowed.

That's true; and if you try to **READ** after falling off the end of the list, you'll cause a crash and get an error message:

DATA exhausted

But you may need to run through a **DATA** list several times—for example, searching for several items, one after the other.

The keyword:

RESTORE

sends the pointer back to the start of the list, ready to begin running through it all over again. Here's a test program.

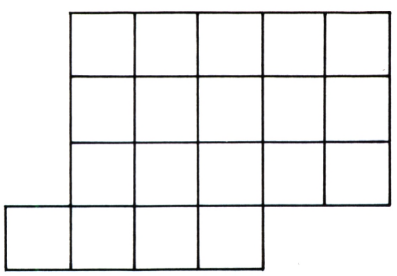
```
10 DATA A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P,Q,
   R,S,T,U,V,W,X,Y,Z
20 FOR K = 1 TO 26
30 READ X$
40 PRINT X$;
50 NEXT K
60 RESTORE
70 GOTO 20
```

$ because a character is a string (with one symbol)

Try this; then delete line 60 and try again. See the difference?

COMPUCROSTIC

Solve for the across words; spot the BASIC keywords in the down direction.



Nose noise

Sour fruit

Small film part

Felines

PROCNATAKADSKI'S PROBLEM

Comrade Sergei Procnatakadski, the other Russian spy, is sending a code message to Leningrad. His code will change the letters of the alphabet like this:

Original	A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
Code	H A M E R N D S I C K L B F G J O P Q T U V W X Y Z

(a) Write a program that will allow Comrade Procnatakadski to input his message, letter by letter, and print out the coded version.
(b) Write a program to *decode* messages the same way.

ANSWERS

Pete's Phonebook

```
10  DATA FRED SUBTLEBUG, 667142
20  DATA AMANDA BANDER-GANDER, 1234567
30  DATA LOLITA NABOKOVA, 9999
40  DATA YTZAK BEN NEVIS, 434772
50  DATA IGOR BIVA, 1001001
100 PRINT "PETE'S PHONEBOOK"
110 PRINT
120 FOR T = 1 TO 5
130 READ X$,Y
140 PRINT X$;"  SPACE  SPACE  SPACE  ";Y
150 NEXT T
```

Quickie

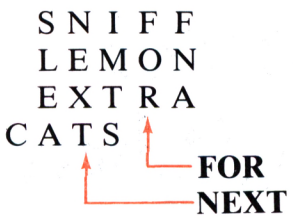
Computerized cows Delete lines 70 and 80. Replace by:

```
70  READ N$, NUMBER
```

Table look-up Delete lines 1ØØ and 11Ø. Replace by:

```
1ØØ READ NAME$,NUMBER
```

Compucrostic



Procnatakadski's Problem

The **DATA** list is the same for both programs: it's broken into bite-sized chunks for convenience.

```
1Ø DATA A,H,B,A,C,M,D,E,E,R,F,N,G,D,H,S
2Ø DATA I,I,J,C,K,K,L,L,M,B,N,F,O,G,P,J
3Ø DATA Q,O,R,P,S,Q,T,T,U,U,V,V,W,W,X,X,
   Y,Y,Z,Z
```

Now to put a message *into* code, add the lines:

```
1ØØ PRINT "NEXT LETTER OF MESSAGE?"
11Ø INPUT M$
12Ø RESTORE
13Ø FOR N = 1 TO 26
14Ø READ A$, C$
15Ø IF A$ = M$ THEN PRINT C$
16Ø NEXT N
17Ø GOTO 11Ø
```

To decode, all you need to do is change line 15Ø to:

```
15Ø IF C$ = M$ THEN PRINT A$
```

The input loop goes on forever. You could add a delimiter command (see Chapter 5):

```
165 IF M$ = "*" THEN STOP
```

which would stop the program if you input '*'.

Bugliness is next to Ugliness

In *Book 1* we took a look at syntax errors and runtime errors. Both of these types of bug make their presence pretty obvious, because they cause the program to crash. Even then they aren't always easy to find. But there are more subtle types of bug, that don't actually halt the program, but just make it do the wrong thing.

Sometimes you can track this kind of bug down by staring at the program listing, but usually all that gives you is eyestrain and a sore head. A better way is to use a bit of detective work; and there are some very useful tricks that involve making small changes to the program to test what it's really doing.

TEST LINES

Hortense Mousebender is writing a program which will let people find out the lunch menu for any day of the week. So far she's got this:

```
10 DATA SUN, ROAST BEEF, MON,
   SPAGHETTI,TUE, FISH AND CHIPS, WED,
   SHEPHERD'S PIE
20 DATA THU, MOUSSAKA, FRI, LASAGNA,
   SAT, STEAK
30 PRINT "SPECIFY DAY"
```

```
40  INPUT SDAY$
50  FOR D = 1 TO 6
60  READ DAY$, MENU$
70  IF DAY$ = SDAY$ THEN PRINT
    "MENU IS [SPACE] "; MENU$
80  NEXT D
```

On running this, she finds it works fine for MON and THU as input days. But when she tries SAT, it doesn't print anything. What's wrong?

* * * * * * *

Holmes sat up irritably. "Yes, Watson?"

"You are in receipt of a telegram from a Madame Mousebender, Holmes."

"Another bug, Watson?"

"Indeed, Holmes."

Holmes took the telegram from my hand. "Let me see it, Watson. Hmmm... I wonder why she has lasagna on Friday...? No matter... Aha! Since the input is causing the problem, I deduce that something must be wrong *after* line 40, where the input first occurs. I suspect the loop, Watson."

"I was about to say the same thing, Holmes."

"No doubt. I have a strong inkling of the solution at this very moment, but to settle the matter beyond any doubt, let us apply a small test."

"I'll get the fingerprint powder at once, Holmes."

"No, no, Watson, you ancient buffoon. Not *that* test. This one!"

Holmes turned to his bedside terminal and added a line:

```
65  PRINT D; DAY$
```

"That should prove extremely revealing," he said.

"*How*, Holmes?" I cried.

"It will show us exactly what is going on within the loop, Watson. In particular, which days are being **READ**."

"Devilishly cunning, Holmes."

Holmes sighed, and typed **RUN**. Across the screen of the TV set there appeared the words SPECIFY DAY. Holmes typed in the offending entry:

```
SAT
```

I watched in anticipation as the screen filled with messages.

```
1  SUN
2  MON
3  TUE
4  WED
5  THU
6  FRI
```

and... stopped.

"But—" I gasped. "Where the devil is SAT?"

"It was never read, Watson. The loop only gets as far as D = 6, which is FRI. And that suggests an obvious culprit—"

"The loop finishnumber!" I cried. "It should be 7!"

"Exactly, Watson."

* * * * * * *

This is one way to see what's wrong with a program. Add *test lines* to print out intermediate values of variables, to see what's happening. Make the computer itself tell you what's going wrong.

If the test lines you try don't seem to help, delete them and try again. When you've found the bug, and swatted it, *take out the test lines.*

Hortense Mousebender got line 5Ø wrong. It should have been:

```
5Ø  FOR D = 1 TO 7
```

correct finishnumber

Fix this line, take out line 65, the test line, and check that the program now works correctly on all seven input days. (Don't forget to use the 3-letter abbreviations for inputs, MON, TUE, etc.)

TRACES

A common piece of bugliness is that the program fails to jump correctly. Marmaduke Mousebender the Manic Mathematician has invented a computer game. The computer chooses a whole number between 1 and 1ØØ. The user makes a guess, and is told whether it is too high or too low. If it is correct, the computer says so.

This is his program.

```
10  DATA 54,72,66,98,4
20  READ CNUMBER          <- number to be guessed
30  PRINT "WHAT IS YOUR GUESS?"
40  INPUT GNUMBER
50  IF GNUMBER >= CNUMBER THEN
    PRINT "TOO HIGH"
60  IF GNUMBER >= CNUMBER THEN GOTO 30
70  IF GNUMBER = CNUMBER THEN GOTO
    100
80  IF GNUMBER<CNUMBER THEN PRINT
    "TOO LOW"
90  IF GNUMBER<CNUMBER THEN GOTO 30
100 PRINT "WELL DONE"
110 PRINT "WANT ANOTHER GO?"
120 INPUT Q$               <- yes or no
130 IF Q$ = "YES" THEN GOTO 20
```

This gives you five goes. For more, add extra numbers to the **DATA** list in line 10.

Well, fine, except that it didn't work. When Marmaduke tried it, he got this:

```
WHAT IS YOU GUESS?
? 43
TOO LOW
WHAT IS YOUR GUESS?
? 55
TOO HIGH
WHAT IS YOUR GUESS?
? 53
TOO LOW
WHAT IS YOUR GUESS?
? 54
TOO HIGH
WHAT IS YOUR GUESS?
?
```

"So it's *between* 53 and 54. And a whole number," said Marmaduke to himself. And he went off to invent a whole new kind of arithmetic, the Theory of Hyperintangible Exotic Numeration Moduli. When that didn't help, he finally realized that there might be a bug in the program.

* * * * * * *

"Holmes, I believe I have an idea!"

"Well done, Watson. Well done."

"I suspect that the program is behaving incorrectly, Holmes."

Holmes slapped his forehead with his palm—a nervous habit I have noted often of late. I continued. "It appears not to be jumping correctly, Holmes."

"Indeed, Watson. And how do you propose to test this wild conjecture?"

I lowered my eyes at the unanticipated query. "I haven't the foggiest idea, Holmes."

The great man furrowed his noble brow in concentration. Or was it sudden, involuntary pain?

"I suggest we put a *trace* on the program, Watson." I confess to my bewilderment. Did Holmes think the program was a horse?

"We must persuade the computer to tell us where it has jumped, Watson. By adding suitable program lines."

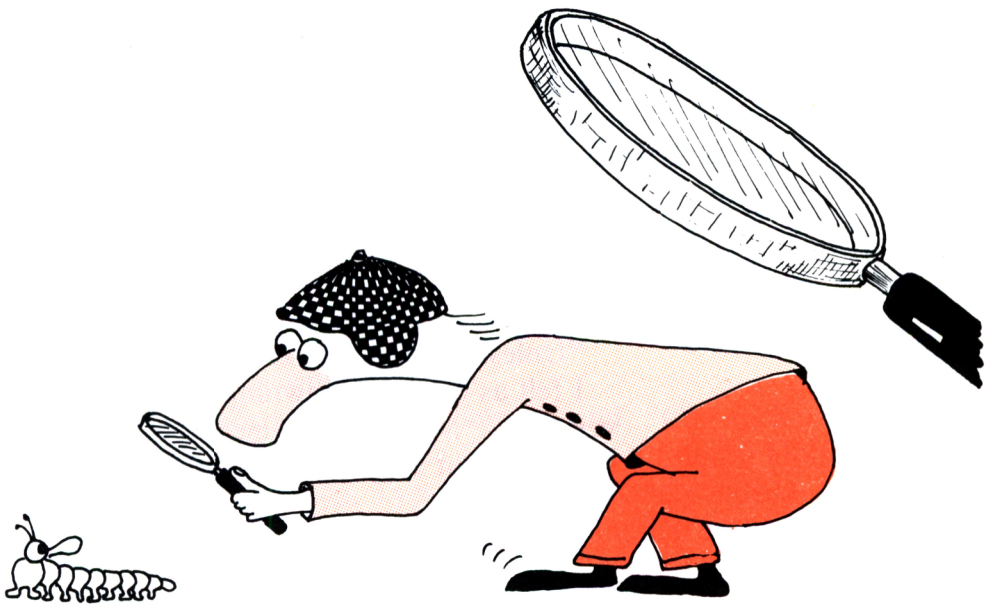
"Aha! Dashed cunning, Holmes!"

"Let me see... The jumps are **GOTO** 3Ø, **GOTO** 1ØØ, and **GOTO** 2Ø. I propose we add lines to say when these jumps have been made."

"But how, Holmes?" He said nothing, but his fingers skipped nimbly over the keyboard:

```
 21  PRINT "2Ø EXECUTED"
 31  PRINT "3Ø EXECUTED"
1Ø1  PRINT "1ØØ EXECUTED"
```

It looked like a typical day during the French Revolution. Then I recalled that in computer jargon, EXECUTE meant CARRY OUT. "My word!" I gasped in admiration. "I see it all now! If it jumps to line 2Ø then the program will execute line 2Ø—a simple **READ** statement—and *then go on to 21*, the trace! And so line 21 will inform us that line 2Ø has been carried out! And the other jumps will be traced by the selfsame means! Holmes, that's *amazing*!"



Holmes sniffed—was he getting a cold? He typed **RUN**. I watched the words build up on the antique monitor.

```
20 EXECUTED
WHAT IS YOUR GUESS?
30 EXECUTED
? 43
TOO LOW
WHAT IS YOUR GUESS?
30 EXECUTED
? 55
TOO HIGH
WHAT IS YOUR GUESS?
30 EXECUTED
? 53
TOO LOW
WHAT IS YOUR GUESS?
30 EXECUTED
? 54
TOO HIGH
WHAT IS YOUR GUESS?
30 EXECUTED
?
```

"Holmes! *It never made the jump to line 100*! But it *must*, in order to end the game!"

Holmes merely smiled the faintest trace of a smile.

"But—confound it, Holmes—line 7Ø will *force* it to jump to line 1ØØ! Look! **IF** GNUMBER = CNUMBER **THEN GOTO** 1ØØ! So how can it have failed to get there, Holmes?"

Holmes refurrowed his brow. I stared in horror at the program. "It must, it must, it *must*! It—" Holmes interrupted my train of thought.

"Unless. . ." he said quietly.

"Unless *what*?" I cried in anguish.

"Unless it never gets to line 7Ø, Watson."

I felt as though a yawning pit had opened beneath my very feet. "My God, Holmes, that's it! Of course! I never—" But Holmes was typing once more:

```
69  IF GNUMBER = CNUMBER THEN PRINT
    "7Ø EXECUTED"
```

I stared at the listing. "Why 69, Holmes? Why not 71?"

Holmes sighed—another affliction I have noted often of late.

"Watson, if it ever has GNUMBER = CNUMBER and reaches line 7Ø—as it should—it will *jump* forthwith to line 1ØØ. Line 71 would never be executed. But line 69 will do the job admirably."

What a genius that man is! I feel privileged to breathe the same air. Except, perhaps, when he is smoking that confoundedly smelly pipe.

Holmes ran the program anew. The print-out was much as before. In fact, the print-out was *exactly* as before! Line 7Ø was never being reached!

Holmes stood awhile in uffish thought. "But we know," he mused, "that lines 5Ø and 8Ø are being executed, because of the messages TOO LOW and TOO HIGH. I *wonder*. . "

I waited with bated breath (but failed to catch anything).

"Let us consider, Watson, lines 5Ø and 6Ø. What if we input the *correct number*, 54? Then GNUMBER and CNUMBER are both 54, so the condition:

```
GNUMBER>CNUMBER
```

is false. And that. . ."

"Takes it on to line 7Ø," I said. "But we have already established that it never *gets* to line 7Ø, Holmes."

"When you have eliminated the impossible, Watson, then whatever remains, however improbable, must be the truth."

"You mean the computer's bust?"

"No, Watson, I most certainly do not! If GNUMBER *equals* CNUMBER then... aaaaaaaaahhhhhhhhhh!"

"What is it, Holmes?"

"Our analysis was at fault. Lines 5Ø and 6Ø do *not* say:

IF GNUMBER > CNUMBER...

They say:

IF GNUMBER >= CNUMBER...

Yes, that will be it."

I failed dismally to comprehend. "Can such a small change be responsible for so great a failure, Holmes?"

"Oh, indeed. Typical, and very likely indeed, indeed. You see, Watson, if we *change* lines 5Ø and 6Ø to read:

IF GNUMBER > CNUMBER...

then when we input the correct answer 54 for CNUMBER, line 5Ø becomes *false*. So we continue to line 6Ø, also false. And thence..."

"To line 7Ø! And then 1ØØ!"

"Exactly, Watson."

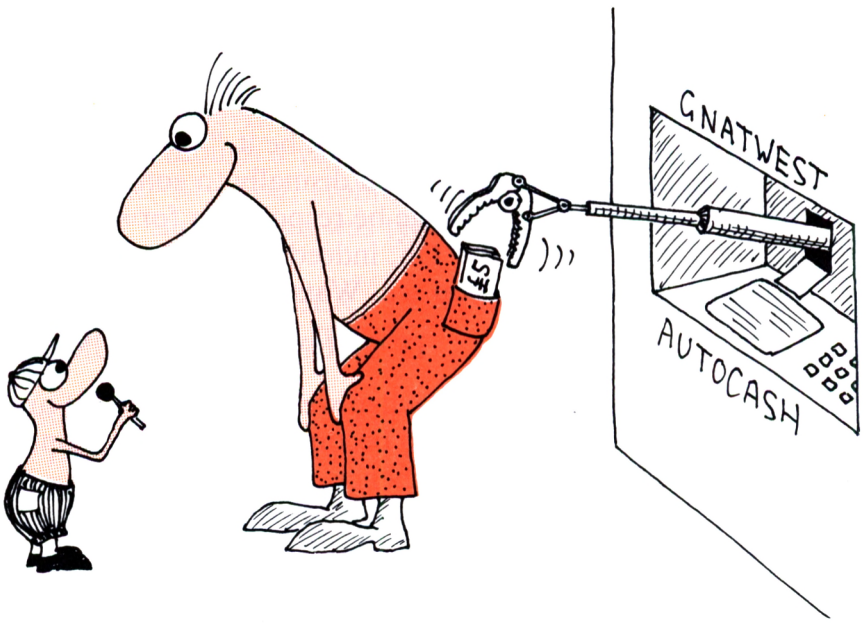
Holmes changed lines 5Ø and 6Ø, and removed the traces of his intervention. The program functioned perfectly. I wondered briefly how he would manage without my able assistance, but was too modest to voice the thought.

GNATWEST STRIKES AGAIN

Carlton Q. Cashsnitcher provides a computer banking service to his best customers. He has written a program to let them input their previous balance, list all debits (cash taken out) and credits (cash put in), and read off their new balance.

He is particularly proud of a trick that he uses in lines 9Ø and 13Ø, using a nonsensical input Ø to break out of an otherwise endless loop of inputs. This technique is known as *using a delimiter*, see Chapter 5.

He is not so proud of the program's record in actual use, because it hardly ever gives the right answer. Worse, the errors are sometimes in the customer's favour. So he has called in Despairing Dan the Debugging Man (you) to put the program right. Can you do it?




```
10  PRINT "GNATWEST SAVINGS ACCOUNT"
20  PRINT "INPUT PREVIOUS BALANCE"
30  INPUT BAL
40  LET PBAL = BAL
50  PRINT "INPUT DEBITS ONE BY ONE"
60  INPUT D
70  LET PBAL = PBAL - D
80  GOTO 30
90  IF D = 0 THEN GOTO 100
100 PRINT "INPUT CREDITS ONE BY ONE"
110 INPUT E
```

```
120  LET PBAL = PBAL + D
130  IF E = 0 THEN GOTO 150
140  GOTO 110
150  LET FBAL = BAL
160  PRINT "FINAL BALANCE IS ";
170  PRINT FBAL
```

Test lines tell you what's going on.

Traces tell you where a jump has gone.

ANSWERS

Gnatwest Strikes Again

There are errors in lines 70, 80, 90, 120. These should read:

```
70   LET BAL = BAL - D
80   IF D = 0 THEN GOTO 100
90   GOTO 60
120  LET BAL = BAL + E
```

5 Things with Strings

A lot of people think of computers as machines that handle *numbers* and do long, complicated sums. This is known in the Trade as *number-crunching*, and computers are certainly very good at it. But only scientists really *need* number-crunching. Computers can do all sorts of other things: draw pictures, keep records for businessmen, control machinery, or write letters to people.

Already you've seen some programs that don't crunch numbers. Programs that print things, programs that look for things. Programs like these are possible because computers can crunch all sorts of symbols, not just numbers. They deal with *information*: facts, figures, anything that can be put into symbols. The design for a bridge, the text of *Lord of the Rings*, the score of a Beethoven symphony, or an advertisement for deodorant.

Lots of businesses now use computers as 'intelligent typewriters' or *wordprocessors*. Here the computer is used to crunch words and letters. It can correct spelling mistakes, change errors, and print out neat copies automatically.

The fundamental idea needed for this kind of work is that of a *string*. Let's remind ourselves from *Book 1:*

> A *string* is a load of characters *strung* together.
>
> String variable names must end with a $ sign.

When you set up a string variable using a **LET** command, you put the value of the string variable in *quotes*, like this:

```
LET A$ = "FRED"
LET CAT$ = "FELIX"
LET G$ = "BACH"
```

and so on.

> The quotes round a string tell you where it starts and where it stops. They are *not* considered to be *part* of the string.

In fact the quotes are like a pair of bookends: these hold the books in place, but aren't books themselves, right? And quotes hold the string in place, but aren't characters *in* the string.

HOW LONG IS A (PIECE OF) STRING?

To find out how long a string is, that is, how many characters it contains, you use the keyword:

LEN

in the form:

LET variable = **LEN**(string)

For example:

```
LET X = LEN("FRED")
LET L2 = LEN(CAT$)
```

In these cases, X will take the value 4 because the string 'FRED' has 4 letters. And L2 is a variable with value 5, because the string variable CAT$ has value 'FELIX' and there are 5 letters in 'FELIX'.

Note the *brackets*:

LEN (string)

The keyword **LEN** is a new kind of keyword. It's not a command: you can't tell the computer:

```
10 LEN something or other
```

All you can do is *use* **LEN** to set the value of some variable. **LEN** is called a *function*.

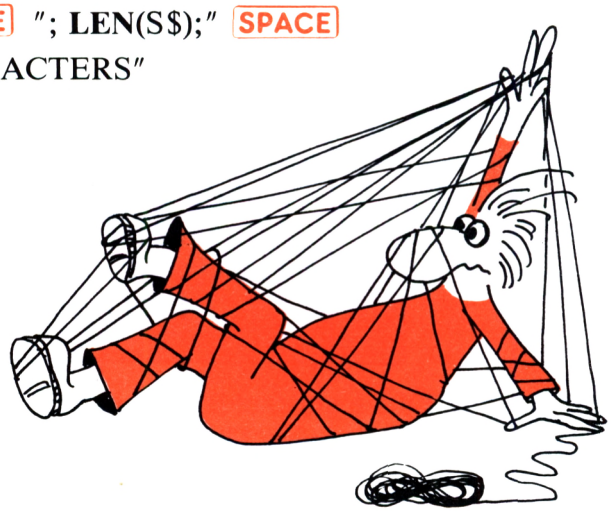
> If you plug a variable into a *function*, it gives you a value, associated to that variable by some definite rule.

Here the rule is 'count how many characters occur'. If you plug a *string* (or string variable) into the slot between the brackets in **LEN** ( ), the result is a *number*.

> If a string contains SPACE s, they still get counted by **LEN**. For instance, **LEN**("TOP SPACE CAT") is 7, because SPACE is counted as a character.

For example, this test program lets you input a string, and tells you how long it is.

```
10 PRINT "INPUT STRING"
20 INPUT S$
30 PRINT "YOUR STRING CONTAINS
   SPACE "; LEN(S$);" SPACE
   CHARACTERS"
```



STRINGING STRINGS TOGETHER

We've already used the sign:

+

to mean 'add two numbers'. But it has another meaning which applies to strings. The expression:

A$ + B$

now means 'jam A$ and B$ end to end'. For instance,

"GREEN" + "FLY" makes "GREENFLY"

The fancy name for this is to *concatenate* A$ and B$.

DRILL PROBLEMS

1. What are the following strings?

   (a) "TOOTH" + "PASTE"
   (b) "SHUF" + "FLE"
   (c) "BUG" + "LER"
   (d) "C" + "ANT" + "ER" + "BURY"
   (e) "X" + "Y" + "L" + "O" + "PHONE"

2. Find as many ways as you can to write:

   "SNOUT"

   by concatenating (adding) smaller strings.

3. **RUN** these two programs:

   (a) 10 **PRINT** "HOUSE" + "BOAT"

   (b) 10 **PRINT** "BOAT" + "HOUSE"

   Are the results the same? What *are* they?

4. Find two strings A$ and B$ such that:

   A$ + B$ = "SLUNG"

   B$ + A$ = "LUNGS"

KOMPUTERS DON'T MAKE SPELLING MISTAKES

Norton Greege-Whirdly, the Managing Director of Unclear Selectronics Inc., has written a 100-page document outlining the company's plans for future expansion. Unfortunately he discovers, very late in the day, that he has spelt a word incorrectly throughout. Nobody had ever told him that KOMPUTER wasn't quite right.

Fortunately his program library inkludes a komputer program (I regret that Greedge-Whirdly wrote this sektion for me, and it's too late to korrekt his text now) which will put everything straight.

```
10  INPUT W$
20  FOR T = 1 TO 20
30  PRINT " SPACE ";
40  NEXT T
50  IF W$< >"KOMPUTER" THEN PRINT W$
60  IF W$ = "KOMPUTER" THEN PRINT
    "COMPUTER"
70  GOTO 10
```

This lets him type in the dokument one word at a time, and korrekts the spelling. The corrected version is printed down the right-hand half of the screen. (With more fiddling you can suppress the input messages, but this is beyond the scope of this volume.)

Try it out on the first sentence in Greege-Whirdly's document:

```
UNCLEAR SELECTRONICS' NEW KOMPUTER
DIVISION HAS JUST PERFECTED THE ZZUB
HOME KOMPUTER SYSTEM.
```

You should have noticed that there's a new symbol in the program:

```
< >
```

This means 'is *not* equal to'. It is obtained by typing:

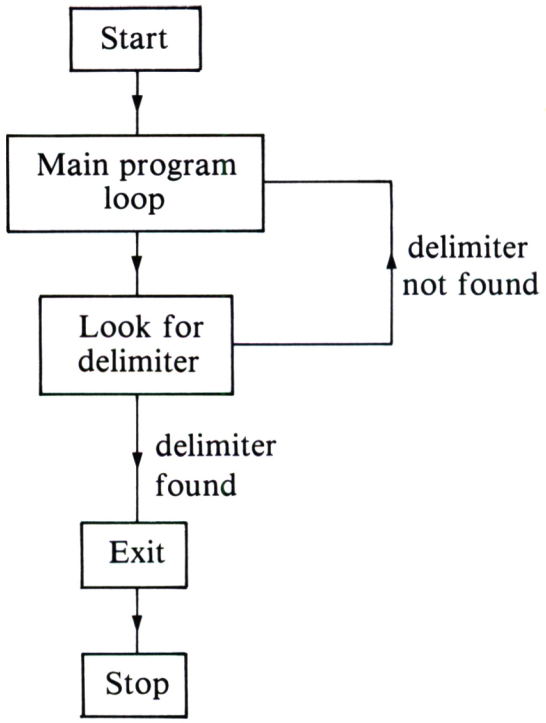
I don't want to give you the impression that real wordprocessors are as clumsy as this. You don't have to type the document in one word at a time! It will have been prepared using the computer, and will *already be stored in memory*. The computer can be told to search right through the document, looking for occurrences of KOMPUTER and changing them to COMPUTER. But the principle is the same.

DELIMITERS

Because of the **GOTO** 1Ø, the above program goes on forever. You'd have to switch off to stop it. To allow you to stop it whenever you feel like it, you can borrow a trick from Carlton Q. Cashsnitcher, and use a *delimiter*. This is any particular string that wouldn't occur in an ordinary text. Say 'XXXXX'. Modify line 7Ø to read:

```
7Ø  IF W$ < >"XXXXX" THEN GOTO 1Ø
```

Now, if you *don't* input the delimiter 'XXXXX' then the program goes to line 1Ø for the next word. But if you *do* type 'XXXXX', then it exits the loop, and stops.

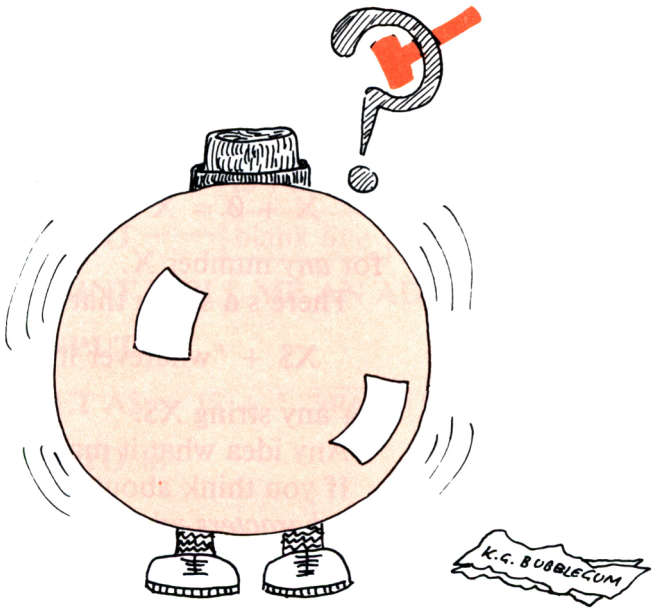


A *delimiter* is a special input that tells the computer to do something different—often to exit from an input loop.

SPY IN THE SKY

Comrade Procnatakadski, the spy, has discovered the secret plans of an American Bubblegum Factory and is sending it back to Moscow in the hope that Russian scientists can invent a bubblegum big enough to blow up the whole world. He conceals the message in a program.

```
10  LET A$ = "DSKI"
20  LET B$ = "Y [SPACE] BY [SPACE] "
30  LET C$ = "IVING"
40  LET D$ = "S [SPACE] ARR"
```




```
50  LET E$ = "PLAN"
60  LET F$ = "ROCN"
70  LET G$ = "OVE [SPACE] P"
80  LET H$ = "DA"
90  LET I$ = " [SPACE] SUN"
100 LET J$ = "E [SPACE] AT"
110 LET K$ = "AKA"
120 LET L$ = "SET [SPACE] L"
130 LET M$ = "AT"
200 PRINT "TOP SECRET CODE MESSAGE"
210 PRINT "ABOUT 50 MEGATON
    BUBBLEGUM"
220 PRINT E$ + D$ + C$ + I$ + H$ + B$ + E$ + J$ +
    I$ + L$ + G$ + F$ + M$ + K$ + A$
230 PRINT "RUN AND DESTROY"
```

When are the plans going to arrive? Should the KGB send agents to the bus station, railway station, or the airport to pick them up?

THE EMPTY STRING

When you're working with numbers, the number Ø is very useful. And it has a special property:

X + Ø = X = Ø + X

for *any* number X.

There's a string that has a similar property:

X$ + "whatever it is" = X$ = "whatever it is" + X$

for any string X$.

Any idea what it may be?

If you think about it, it has to be a string that contains *no characters whatsoever*. This curious beast is called the *empty string*, and it is written (logically enough) as:

""

That is, two quotes with nothing in between. (An empty bookcase consists of two bookends with nothing in between, right?)

The 'Computavote' program in Chapter 1 records the votes by keeping a 'running total' for each candidate – the variables are given the initial value Ø, then incremented in a loop as the votes are cast. The empty string "" is often used in the same way, in a program that uses a loop to build up a complicated string. For instance:

```
1Ø  LET E$ = ""
2Ø  LET E$ = E$ + "TICK"
3Ø  PRINT E$
4Ø  GOTO 2Ø
```

Try to guess what this does, then **RUN** it and see if you were right.

BABOON BUILD-UP

The next program uses the empty string "" and concatenation + to let you play a word-game with the computer.

```
10  LET Y$ = "YESTERDAY I SAW A SPACE "
20  LET B$ = "BABOON"
30  LET A$ = ""  <-- empty string
40  PRINT Y$ + A$ + B$
50  PRINT  <-- blank line
60  PRINT "TELL ME AN ADJECTIVE"
70  INPUT I$
80  LET A$ = I$ + " SPACE " + A$
90  GOTO 40
```

RUN this. Every time you are asked for an adjective, input one. (An adjective is a word that describes things—like:

GREEN
ECCENTRIC
RUBBERY
INTERCONTINENTAL
WILY

and so forth.) The computer builds up a description of your baboon by adding all the strings together. If you input the above in turn, it will end up by printing:

YESTERDAY I SAW A WILY
INTERCONTINENTAL RUBBERY ECCENTRIC
GREEN BABOON

See how long a sentence you can build. (The computer may stop when it gets too long for its string-handling system. Tough.) I got as far as:

YESTERDAY I SAW A MAGNIFICENT HAIRY
COLLAPSIBLE CHEAP FRAGRANT
INCOMPREHENSIBLE VAST SWEATY
INVISIBLE WILY INTERCONTINENTAL
RUBBERY ECCENTRIC GREEN BABOON

But I bet you can do better!

ANSWERS

Drill Problems

1. (a) "TOOTHPASTE"
   (b) "SHUFFLE"
   (c) "BUGLER"
   (d) "CANTERBURY"
   (e) "XYLOPHONE"

2. Sixteen ways!

```
"SNOUT"
"SNOU" + "T"
"SNO" + "UT"
"SNO" + "U" + "T"
"SN" + "OUT"
"SN" + "OU" + "T"
"SN" + "O" + "UT"
"SN" + "O" + "U" + "T"
"S" + "NOUT"
"S" + "NOU" + "T"
"S" + "NO" + "UT"
"S" + "NO" + "U" + "T"
"S" + "N" + "OUT"
"S" + "N" + "OU" + "T"
"S" + "N" + "O" + "UT"
"S" + "N" + "O" + "U" + "T"
```

3. No. (a) is "HOUSEBOAT" (b) is "BOATHOUSE"

4. A$ = "S" B$ = "LUNG"

Spy in the Sky

The message that the program prints out is:

> PLANS ARRIVING SUNDAY BY PLANE AT
> SUNSET LOVE PROCNATAKADSKI

So the plans arrive sunset on Sunday, and the KGB should send its agents to the airport.

The Empty String

It prints things like:

> TICK
> TICKTICK
> TICKTICKTICK
> TICKTICKTICKTICK

until space runs out or you get fed up and put it out of its misery by switching off.

Array of Sunshine

It is 5 o'clock in the morning on a cold, winter's day. As I lie in bed I can hear the next door neighbours clambering about on the flat roof of their garage. The occasional flash of a torch beam stabs through a gap in the curtains into my bedroom.

"Pass the pluviometer, Minnie."

"In a moment, Max. I seem to have got my foot stuck in something. And the barograph paper has wrapped itself around my neck."

"The anemometer's bent, must have been the hurricane."

"Or Mrs Tankwimple's cat, after pigeons again."

"Come on, hand me the pluviometer, woman!"

"It's the confounded pluviometer that my foot is stuck in, you clod. I told you not to put it right where somebody might tread in it!"

"Well, there's not much room up here, what with the sonic transponder system, two old bicycles, and a box full of dead begonias. Where else do you imagine I could – "

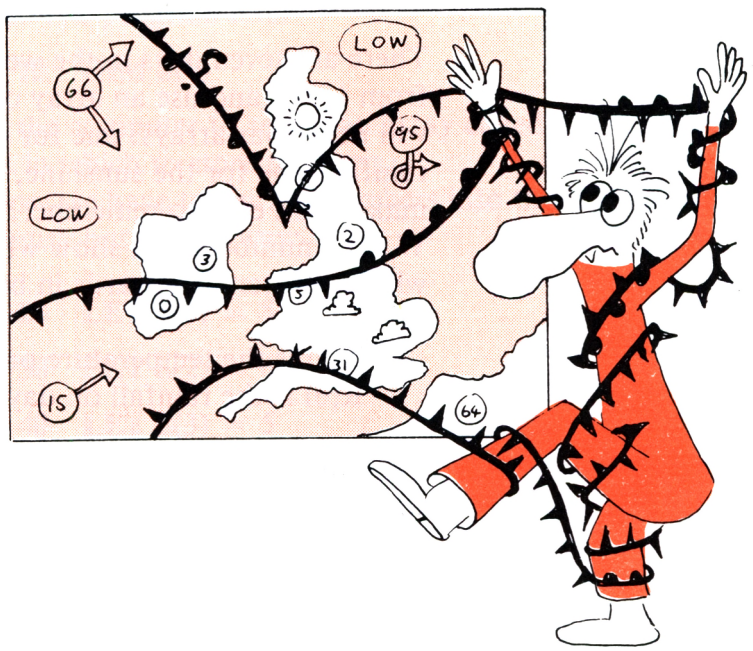
It is Minnie and Max Monsoon, the Merciless Meteorologists. Every morning they go out and measure the weather. There is an enormous splash. I think Max just tripped over the bicycles and fell into the water-butt beneath. I hope he wasn't carrying the barometer. They say a falling barometer is a sign of bad weather.

There must be a better way to make recordings of what the weather is doing. Why don't they *automate* their weather-station? Then they could use a computer to analyse the recorded data. I open the window and offer this suggestion to them. There is a stunned silence, followed by an outraged cry from Minnie Monsoon:

"You and your confounded computers! Bleep-bleep-bleep at all hours of the night! Zap! Powww! Kaboom! People can't get a decent night's sleep around here!"

I duck as the remains of an ancient turnip hit the wall inches from my nose. I close the window in triumph.

I think I interested them in the idea.



DATA STORAGE

Obviously I need to think this through before I approach them again. One problem will be storing the recorded data in a form that the computer can use. I suppose I *could* use **DATA** lists, but those aren't designed to be modified easily. I think I'll use an *array*.

An array is a numbered list. It has a name, just like a variable does, but it also has a *number* which tells you whereabouts you are in the list.

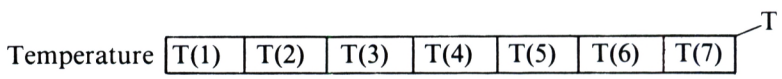
Here's a week's data from the Monsoon's weather station.

	SUN	MON	TUE	WED	THU	FRI	SAT
Temperature (°C)	15	3	0	8	7	2	11
Rainfall (cm)	0	1	1	0	2	3	0
Sunshine (hr)	8	6	5	2	0	1	9
Windspeed (kph)	24	93	11	14	6	2	18

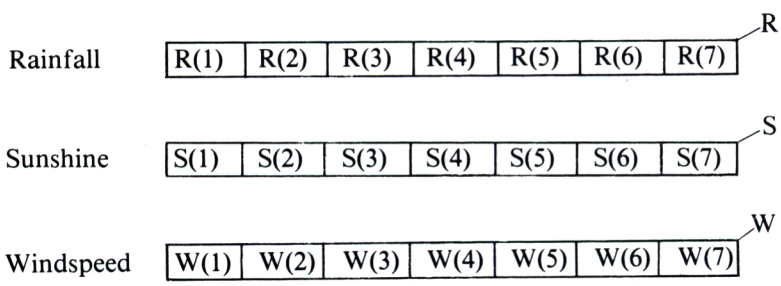
There are seven days in the week, so I could number them from 1 to 7 and use an array of size 7 to hold each item. I will need four arrays: one for the temperature, one for the rainfall, one for the sunshine, and one for windspeed. To make them easy to remember, I'll name them T, R, S and W. The numbers that show whereabouts in the list we are will be written afterwards in brackets. For instance,

T(3) is the temperature on day 3 (Tue), which is 0.
R(6) is the rainfall on day 6 (Fri), which is 3.

Then the whole set of data will be stored in the computer something like this:



which is *one* array of size 7; and three more:



The question is, how can we do this in BASIC?

SETTING UP AN ARRAY

To make life easier, let's concentrate on just one array: the Sunshine array, S.

First we have to tell the computer to make room in memory for the array, ready to store the numbers. This is called *dimensioning* the array, and it uses the command **DIM**. The program line:

```
10 DIM S(7)
```

tells the computer to set up an array called S, with seven slots in which to store numbers.

Now we have to get the numbers into those slots. Here is a clumsy but simple way:

```
20 LET S(1) = 8
30 LET S(2) = 6
40 LET S(3) = 5
50 LET S(4) = 2
60 LET S(5) = 0
70 LET S(6) = 1
80 LET S(7) = 9
```

An *array* holds a list of numbers. Before using an array, you should tell the computer how big the list will be by using the DIM command. This is called *dimensioning the array*.

TRANSFERRING FROM A DATA LIST

Another way is to use a **DATA** list, and transfer it automatically into an array, where it will be easy to manipulate. Like this:

```
10 DATA 8, 6, 5, 2, 0, 1, 9
20 DIM S(7)
30 FOR N = 1 TO 7
40 READ X
50 LET S(N) = X
60 NEXT N
```

You can even replace lines 40 and 50 with just:

```
40 READ S(N)
```

to save time.

GETTING DATA FROM AN ARRAY

Whichever of these methods you choose, we're now ready to do some work with the array. Our first task is to write a program that will allow us to specify a day number, and tell us the amount of sunshine. That's easy. Type lines 10–80 of the clumsy version (or lines 10–60 of the DATA version) above, and add:

```
100 PRINT "SPECIFY DAY NUMBER (1–7)"
110 INPUT DAY
120 PRINT "THE AMOUNT OF SUNSHINE
    WAS";
130 PRINT S(DAY)
```

Now, let's just make sure you understand what's going on. Suppose you want to know how much sunshine there was on Thursday. That's day number 5. So at line 11Ø, after the prompt in line 1ØØ, you should input the number 5.

The computer then sets the variable DAY to the value 5.

Now, in line 13Ø, the computer looks for S(DAY). First it finds the value of DAY, which is 5; so now it knows it must look for S(5). But line 6Ø of the 'clumsy' program above (or the fifth entry in the DATA list version) tells it that S(5) is equal to Ø. So the output you get is:

THE AMOUNT OF SUNSHINE WAS Ø

In a way, what the array does is to give you a set of seven different variables S(1), S(2), . . ., S(7). But more than that: it lets you refer to any one of them as S(N), where N can be any number in the range 1–7. A program can then manipulate N, and so deal with different choices from the seven variables, depending on the value N has been set to.

Without arrays you can still have seven variables S1, S2, . . ., S7, but when you ask the computer to look at SN, it looks for a variable *with the name* SN. Even if it knows N is 4, it does *not* look for S4. Putting brackets round the N, and telling the computer that S is an array by dimensioning it, makes a tremendous difference!

DRILL PROBLEMS

1. What are the values of the following expressions, with S(1)–S(7) taking the values listed above?

   (a) S(1) + S(2)
   (b) S(3) – S(7)
   (c) 2 * S(6)
   (d) S(3) * S(7)
   (e) S(2 + 2)
   (f) S(1 + 2 + 3) * S(2 * 3) – 2 * S(2 * 2 + 1) + S(28 / 4)

   Work these out by hand, and then check them on the computer. (How?)

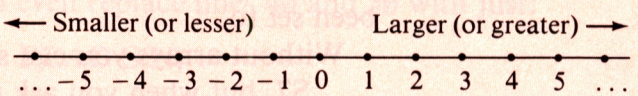
2. Find ways to write the numbers Ø, 1, 2, . . . , 9 in terms of S(1)–S(7). Again, do it by hand and then use the computer to check.

GETTING THE WIND UP

Write a program that will store a week's windspeed data in an array W, and let the user select a particular day and find out what the windspeed was.

Numerical Order

This box is just to remind you about the way numbers can be given an order, so that we can speak of the *greater* or the *smaller* of two numbers. If you think of the numbers as being arranged in a line, then the greater numbers are to the right and the smaller ones to the left.



There are some standard symbols that the computer uses to deal with the order of two numbers:

M = N means M is *equal to* N. (You know that one, naturally!)

M < > N means M is *not equal to* N. (Examples: −2 < > 4 is a true statement; 2 + 2 < > 4 is false.)

M > N means M is *greater than* N. (Examples: 5 > 4 is true, 3 > −4 is true, −4 > −5 is true; −5 > −4 is false, 2 > 7 is false.)

M < N means M is *less than* N. (Examples: 2 < 3 is true, −1 < 4 is true, −3 < −2 is true; 3 < 2 is false, −1 < −99 is false. Note that M < N means the same as N > M.

M >= N means M is *greater than or equal* to N. (Examples: 3 >= 2 is true, 3 >= 3 is true; 7 >= 9 is false.)

M <= N means M is *less than or equal to* N. (Examples: 2 <= 7 is true, −3 <= −3 is true, 5 <= 5 is true; 6 <= 5 is false, 999 <= 73 is false.) Note that M <= N means the same as N >= M.)

The same goes for decimal numbers as well as whole numbers.

That's simple enough to follow easily, but not really terribly fascinating. If that was all we could do with arrays, it wouldn't be worth the bother. But computers don't just *store* data: they process it.

What sort of things might the Monsoons want to know? They could include:

1. The total sunshine for the week.
2. The average sunshine per day.
3. The day with the largest amount of sunshine (or days if more than one), and how much that was.
4. Ditto for the least sunshine.

Then there might be more fancy questions, like:

5. On which days of the week were there between 1 and 5 hours of sunshine?
6. Were there any days with no sunshine at all? If so, which?
7. What was the average sunshine at weekends (Saturday and Sunday)?
8. Was there more sunshine on Sunday than on Saturday?

Once the data is stored in the computer as an array, it is possible to write short programs to find the answer to each of these questions. Make sure that *only* the data-loading routines (lines 1Ø–8Ø, *or* lines 1Ø–6Ø on pages 53 and 54) are in the machine, and then add the relevant group of lines below.

Total Sunshine

```
100 LET SUM = 0
110 FOR N = 1 TO 7
120 LET SUM = SUM + S(N)
130 NEXT N
140 PRINT "TOTAL SUNSHINE IS";SUM;
    "HOURS"
```

Average

This one's your job. The *average* of a set of numbers is their total, divided by how many numbers there are. For instance, take the set of numbers:

1, 9, 7, 2, 2, 3

Their total is 24 because 1 + 9 + 7 + 2 + 2 + 3 = 24. There are six of them. So the average is given by:

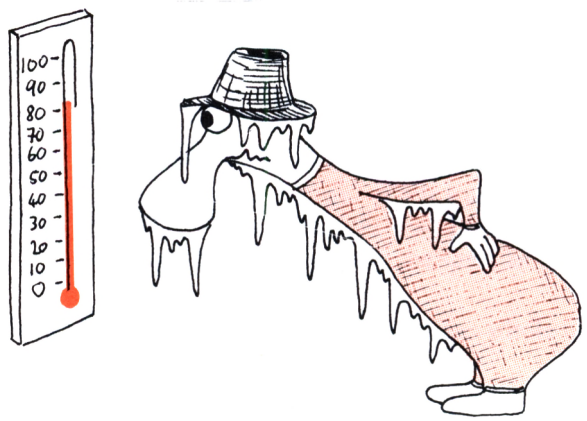
$$\text{average} = \frac{\text{total}}{\text{how many there are}} = \frac{24}{6} = 4$$

Can you change *one line* only in the above 'Total Sunshine' program so that it prints out the average sunshine per day?

Maximum Sunshine

To keep things simple, let's start by trying to find the largest amount of sunshine, without worrying what day or days it happens on. The idea is to start with a variable, let's call it MX, and to increase it whenever we find a value of S(N) that is larger. (Recall from page 56 that A > B means 'A is greater than B'.) This is the program:

```
100  LET MX = 0
110  FOR N = 1 TO 7
120  IF S(N) > MX THEN LET MX = S(N)
130  NEXT N
140  PRINT "MAXIMUM AMOUNT OF
     SUNSHINE IS";MX
```



To check that this really works, let's *dry-run* it (see *Gateway to Computing Book 1*, Chapter 9).

Here's the dry-run table showing how N, S(N) and MX change as we go round the loop:

Line Number	MX	N	S(N)	S(N) > MX?
100	0	—	—	
110	0	1	8	Yes
120	8	1	8	
130	8	2	6	No
120	8	2	6	
130	8	3	5	No
120	8	3	5	
130	8	4	2	No
120	8	4	2	
130	8	5	0	No
120	8	5	0	
130	8	6	1	No
120	8	6	1	
130	8	7	9	Yes
120	9	7	9	
130		exit loop		
140	9 ←	the required maximum		

QUICKIE

Would this method work if line 100 had been **LET** MX = 10? If so, why? If not, why not? What are the 'good' starting values for MX if you want to use this technique?

Minimum Sunshine

Your job again! Think how MX works and adapt it to a variable MN that stores the smallest number found so far. *Be careful what value you start MN at!* Ø is *not* a good idea. Recall that A < B means 'A is less than B'.

On which Days?

Having found the maximum amount of sunshine, we can do another search through the array to find the day(s) on which it occurs. Keep lines 1Ø–8Ø (or 1Ø–6Ø) and the *Maximum Sunshine* program (lines 1ØØ–14Ø, page 58). Add:

```
2ØØ  FOR N = 1 TO 7
21Ø  IF S(N) = MX THEN PRINT"OCCURRING
     ON DAY";N
22Ø  NEXT N
```

Now that we *know* the value of MX, this just checks each day in turn to see if that amount of sunshine happened, and if so, it prints a message to say so.

Fancy Stuff

Programs to answer questions 5, 6, 7 and 8 are left to you as a problem.

BIG STUFF

Of course, you really wouldn't go to all this trouble just for one week's sunshine figures. But what about 1Ø years? That would be 365Ø days (ignoring leap years; 3653 at most if they are included); and all you have to do is change the 7s to 365Øs throughout. The array would probably be set up using an input loop, and everything would be saved on a permanent memory device like tape or floppy disc once this was done. I can't go into that sort of technique at present; but it's when the numbers get big that the ideas in this chapter tend to pay off.

There are plenty of uses for small arrays in programs, too: they're one of the most versatile weapons in the programmer's armoury. You'll see plenty of them as we proceed.

COMPUTER EXPERIMENT

Here is an experiment you can do to illustrate how weather information could be gathered and processed automatically. In the absence of lots of fancy electronic equipment, however, you will have to play the part of the automatic recording devices yourself.

Collect weather data for a month. Either take a thermometer and measure the outside temperature every day at the same time, or write down what the newspaper says the temperature was in (say) Reykjavik. (Or cheat and use the data list given below.) You will end up with about 30 temperature values. The following program will tell you the average, maximum and minimum temperatures. It can easily be adapted to allow you to deal with rainfall, air pressure, windspeed, and so forth. Most of the effort goes into recording the numbers and feeding them into the machine.

```
10  PRINT "TEMPERATURE ANALYSER"
20  PRINT                                  <- for a blank line
30  PRINT "HOW MANY VALUES RECORDED?"
40  INPUT D
50  DIM T(31)                              <- maximum no. of days in a month
60  PRINT "INPUT VALUES ONE BY ONE"
70  FOR N = 1 TO D                         <- input loop
80  PRINT "TEMPERATURE ON DAY";N           <- input loop
90  INPUT T(N)                             <- input loop
100 NEXT N                                 <- input loop
110 LET SUM = 0                            <- initialize values
120 LET MX = T(1)                          <- initialize values
130 LET MN = T(1)                          <- initialize values
```

```
140  FOR N = 1 TO D
150  LET SUM = SUM + T(N)
160  IF T(N) > MX THEN LET MX = T(N)
170  IF T(N) < MN THEN LET MN = T(N)
180  NEXT N
190  PRINT
200  PRINT
210  PRINT "AVERAGE TEMPERATURE:";
     SUM/D
220  PRINT "MAXIMUM TEMPERATURE:";MX
230  PRINT "MINIMUM TEMPERATURE:" ;MN
```

two blank lines for tidiness

Note the use of an *input loop* in lines 70–100: yet another way to feed data into an array. However, you have to type the numbers in every time you run the program. An alternative is to store them on tape as a *file*; but that's beyond the scope of this volume.

If you're bone idle and don't want to spend a month standing in the garden in rain, sleet, and snow, waving a thermometer, I just happen to have in my possession the data obtained by Minnie and Max Monsoon, for my home village. Here it is.

Temperature Chart

Observers: Minnie and Max Monsoon

Weather station: Flannel-under-Wave

Month: April 1984

Day	Temperature	Day	Temperature	Day	Temperature
1	11	11	15	21	10
2	13	12	14	22	11
3	14	13	13	23	16
4	9	14	10	24	17
5	21	15	9	25	16
6	16	16	13	26	20
7	15	17	11	27	19
8	15	18	8	28	20
9	3	19	2	29	14
10	12	20	7	30	16
				31	—

What was the average, maximum and minimum temperature in the sleepy village of Flannel-under-Ware, in April 1984?

RULES FOR ARRAY NAMES

The rules for array names are the same as those for variable names. For *numeric* arrays, that hold numbers, use names suitable for numeric variables. (There are such things as *string* arrays, but they are beyond the scope of this book.)

You do not need to dimension an array if it holds 1Ø or fewer items. Also, an array S has an entry S(Ø). These features are useful, but can be confusing at first, so I'd advise you to forget about them until you feel happy with everything else in this book.

> You can use any legal name for an array – but don't forget the brackets! TAKE5 is just another variable; TAKE(5) is an item in an array.

MOUSEBENDER'S MUSIC-HALL

Hortense Mousebender, and her husband Marmaduke Mousebender the Mad Mathematician, have made a fortune selling computer software. They have decided to invest it in a theatrical production, to be named:

MOUSEBENDER'S MULTICOLOURED
MUSIC-HALL

There will be 6 acts. The time taken by each is as follows:

1. Lucy Laine the Liverpool Lark	11 minutes
2. Mystro the Mediocre Magician	8 minutes
3. The Wormwood Scrubbs Quartet	13 minutes

4. Oo-La-La Can-Can Chorus 25 minutes
5. Seamus Android, the Irish Tenor* 2 minutes
6. Huge Harry, the Hopeless Humorist 7 minutes

They wish to store the times on their computer, and to work out the average time per act. How would you set about doing this?

Suppose they want the computer to find out which act is the shortest. What changes should be made to the program?

ANSWERS

Drill Problems

1. (a) 8 + 6 = 14
   (b) 5 − 9 = −4
   (c) 2 * 1 = 2
   (d) 5 * 9 = 45
   (e) S(4) = 2
   (f) S(6) * S(6) − 2 * S(5) + S(7) =
   1 * 1 − 2 * Ø + 9 = 1Ø

2. There are lots of ways. Here's one set:

Ø	S(5)
1	S(2) − S(3)
2	S(4)
3	S(4) + S(6)
4	S(2) − S(4)
5	S(3)
6	3 * S(4)
7	S(2) + 1
8	S(4) * (S(3) − S(6))
9	S(7)

To get the computer to check, add program lines like:

```
1ØØ PRINT S(1) + S(2)
```

to lines 1Ø−8Ø (or 1Ø−6Ø, page 54) which were used to set up the array.

* An Irish tenner is worth about £9.31 because of the exchange rate.

Getting the Wind Up

```
10  DIM W(7)
```

Don't forget to dimension the array!

```
20  LET W(1) = 24
30  LET W(2) = 93
40  LET W(3) = 11
50  LET W(4) = 14
60  LET W(5) = 6
70  LET W(6) = 2
80  LET W(7) = 18
90  PRINT "SPECIFY DAY NUMBER (1-7)"
100 INPUT DAY
110 PRINT "THE WINDSPEED WAS";
120 PRINT W(DAY)
```

Average

Change line 140 to:

```
140 PRINT "AVERAGE SUNSHINE PER DAY IS";
    SUM/7; " HOURS."
```

Quickie

No: MX would stay at 10 throughout. You need to choose the starting value nice and low, so that at least one S(N) is equal or bigger. In fact, a good bet is to start with:

```
100 LET MX = S(1)
```

Minimum Sunshine

```
100 LET MN = 1000
```

nice and large for a minimum calculation

```
110  FOR N = 1 TO 7
120  IF S(N) < MN THEN LET MN = S(N)
130  NEXT N
140  PRINT "MINIMUM AMOUNT OF
     SUNSHINE IS";MN
```

Line 100 could also be replaced by:

```
100  LET MN = S(1)
```

Fancy Stuff

You may have had to think quite hard to do some of these. Here are my solutions.

5.
```
100  FOR N = 1 TO 7
110  IF S(N) < 1 THEN GOTO 140
120  IF S(N) > 5 THEN GOTO 140
130  PRINT N
140  NEXT N
```

Note how lines 110 and 120 cause the computer *not* to print the day number N (in line 130) if the sunshine is outside the range 1–5 required.

6.
```
100  FOR N = 1 TO 7
110  IF S(N) = 0 THEN PRINT "NO SUNSHINE
     ON DAY";N
120  NEXT N
```

7.
```
100  LET AV = (S(1) + S(7)) / 2
110  PRINT "WEEKEND AVERAGE:";AV
```

8.
```
100  IF S(1) > S(7) THEN PRINT "MORE ON
     SUNDAY"
110  IF S(7) > S(1) THEN PRINT "MORE ON
     SATURDAY"
120  IF S(1) = S(7) THEN PRINT "THE SAME
     BOTH DAYS"
```

Remember Make sure lines 10–80 (or 10–60; pages 53 or 54) are in the machine before you RUN any of these; and make sure no lines from previous programs are still left apart from those specified.

This is best done by editing the unwanted lines out each time. If you use **NEW**, you'll have to retype those data input lines, which gets kind of boring on the seventeenth try.

Computer Experiment

```
AVERAGE TEMPERATURE:  13
MAXIMUM TEMPERATURE:  21
MINIMUM TEMPERATURE:  2
```

Mousebender's Music-Hall

```
10  DIM T(6)
20  LET T(1) = 11
30  LET T(2) = 8
40  LET T(3) = 13
50  LET T(4) = 25
60  LET T(5) = 2
70  LET T(6) = 7
100 LET SUM = 0
110 FOR N = 1 TO 6
120 LET SUM = SUM + T(N)
130 NEXT N
140 LET AV = SUM / 6
150 PRINT "AVERAGE TIME IS:";AV
```

To find the shortest act, change lines 100–150 as follows:

```
100 LET MN = T(1)
110 FOR N = 1 TO 6
120 IF T(N) < MN THEN LET MN = T(N)
130 NEXT N
```

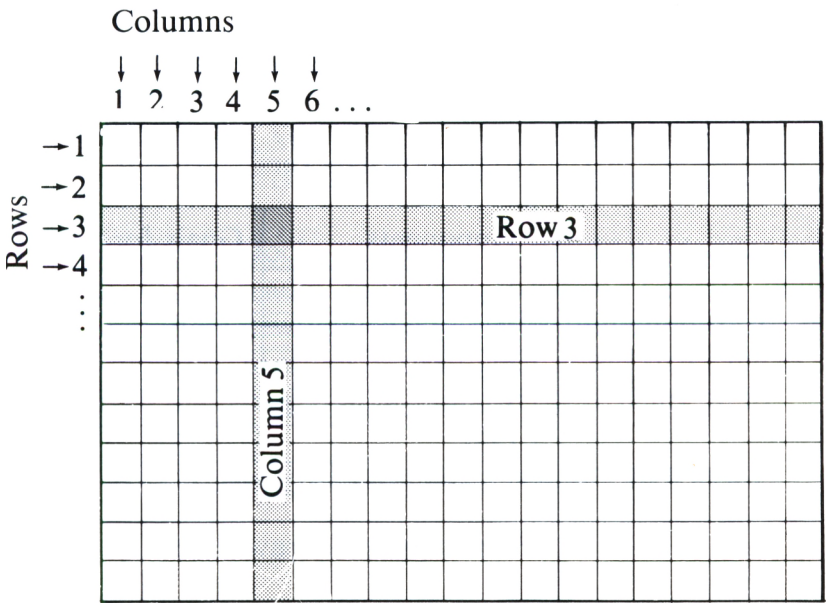
```
140  FOR N = 1 TO 6
150  IF T(N) = MN THEN PRINT "SHORTEST
     ACT IS NUMBER";N
160  NEXT N
```

7 Squire Stoatthrostle Picks up the TAB

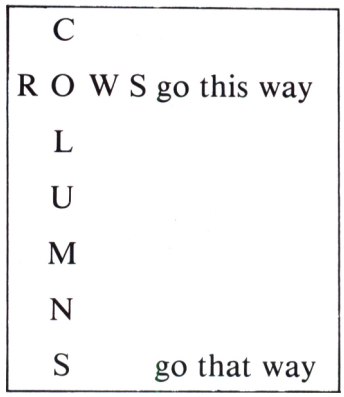
So far, we haven't worried very much about *where* the printing on the TV screen goes, just as long as the items we need appear when we need them. That's fair enough: when you start learning programming you don't want to be bothered with fiddly details.

The time has come (the Walrus said . . .) to take a look at one way to tidy up the screen display: *tabulation*.

Your TV display is divided into a number of cells, each of which holds one *character* (letter, number, graphics symbol, etc.). Lines of cells that run across the screen are *rows*; lines that run down the screen are *columns*.



On the CPC464 computer there are 25 rows and 40 columns. It is the columns that concern us in this chapter.



THE LUMMOXSHIRE LEAGUE

A dozen or so miles from Flannel-under-Ware is the tiny village of Hogwallow. Although it has a very small population (1980 census: 122), Hogwallow has one of the finest village soccer teams in the area – the *Hogwallow Hackers*. Every year the team plays in the local Lummoxshire League, against such renowned teams as Bumbleforth Benighted, Prongworthy Ravers, Cow Green Agricultural and Mechanical, Womblehampton Waverers, and the Gentle Elastic Company.

At the current stage in the season, the score sheet looks like this.

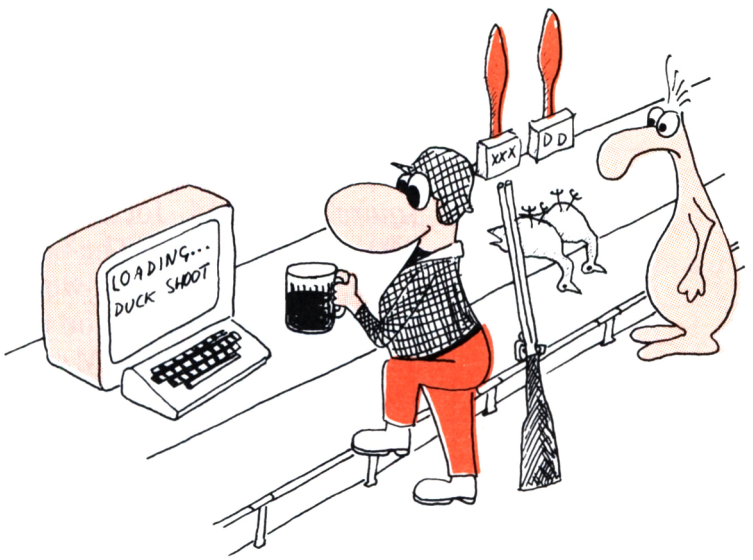
Lummoxshire League Official Score Sheet

Hogwallow Hackers

Opponents	Goals for	Goals against	Total points
Bumbleforth	7	Ø	2
Prongsworthy	2	2	3
Cow Green	4	3	5
Womblehampton	4	5	5
GEC	3	1	7

Points: Win 2, Draw1, Loss Ø.

It is 8.30 in the evening in the saloon bar of the *Paralytic Pig*, the local hostelry. The Hackers have just beaten the Twangers (the nickname for the Gentle Elastic Company) and spirits are high. Not to say over-priced. The Captain and the Secretary of the Hackers, Alf Thyme and Jock Strappe, are about to chalk up the latest win on the special blackboard that hangs beside the skittle table . . .



Alf: Oh-arrr, that were a good game, Jock!
Jock: Sure wuz, Alf. Now where be that ferdanged chalk? Oy, Mavis! Oo's nicked the chalk?
Mavis (The barmaid): Oi dunno, Jock. Bob Frapples 'ad it last noight fer the darts match against Bumbleforth 'B' team.
Jock: Tarnation, Mavis! That be the fifth toime young Bob's snaffled our chalk! 'Ow can oi mark up the footy score?
Alf: Yer know, Jock, moight be we should modernize our methods. Oi bin readin' in the *Goathandler's Gazette* 'bout 'ow one o' these newfangled compyuter thingummies is a-revolutionizin' livestock farmin'. An' it *did* strike me as 'ow maybe the 'ackers could put their score chart on to one.
Jock: Ridicklus, Alf. Puttin' a score chart on a goat, indeed!
Alf: No, Jock – on a compyuter!

Jock: That'd be a good one, wouldn'it, Mavis? Hur-hur-hur.
Alf: Nay, Jocko, oi be serious. Look, there be Squire Stoatthrostle. Oi hear he bought one of 'em a few weeks back. (Turns to Squire.) Hey, Squire!
Squire S: Oh, good evening, chaps. Yes please, Mavis, the usual. Double vodka and antifreeze.
Alf: Squire, oi do 'ear tell as 'ow you'm hacquired one o' them danged compyooooter doofers.
Squire S: That's right, a Stork-37 with dual disc drives and a 64K RAM.
Jock: Oi hear Ned Scraggitt bought a few dozen rams at Nerdsby market, too. But what's sheep-farmin' got ter do with compyuters?
Squire S: RAM, Jock, is Random Access Memory.
Alf: Y'see, Jocko? The Squire be a h'expert already! Y'see, Squire, we wuz a-wonderin' if it be possible to put the 'ackers' score-sheet on a compyuter, so we could see the results on the bar TV set.
Mavis: Not durin' *Consternation Street*, you don't! Ol' Mother Creepscuttle'll throw a fit!
Jock: That oi'd loike ter see! Do the ol' biddy a power o' good, it would. Whoi, oi do recall that toime when she . . .

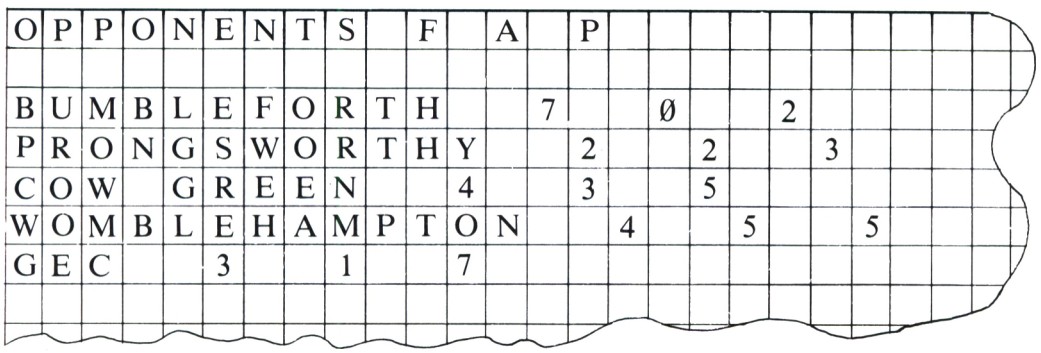
* * * * * * *

And talk drifted to other matters. But late that same evening, Squire Stoatthrostle decided it might after all be jolly good fun to have a go, don't y'know? After taking a crafty peek at Chapter 3 he decided to use **DATA** statements, and eventually came up with a program:

```
10 DATA BUMBLEFORTH, 7, 0, 2
20 DATA PRONGSWORTHY, 2, 2, 3
30 DATA COW GREEN, 4, 3, 5
40 DATA WOMBLEHAMPTON, 4, 5, 5
50 DATA GEC, 3, 1, 7
60 PRINT "OPPONENTS [SPACE] F [SPACE] A
   [SPACE] P"
70 PRINT
```

```
100 FOR N = 1 TO 5
110 READ O$, F, A, P
120 PRINT O$;"  SPACE  ";F;A;P
130 NEXT N
```

And the following evening, when Alf and Jock wandered into the *Paralytic Pig*, they were surprised to see on the TV screen:



The Squire was very proud of his program, but although Alf and Jock were very polite about it, he sensed that they weren't altogether satisfied with the result.

Why?

STOATTHROSTLE'S BLUNDER

Eventually the Squire realized what was troubling them. As Noddy Numskull the village idiot put it, the columns were 'loike t'hind leg of a rheumatic goat'. The table of results was out of alignment and very hard to read, not at all like the score sheet.

Stoatthrostle knew that something had to be done.

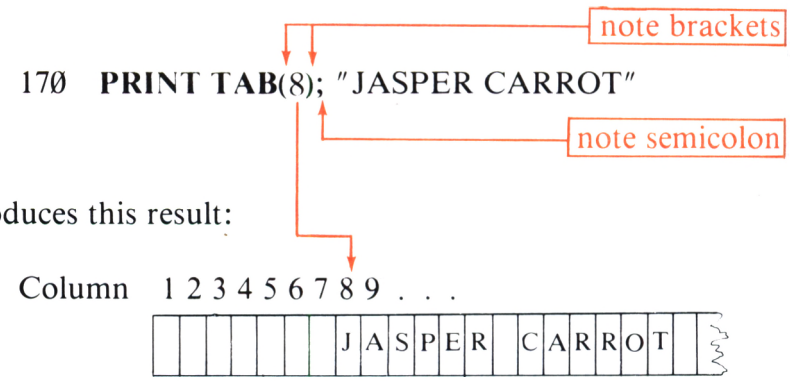
But what?

TAB

The answer is a new BASIC keyword:

TAB

This tells the computer to print something, and to *start it from a chosen column* on the TV screen. For instance,



In general, a command:

120 **PRINT TAB**(number); string

will print out the chosen string starting in the column given by the number. This will be in:

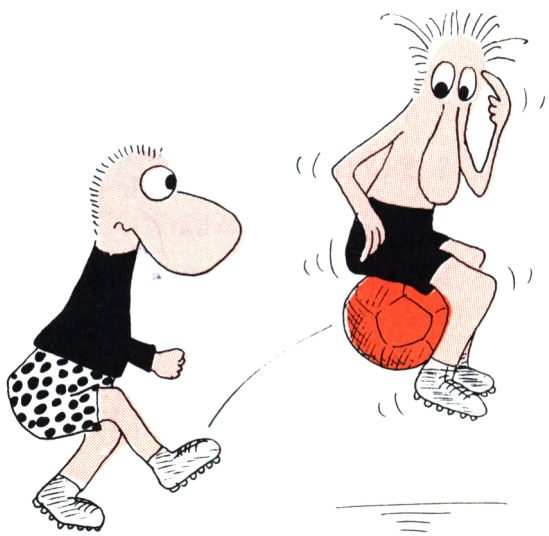
1. The current row of printing, if that column has not yet been reached.
2. The next row if it has.

Compare these two programs:

(a) 10 **PRINT TAB**(2);"TWO";**TAB**(12);"TWELVE"

(b) 10 **PRINT TAB**(12);"TWELVE";**TAB**(2);"TWO"

Note how in (b) the TWO is printed on the next line, because the print position has already moved beyond column 2.



To keep **PRINT**ing in tidy columns, use

TAB

in your **PRINT** statement. Don't forget the brackets.

STOATTHROSTLE RECONSIDERS

After dropping several heavy hints, the Hogwallow Hackers presented Squire Stoatthrostle with a copy of *Gateway to Computing Book 2*. An there, on page 69, the Squire found the chapter criticizing his program. (Hi, Squire! How's the wife?) By reading up on **TAB** he was able to come up with a dramatic improvement to his program. I would be happy to express gratitude for his solution, which is reproduced below, except for a vague suspicion that he may have pirated it from the copy of this book that was presented to him.

Anyway, here it is.

```
10  DATA BUMBLEFORTH, 7, 0, 2
20  DATA PRONGSWORTHY, 2, 2, 3
30  DATA COW GREEN, 4, 3, 5
40  DATA WOMBLEHAMPTON, 4, 5, 5
50  DATA GEC, 3, 1, 7
60  PRINT "OPPONENTS";TAB(20);"F";TAB(23);
    "A";TAB(26);"P"
70  PRINT
100 FOR N = 1 TO 5
110 READ O$, F, A, P
120 PRINT O$;TAB(20);F;TAB(23);A;TAB(26);P
130 NEXT N
```

The idea is simple: use columns 0, 20, 23, 26 as standard positions for the name of the opponents, goals for, goals against, and total points. It turns out that there's no need to use **TAB**(1) at the start of the **PRINT** command:

PRINT TAB(1);"OPPONENTS"; . . . etc.

because the computer automatically starts each line in column 1 anyway. Here's the result:

```
1                                                    20      23      26
OPPONENTS                                            F       A       P

BUMBLEFORTH                                          7       Ø       2
PRONGSWORTHY                                         2       2       3
COW  GREEN                                           4       3       5
WOMBLEHAMPTON                                        4       5       5
GEC                                                  3       1       7
```

Now it's all neat and tidy. The *Paralytic Pig* had to get a licence extension for the celebrations.

AGONY COLUMN

If you trusted me to be telling the truth, and didn't try the Squire's program out for yourself, I advise you to do so at once.

You see, *I lied*.

It doesn't quite work!

The numbers in the F, A, P columns all come out one space too far right. I'll explain why in a minute, but first let me demonstrate an important debugging principle:

> You don't always have to understand *why* something went wrong, in order to fix it.

To see this, let's just move everything one space to the left in line 12Ø, like this:

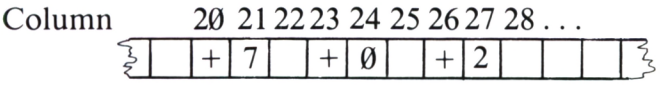
```
12Ø PRINT O$;TAB(19);F;TAB(22);A;TAB(25);P
```

all **TAB** numbers reduced by 1

Try again: gee whillikers, it works! It's a bit chewing-gum-and-stringy, but it *works*!

Point taken. But, of course, *ceteris paribus* and *ipso fatso*, it would be much more satisfactory to know *why* stuff came out in the wrong place.

The version of BASIC used on your computer is rela ed to an industry standard called *Microsoft BASIC*. In Microsoft BASIC all *strings* will be printed where you expect them to be, when **TAB** is used. But *numbers* may not. This is because Microsoft pretends that any number which isn't negative has an 'invisible' plus (+) sign in front of it. (Numbers which *are* negative have a distinctly visible minus sign!) So instead of printing 7, Ø, 2 in columns 2Ø, 23, 26 the machine puts their '+' signs there:



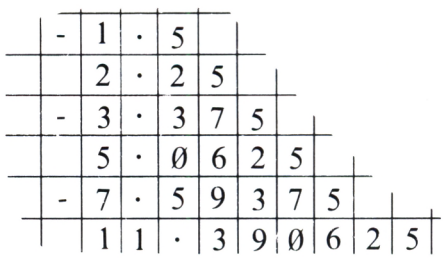
The + signs can't actually be seen (like I said, they're invisible) but *they still occupy the spaces in those columns.* That shoves the actual numbers further right. (Of course, the machine also prints a SPACE *after* the number, – in columns 22, 25, and 28 – but this is not caused by **TAB**; numeric variables *always* have a SPACE printed after them.)

A negative number, like –7, will come out in the same way; but now the – sign in the correct column is *visible* and everything looks OK. The reasoning behind this bizarre idea is that columns of numbers may look tidier if a space is reserved for the plus or minus sign. But then, of course, nobody in his right mind would actually *print* the plus signs, so . . .

Try this demonstration program.

```
1Ø  LET X = 1
2Ø  FOR T = 1 TO 1Ø
3Ø  LET X = –1.5 * X
4Ø  PRINT TAB(5);X
5Ø  NEXT T
```

You should get a print-out that has alternately positive and negative numbers in it, something like this:



and so on. Notice that despite Microsoft's heroic efforts, the sixth line has the decimal point out of alignment. You can't win 'em all.

> If **TAB** causes numbers to appear one column too far right, there is an 'invisible plus sign'. Change the **TAB** number to 1 less.

MARMADUKE'S POWER-DRILL

Marmaduke Mousebender, the Multidimensional Mathematician, wishes to print out on a TV screen the:

Squares	N*N
Cubes	N*N*N
Fourth powers	N*N*N*N

of a number N, ranging from 1 to 2Ø. He also wants them arranged in columns 6, 12, 18, 24 like this:

Column	6	12	18	24
	↓	↓	↓	↓
	N	N*N	N*N*N	N*N*N*N
	1	1	1	1
	2	4	8	16
	3	9	27	81

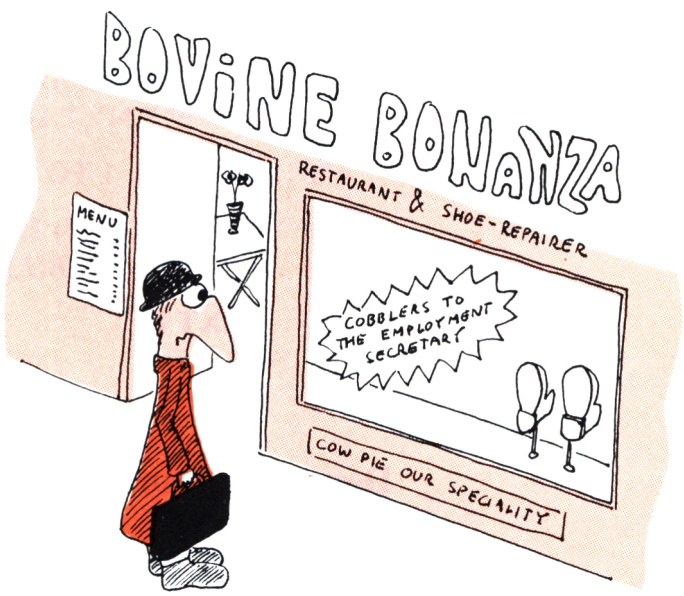
and so on. How can he do this, using **TAB** to set the column positions?

DESPAIRING DAN, THE DIAGONAL MAN

Despairing Daniel, you may recall, *loves* Cow Pie. He is designing an advertising display for a nationwide chain of restaurants, patronized by high government officials, to be known as the *Bovine Bonanza Restaurant and Drive-in Cobbler's*. The display calls for the words COW PIE to appear diagonally like this:

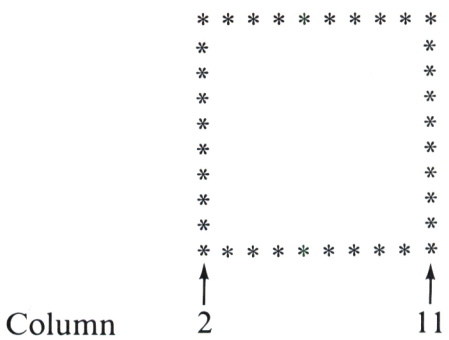
```
C
  O
    W
      P
        I
          E
```

How can this be done?



FRED'S FRAME

Fred Fenderbender, the Futile Freelancer, is experimenting with computer graphics. He has had a delightfully original idea: to print out a frame made up of asterisks. It must be a 1Ø × 1Ø square, like this:



Can you think of a way for him to do it?

(*Hint* Use **PRINT** "**********" for top and bottom, and some **TAB**s to put the sides in.)

ANSWERS

Marmaduke's Power Drill

```
10  PRINT TAB(6); "N";TAB(12);"N * N";TAB(18);
    "N * N * N";TAB(24);"N * N * N * N"
20  FOR N = 1 TO 20
30  PRINT TAB(5);N;TAB(11);N * N;TAB(17);
    N * N * N;TAB(23);N * N * N* N
40  NEXT N
```

Microsoft number trouble: change columns to 5, 11, 17, 23

Despairing Dan, the Diagonal Man

This one calls for brute force and ignorance. (More sophisticated approaches are possible, but not using what we know so far.)

```
10  PRINT "C"
20  PRINT TAB(2);"O"
30  PRINT TAB(3);"W"
40  PRINT TAB(4);"P"
```

```
50 PRINT TAB(5);"I"
60 PRINT TAB(6);"E"
```

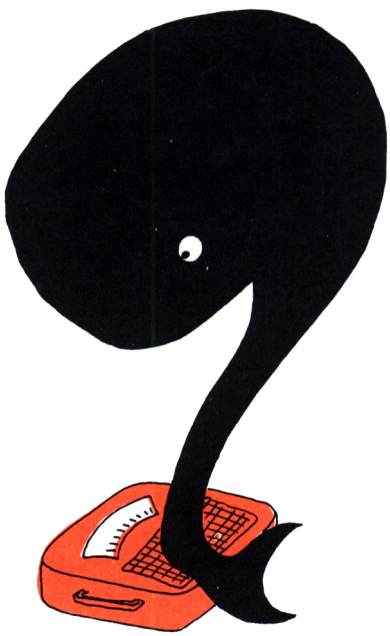
You might also think about using a loop, and a **DATA** list.

Fred's Frame

```
10 PRINT TAB(2);"**********"
20 FOR K = 1 TO 8
30 PRINT TAB(2);"*";TAB(11);"*"
40 NEXT K
50 PRINT TAB(2);"**********"
```

Logic Chopping

1. Only an elephant or a whale gives birth to a creature that weighs over 1ØØ kilograms.
2. The Prime Minister's son weighs 11Ø kilograms.
3. Therefore . . .



Logic is the art of making true deductions from true assumptions. The above deduction, when completed, amuses us because we know it must be wrong; yet on the face of it, the logical steps appear correct. The fallacy, of course, is that it is only the weight *at birth* that counts, and the Prime Minister's son was a normal, bouncing baby boy of some 5kg. (Well, a bouncing baby boy of some 5kg, certainly.)

Logic is about *statements*. A statement is something that is definitely either *true* or *false* (though it may be very hard to find out which!). Examples of statements are:

- $2 + 2 = 5$ (False)
- This sentence is on page 97. (False)
- All cows are mammals. (True)
- All mammals are cows. (False)
- Elvis Presley's Army serial number was 53310761. (True)
- Alexander the Great had a horse called Bucephalus. (True)
- Alexander the Great had a goat called William. (Almost certainly false, and definitely a statement; but lacking the proper historical documents it's not easy to decide whether or not it's true.)

You can't think of any phrases that are *not* statements? How about these?

- Ugly green furry things.
- Why is a mouse when it spins?
- What is the difference between a duck?
- Good Morning!
- And so on.

It's either

TRUE

or it's

FALSE

or it's not a statement.

DRILL PROBLEM

Which of the following are statements? Of those that are, which are true and which are false?

(a) How now, brown cow!
(b) Giraffes have long necks.
(c) What goes up and down the washing-line at 60 miles per hour?
(d) $14 + 22 > 5$.
(e) July 4th, 1776.

(f) If pigs had wings, we'd all carry umbrellas.
(g) Either it's snowing, or it's not.
(h) Today is Tuesday, or 2 + 2 = 4.
(i) You are the Bisto Man, and I claim my five pounds.
(j) If today is Tuesday, then tomorrow must be Wednesday.
(k) Cod and chips twice.
(l) This sentence is false.

COMPOUND STATEMENTS

Sometimes a statement is built up from other statements by combining them using the words:

AND
OR

For example:

(m) It's Monday, **AND** I'm bored, bored, bored, bored, bored.
(n) Roses are red **AND** violets are blue.
(o) I'll come by car **OR** I'll take the train.
(p) These shoes are too small **OR** my feet are too big.

See also (g, h, i) above. These are called *compound* statements. They're used in computing with **IF** . . . **THEN** branch commands, to take action when a combination of things happens. But the computing must wait for a few pages. First we must decide when such compound statements are, or are not, true.

For example, consider (m) above. When is that true?

Suppose today is Thursday. Then (m) is clearly *false*. Even if I really *am* bored, bored, bored, bored, bored.

Suppose it's Monday all right, but instead of being bored, bored, bored, bored, bored, I'm full of the joys of spring, bright-eyed and bushy-tailed, and raring to go. Then (m) is *still* false.

So, for a statement using **AND**, *both* parts have to be true in order for the compound statement to be true. We can write out a little table to show all the possibilities:

It's Monday	I'm bored, bored, bored, bored, bored	It's Monday **AND** I'm bored, bored, bored, bored, bored
True	True	True
True	False	False
False	True	False
False	False	False

What about **OR**? Let's take example (p). Suppose these shoes are *not* too small, but nonetheless my feet are too large. Then it's only fair to assume (p) is true. (It doesn't claim both things hold, just one **OR** the other.) The same goes if my feet are perfectly sweet and dainty, and nothing like too large, but the shoes are cramped and uncomfy because some twerp in the factory made them five sizes smaller than it says on the box. So now the compound statement is true provided at least one (possibly both) parts are true. To be false, both parts have to be false. As a table:

These shoes are too small	My feet are too large	These shoes are too small **OR** my feet are too large
True	True	True
True	False	True
False	True	True
False	False	False

A logician would summarize all this very briefly, by calling the two parts of the statements P and Q, and using T and F for True and False:

P	Q	P **AND** Q	P **OR** Q
T	T	T	T
T	F	F	T
F	T	F	T
F	F	F	F

This is called a *truth table.*

That sizes it up in a nutshell, so to speak; but it's awfully *dull.* But then, logicians are awfully dull people!

You can use

AND

and/or

OR

to combine several statements into one.

THE DUCHESS'S POTATOES

The door-bell rang, and I went to see who it was. It was a Post-Office messenger on a bicycle. I took the slim brown envelope from his hands, tipping him a shilling for his pains.

"Holmes! Holmes! A telegram!"

The great man was engaged in a delicate operation with a specimen of blood and some chemicals. He shook a test-tube containing a vile blue liquid. "Watson, if this tube turns yellow, it means a man's life! A neat example of the forensic arts, if I say so myself. See!"

"But Holmes, that's green, not yellow!"

"Close enough, Watson. I'm colour-blind anyway. Have you opened the telegram?"

"Yes, Holmes. But – I cannot understand a single word of it!"

He snatched it from my hand.

Post Office Telegram

The Royal Snail
Safe, Sure, *Slow*

TO MR. SHERLOCK HOLMES, 221B BAKER ST., LONDON,
FROM LADY INDORA BADEN-POOLE, DUCHESS OF WESTHAMPTONSHIRE
PIGS HAVE WINGS AND QUEEN ANNE IS DEAD; OR FOZZIE BEAR IS
PRESIDENT OF THE USA; OR THE DUKE'S PRIZE POTATO CROP HAS BEEN
HIJACKED; OR SPIDERS HAVE NINE LEGS; OR 'CHACUN A SON GOUT'
IS FRENCH FOR 'EVERYBODY HAS THE GOUT'. COME AT ONCE,
USUAL FEE, INDORA.

"Surely, Watson," said Holmes testily, "you can understand the word PIGS?"

"Well, yes, Holmes –"

"And yet you say you cannot understand a single word."

"Confound it, Holmes, what I meant was, it makes no sense to me at all! Why do you have to be so cold-blooded and logical –"

"But, my dear Watson, it makes *perfect* sense; and a logical analysis is the key to its comprehension. The only question we need to answer is: *is it true*?"

I frowned. Intelligent thought has never come naturally to me. "Well ... supposing it *were* true, Holmes – what would it *mean*?"

"That," said Sherlock Holmes, "is for you to decide."

Can you help Watson out? Assuming that the *whole statement* is *true*, what is the message contained in the telegram?

THE SAD SAGA OF SHIFTY SYD

Shifty Syd, the Scurrilous Software Salesman, is trying to sell a stockmarket program (called PECULATOR-SPECULATOR, developed by Apfelsoft Inc. for the Stork-37 micro) to J. Paul Grotty, the Morecambe Bay gas magnate. It comes in two parts. One displays on the screen the latest prices of stocks and shares. The second allows the user to choose which items he wants to buy and sell, and then automatically contacts his stockbroker's computer via the Micronit Network, to make the transaction.

For copyright reasons, we are unable to reproduce most of this program, but the relevant portions of it are given below. Acknowledgements are due to Apfelsoft Inc. for their generous permission to include these program lines.

```
10  PRINT "PART ONE"
20  PRINT "STOCKMARKET LISTING"
    (the next 1970 lines have been deleted)
2000  PRINT "DO YOU WISH TO CONTINUE?"
2010  PRINT "INPUT YOUR ANSWER: Y/N"
2020  INPUT A$
```

```
2030 IF A$ = "NO" THEN GOTO 7000
2040 PRINT "PART TWO"
2050 PRINT "AUTOMATED STOCK
     TRANSACTION"
     (the next 4940 lines have been deleted)
7000 PRINT "THANK YOU FOR USING
     APFELSOFT."
7010 PRINT "HAVE A NICE DAY."
7020 STOP
```

J. Paul Grotty was extremely interested in this program. In fact, he was thinking of buying six hundred copies for his international consortium of companies. So this was potentially a very big order for Shifty Syd.

Everything went perfectly, until Grotty reached line 2000 of the program – an input prompt that would allow him to exit the system if he did not wish to continue with the second part of the program. Now, you don't get to be a gas magnate by saying "Yes" to every dumb question, and Grotty's natural inclination was always to say "No" until he had time to think things over. He noticed that line 2010 was asking for Y/N as an input, by which he intelligently assumed that "Y" would mean 'Yes' and "N" would mean 'No'. (He wasn't born yesterday, and it never even crossed his mind to input "Y/N".)

So J. Paul Grotty typed "N", and pressed ENTER

Instead of exiting the system, however, he was astonished to see:

```
PART TWO
AUTOMATED STOCK TRANSACTION
```

appear on the screen. It took a further twenty minutes to reach the end of the program (which Grotty conservatively estimated as costing him some $42,074 in wasted time).

In vain did Shifty Syd point out that the correct input was "NO", not "N". The program had looked for "NO" in line 2030, not found it, and hence not jumped to line 7000, the exit routine.

J. Paul Grotty was not impressed.

Shifty Syd went back to his office, telephoned the Software Department, and let them know exactly what he thought of them.

Protocol Pete, the Prosaic Programmer, who had just taken a job working for Apfelsoft Inc., rewrote the program. He changed just one line, to:

```
2030  IF A$ = "N" THEN GOTO 7000
```

He also made several sarcastic comments about being given trivial jobs to do that wasted his remarkable talents.

Shifty Syd spent two hours on the telephone persuading J. Paul Grotty's secretary to give him a second chance. A week later, he visited Grotty in his office, and tried again.

All went well, until Grotty came to line 2010. Recalling that programmers often used "Y/N" as an abbreviation for "YES or NO", Grotty typed:

```
"NO"
```

Well . . . of course, this time the computer was looking for the input "N", and didn't find it; so it didn't jump to the exit routine either. That was another $42,074 in wasted time.

Syd lost the order, and is now working as a yak-herder in Mongolia. But how different it could have been, if only he'd known about . . .

THE USER-FRIENDLY INPUT

Yes indeed. And now we come to the part you've all been wondering about, namely: *what on earth does this have to do with logic commands*?

It's a common problem. Computer users are often given multiple choices, where they input their selection and the

program proceeds accordingly. The confusion between "N" and "NO", or "Y" and "YES", is notorious.

Of course, you could just make it absolutely clear to the user which inputs he is expected to use. But people, unlike computers, make mistakes. So a better idea would be to arrange things so that it didn't *matter* whether "N" or "NO" was used.

In other words, we want to jump to line 7000 if either of the two conditions:

1. A$ = "N"
2. A$ = "NO"

holds.

And the way to achieve this is to use:

```
2030 IF A$ = "N" OR A$ = "NO" THEN GOTO 7000
     |_condition 1_|  |_condition 2_|
```

Think how much happier J. Paul Grotty would have been! (And Shifty Syd, possibly even more so . . .)

Programs that are written so that an untrained person can use them easily, without having to worry about all sorts of fiddly details, are called *user-friendly*. Writing a user-friendly program isn't just a matter of programming *technique*: it's a question of *style*.

A program may *work* fine. But if it keeps going wrong for silly technical reasons, it's not user-friendly at all. It's user-frustrating.

> An **INPUT** command that checks for errors or alternatives is said to be *mugtrapped*. A mugtrapped program is more user-friendly.

INTERNATIONAL INPUT INQUIRY

J. Paul Grotty decided that if the existing software houses couldn't write the kind of programs he wanted, he'd have

to set up his own. A month later, *Grotty Programs* was in operation.

Its first product was an **INPUT** routine that would request a Y/N response from the user. But, to cater for the international market, it would accept all of the following as being the same as "NO".

NO
NON
NEIN
NYET
NOT ON YER NELLY

Can you work out how this could be done? You are allowed to string a whole series of conditions together using **OR**, like this:

IF A$ = "NO" **OR** A$ = "NON" **OR** A$ = "NEIN" **OR** ...

MELANIE MONSOON'S MELON MARKET

Minnie Monsoon's mother Melanie grows melons. Her melons are among the best in Lummoxshire, being a new and tasty variety, *Ponsonby's Delight*, imported as seedlings from the Minnesota Melon Belt. (The previous variety, *Yellow Revolting*, developed in Nice, wasn't.) (Nice, that is.) The only problem with Ponsonby's Delight is that it can't abide temperatures lower than 13°C, or higher than 21°C. So Melanie gets weather information every day from her daughter Minnie, and runs it through a small computer program that tells her whether or not to open the windows in her greenhouse. The windows should only be open when the temperature is between 13°C and 21°C (inclusive).

Now there is no **BETWEEN** command in BASIC. But Melanie had a quick word with Prudent Pete the Pineapple Programmer, and he pointed out that a temperature TEMP lies between 13 and 21, provided *both* the conditions:

1. TEMP > = 13
2. TEMP < = 21

see the box on page 56

hold. And the way to see if that is true is to use **AND**, like this:

IF TEMP > = 13 **AND** TEMP < = 21 **THEN** (whatever)

condition 1 — condition 2

So now Melanie's machine-minded melons ripen in the autumnal blizzards, thanks to a program that goes like this:

```
10 PRINT "INPUT TEMPERATURE"
20 INPUT TEMP
30 IF TEMP > = 13 AND TEMP < = 21
   THEN PRINT "OPEN WINDOWS"
```

PROCNATAKADSKI'S CODE-CHECKER

Comrade Sergei Procnatakadski, the Russian Spy, has been instructed by the Kremlin to change to a new code system. The new code uses a *key* word (not to be confused with a keyword) such as, for instance,

RUMBLING

This will be used to change the letters in a message, by altering the alphabet like this:

```
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
R U M B L I N G A C D E F H J K O P Q S T V W X Y Z
```

key word — other letters in order

The key word must have between 6 and 11 letters (because Moscow has determined that this is the optimum range for indecipherability of the code). Now poor old Procnatakadski never learned to count beyond 5, so he's in trouble.

Can you write a computer program that will accept a word as input, and print 'USE THIS ONE' if its length is between 6 and 11 (inclusive)? Use **LEN** to find the length, and make the program try again if the word that is input has the wrong length.

Ugly green furry things

There is no answer to ugly green furry things: you just have to learn to live with them.

Why is a mouse when it spins?

The higher, the fewer.

What is the difference between a duck?

One of its legs is both the same.

Drill Problem

(a) Remark, not statement.
(b) True statement.
(c) Honda pants; but it's not a statement, it's a question.
(d) True statement.
(e) Date, not statement.
(f) True statement. (Well, *I*'d carry an umbrella, I can assure you.)
(g) True statement (whatever the wevver).
(h) True statement (whether or not today is Tuesday).
(i) Statement. Truth depends: only true if you *are* the Bisto Man, **AND** I do claim my five pounds.
(j) True statement.
(k) Lunch, not a statement.
(l) *Paradox.* If it's true it must be false; if it's false it must be true. Best considered not a statement (or your computer will have a nervous breakdown).

The Duchess's Potatoes

"It's impossible, Holmes!"

"Nonsense, Watson. In a few minutes you will be telling me how utterly obvious it all is."

"I would never say such a thing about so baffling a problem, Holmes."

The great man sighed. "Observe, Watson, that the telegram is a compound statement, involving five components:

1. Pigs have wings **AND** Queen Anne is dead.
2. Fozzie Bear is President of the USA.
3. The Duke's prize potato crop has been hijacked.
4. Spiders have nine legs.
5. 'Chacun à son gout' is French for 'everybody has the gout'.

"Of these, number (1) is itself compound.

"Next, observe that statements (1), (2), (4) and (5) are all clearly *false*. Although Queen Anne *is* dead, pigs are not airborne. Fozzie Bear is not President of the USA –"

"I had my doubts about that one, Holmes."

"Me also, but we digress. Spiders have eight legs, not nine. And 'chacun à son gout' means 'each to his own taste'."

"Brilliant, Holmes. But . . ."

"Indeed. But: we have no idea as to the truth of (3). However, I asked you to suppose the whole statement were true. Now a compound statement:

(1) **OR** (2) **OR** (3) **OR** (4) **OR** (5)

can be true only when *at least one component* is true. And we know already that statements (1), (2), (4), (5) are false. Therefore . . ."

"Statement (3) must be true!" I cried. "The Duke's potato crop *has* been hijacked! Don't you see, Holmes? You really are slow today." Holmes had a peculiar look on his face. Blank incomprehension. I tried to explain it in words of one syllable. "Holmes, it's like you have said yourself: "Once you have eliminated the impossible, then whatever remains, however improbable, must be the truth!"

Holmes was turning bright red. Curious. A sudden heat-stroke? Astonishment at my brilliant powers of deduction? I felt the need to reassure him.

"Why, Holmes," I said solicitously. "Do you not see how utterly obvious it all is?"

International Input Inquiry

```
10 PRINT "INPUT YOUR DECISION: YES OR
   NO"
20 INPUT D$
30 IF D$ = "NO" OR D$ ="NON" OR D$ =
   "NEIN" OR D$ = "NYET" OR D$ = "NOT ON
   YER NELLY" THEN GOTO 500
   . . .
500 (take relevant action)
```

Procnatakadski's Code Checker

```
10 PRINT "INPUT WORD"
20 INPUT W$
30 LET L = LEN(W$)
40 IF L > = 6 AND L < = 11 THEN PRINT
   "USE THIS ONE"
50 IF L < 6 OR L > 11 THEN PRINT "TRY
   AGAIN"
60 IF L < 6 OR L > 11 THEN GOTO 10
```

INTs and INTeaters

Bernard Gasquet, who had been snoozing in an armchair, awoke with a start. His wife Ermintrude has just thrown her notebook on to the floor with a tremendous THUMP! She looked unhappy: he could tell by the way she was chewing the edge of the carpet.

"What's the matter, 'Trude?"

"It's this *awful* correspondence course that I'm doing for the Opeless University, Bernard!" (She pronounced his name 'Ber-naaahd'.) "Just *look* at these *questions*!"

Bernard picked up the notebook with its garish yellow cover and the title:

AAAA 001: *ARITHMETIC FOR ANTEATERS, AARDVARKS AND ARMADILLOS.*

A trifle bizarre, thought Bernard; but then, 'Trude *was* a trifle bizarre. Perhaps it was inherited from her mother, Charlotte Russe. The trifle part, that is. He opened the book to a page that 'Trude had marked.

– 474 – AAAA 001

CMA 63: If 29,453 anteaters have 6,396,375 ants to share equally between them, how many will each get, and how many will be left over?

CMA 64: An aardvark goes into a shop and buys 4,992,641 quadruples of socks. (Note: aardvarks have four feet, so *pairs* of socks will not suffice.) If he uses one quadruple every day, and then throws it away (which is wise – have you ever smelt an aardvark's socks? Come to that, have you ever smelt an aardvark?) how many years will they last? You may ignore extra days in leap years, since aardvarks always sleep through February 29ths.

CMA 65: An armadillo inside a space-probe encircles the planet Saturn once every 79 days. Assuming Saturn goes once round the Sun in 10,757 days, how many times will the armadillo encircle Saturn during one such rotation round the Sun?

Bernard thought for a moment. "Well, 'Trude, I'd guess they were some kind of oddball division sums."

"I *know*, Bernaaahd dahling; but you *know* how *terrible* I am at oddball division! Why, I have trouble adding up my charge account at Portnoy and Maidstone's!"

"I'm well aware of that, 'Trude," sighed Bernard. "Um. What does CMA mean? Careful of My Aardvark?"

"Computer-Marked Assignment, Bernaaahd."

"Oh. Compu – Hey! That's it, 'Trude! Why not use the computer? The Stork-37 can do division faster than you can say 'Megaflop'!"

And so the program was born:

```
10 PRINT "CMA 63:"; 6396375 / 29453
20 PRINT "CMA 64:"; 4992641 / 365
30 PRINT "CMA 65:"; 10757 / 79
```

number of days in year

And they printed it out and sent if off to be marked.

BUT IT WASN'T THAT EASY...

Two weeks later the assignment came back, looking like this:

	RIGHT	WRONG
CMA 63: 217.172275 CMA 64: 13678.4685 CMA 65: 136.164557		X X X
MARKER'S COMMENT: YOU IGNORED THE INSTRUCTION ON PAGE 473, DUMMY.		

After a certain amount of grubbing around they found the instruction referred to:

> *When answering CMAs 60–80 ignore any fractional part. Give answers in whole numbers.*

Which was a problem, because the Stork-37 micro always seemed to give the answer in decimals.

What were the Gasquets to do?

INTEGER PARTS

Eventually Ermintrude Gasquet found the answer in the *Stork-37 Manual*. The BASIC keyword:

INT

may be used to convert a decimal number to the largest whole number that is less or equal to it. That is, to throw away anything after the decimal point (at least, for positive numbers – negative ones work a little differently). For instance,

INT(3.14159) is 3

INT(22.222222) is 22

INT(217.172275) is 217

In general,

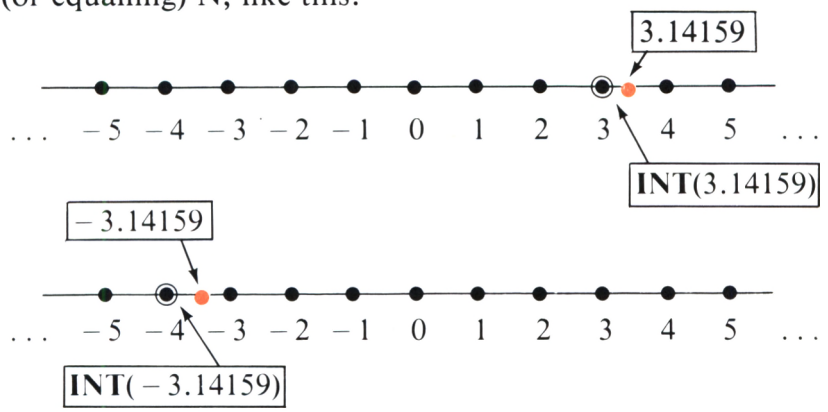
INT(number)

gives the *integer part* of the number. For a positive number this is everything that occurs before the decimal point. For a negative number it is one less (except when the number is already an exact whole number). Try this program:

```
10 PRINT INT(3.14159)
20 PRINT INT(-3.14159)
30 PRINT INT(3)
40 PRINT INT(-3)
```

You should get the answers 3, −4, 3, −3. Note how **INT**(−3.14159) is *not* −3, but −4, because the number is negative and *not* a whole number; but **INT**(−3) *is* −3 because, although negative, it *is* a whole number.

All this makes good sense on a number-line, because then **INT**(N) is the largest whole number less than or equal to N, which is the largest whole number lying to the left of (or equalling) N, like this.



Technically, **INT** is not a command, but a *function*: if you give it a number, it gives you one back. So you can say things like:

```
10 PRINT INT(10757/79)
20 LET Y = INT(12.35)
30 LET M(4) = 3 * INT(Z/7)
```

but *not*:

```
40 INT(10757/79)
```

because the computer doesn't know what to do with the **INT** when it's found it.

INT rounds a number *down* to the previous whole number – unless it's a whole number already, in which case it leaves it alone.

THE GASQUETS TRY AGAIN

Ermintrude and Bernard rewrote their program to read:

```
10 PRINT "CMA 63:"; INT(6396375/29453)
20 PRINT "CMA 64:"; INT(4992641/365)
30 PRINT "CMA 65:"; INT(10757/79)
```

What answers did they get?

...AND NEARLY SUCCEED

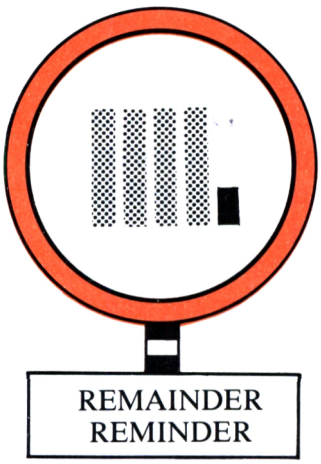
What they got was right; but they'd forgotten part of CMA 63, which asked not just for the number of ants per anteater, but also *how many were left over*. In other words, what is the *remainder* you get when you divide 6,396,375 by 29,453?

Let's take a look at an easier problem. If you divide 24 by 5, what remainder do you get?

Well, obviously 4. But how do you persuade a computer to work that out for you?

You can do it this way.

Step 1	Divide 24 by 5 (to get 4.8)	10 **LET** D = 24 / 5
Step 2	Take the integer part (to get 4)	20 **LET** I = **INT**(D)
Step 3	Multiply this by 5 (to get 20)	30 **LET** M = 5 * I
Step 4	Subtract the result from 24 (to get 4, the remainder required)	40 **LET** R = 24 – M



Get it? We know that 5 goes into 24 four times. That uses up 4 * 5 = 2Ø, so 24 – 2Ø = 4 are left.

Looking at the computer program, it's clear we can shorten it by combining the calculations like this:

```
1Ø  LET R = 24 – 5 * INT(24 / 5)
```

And in general, to get the *remainder* on dividing a number N by a number K we replace 24 by N, 5 by K, to get:

```
1Ø  LET R = N – K * INT(N / K)
```

how to find a remainder

Here's a simple test program that lets you **INPUT** two numbers N and K, and tells you how many times K goes into N and what the remainder is.

```
1Ø  PRINT "INPUT NUMBER"
2Ø  INPUT N
3Ø  PRINT "INPUT DIVISOR"
4Ø  INPUT K
5Ø  PRINT
6Ø  PRINT K;" INTO";N; "GOES";
7Ø  PRINT INT(N / K);"TIMES"
8Ø  PRINT "WITH REMAINDER";
9Ø  PRINT N – K*INT(N / K)
```

You should now be able to answer the rest of CMA 63.

THE INT-EATER PROBLEM

How many **INT**s *do* you get left over if you share 6,396,375 of them between 29,453 **INT**eaters? Or have I got that muddled somehow?

> If you divide N by K the result is:
>
> **INT**(N / K)
>
> and the remainder is:
>
> N – K * **INT**(N / K)

NINETEEN EIGHTY-FOUR

. . . and Big Brother will be watching *you*! Using sophisticated computerized surveillance techniques, of course. But Big Brother has a problem. He knows that 1984 starts on a Sunday; and his computer will tell him such juicy items as 'On the 85th day of 1984 Fred Nagswindler was seen reading a copy of a subversive publication, namely *The Beano*.' Unfortunately, Big Brother's computer won't tell him what day of the week the 85th day in 1984 is, and that's important, because it's not only legal, but *compulsory* to read *The Beano* on a Tuesday, and not even Big Brother wants to put someone in the pokey for obeying the law.

Can you write him a program that accepts as input the day number, between 1 and 366 (1984 was a leap year), and prints out which day it is?

Hint Think about remainders after dividing by 7.

PRIME TIME

And now, to finish this chapter with a bang, I'll –

Hang on, someone's knocking at the door. I'll open it. There. Oh dear, I think I boobed. There are five of them, wearing jackboots and trenchcoats. That's what you get for making jokes about Big Brother . . .

"Is your name Stewart?"

"Yes, . . . Sir."

"We represent COPSAC."

"Unh?"

"The Citizens' Organization for the Promotion of Sensible Applications of Computing."

"Oh."

"It has been drawn to our attention that this book contains an unusually high proportion of frivolous examples."

"Yes, well, of course, I do try to present them in an entertaining way, Sir, but you'll find that underneath the frivolity there is a serious purpose, and anyone who reads them will get a very basic – "

"COPSAC is not concerned with what lies beneath the surface, Stewart! Like all pressure-groups, we have enough trouble handling superficialities, without worrying about what people are *really* trying to do!"

"I apologize. I will try to be more obvious in – "

"You will do more than apologize! *You will include a really serious application of computing*! *Now*! *This very instant*!"

"I refuse! This is *my* book, and I'll write what I want! You're just characters, I can make you go away whenever I feel like it! I can – "

They elbow their way into my house. This is a nightmare. I don't seem to have control of my own book any more. I guess there's only one thing for it . . .

* * * * * * *

You can often build up big numbers by multiplying smaller ones together. For instance, 72 = 3 * 24. A number that can be formed in this way is called *composite*. If a number cannot be written as a product of smaller ones, it is said to be *prime*. For example, the numbers:

2 3 5 7 11 13 17 19 23 29

are the first ten primes.

You can't get a *prime* by multiplying two smaller numbers together.

The following program will let you **INPUT** any number with up to eight digits, and it will factorize it completely into primes. For example, if you **INPUT** 60 it will print:

60 = 2 * 2 * 3 * 5

or if you **INPUT** 232841 it will print:

232841 = 7 * 29 * 31 * 37

You'll be amazed by the remarkable arithmetical capabilities of your computer. (Hey, this is going easily. I'll just sneak a joke in while they're not watching. Did you hear the one about the Archbishop and the Belly-dancer? Well, it seems there was this – Aaaaaaaaaaaaagghhh!
Sorry. I tried.)
Here's the program.

```
10  PRINT "PRIME FACTORIZATION"
20  PRINT
30  PRINT "NUMBER TO BE FACTORIZED?"
40  INPUT N
50  PRINT
60  PRINT N;" =";
70  LET N0 = N
80  IF 2*INT(N/2) = N THEN PRINT
    " SPACE 2 SPACE *";
90  IF 2*INT(N/2) = N THEN LET N = N/2
100 IF 2*INT(N/2) = N THEN GOTO 80
110 LET K = 3
120 IF K*INT(N/K) = N THEN PRINT K; "*";
130 IF K*INT(N/K) = N THEN LET N = N/K
140 IF K*INT(N/K) = N THEN GOTO 120
150 IF K*K > N THEN GOTO 180
160 LET K = K + 2
170 GOTO 120
180 IF N = N0 AND N0 > 1 THEN PRINT
    " SPACE PRIME"
```

```
190  IF N < NØ AND N > 1 THEN PRINT N
200  PRINT
210  PRINT
220  PRINT "THIS PROGRAM WAS SPONSORED
     BY COPSAC'S"
230  PRINT "CLEAN UP COMPUTING
     CAMPAIGN"
```

What the program does is first test for divisibility by 2, and then, in turn, for divisibility by odd numbers 3, 5, 7, ... up to the square root of N, beyond which it is impossible to go.

The next bit is hard going, and you can skip to the next section (DRILL PROBLEMS) if you wish.

Let's look at the factorization program in more detail. To test whether a number K divides a number N exactly, you test whether the *remainder* is zero. Now we've seen that the remainder is given by:

N − K*INT(N/K)

and this is zero precisely when:

K*INT(N/K) = N

So lines 8Ø−1ØØ test for divisibility by 2; and 12Ø−14Ø test for divisibility by K. Note that K starts at 3 in line 11Ø, and increases by 2 in line 16Ø, to 5, 7, 9, ... etc. A certain amount of looping goes on, caused by the **GOTO** commands, to make sure all factors of 2 are used up before going on to 3, and so on.

The **IF** ... **THEN** parts of lines 8Ø−1ØØ or 12Ø−14Ø are the same: this is because we want *three* distinct actions to be taken if the condition holds. We can't write:

IF condition **THEN** action 1 **AND** action 2 **AND** action 3

because **AND** refers to statements, not actions! But we can split this up into three program lines:

IF condition **THEN** action 1

IF condition **THEN** action 2

IF condition **THEN** action 3

using the same condition each time.

(Most computers, including the CPC464, will also allow *multi-statement lines* of the form:

IF condition **THEN** action 1 : action 2 : action 3

But these are a minor side-issue which I'm not going to discuss here.)

To see what's going on, let's do a dry run with N = 21ØØ. It goes like this:

Line number	N	K	NØ	Comments
1Ø–3Ø	Ø	Ø	Ø	Instructions
4Ø	21ØØ			Input
5Ø–6Ø				Print formatting
7Ø			21ØØ	Remember value of N in a variable that won't be changed.
8Ø				Condition true: **PRINT** 2*
9Ø	1Ø5Ø			Condition true
1ØØ				Condition true: **GOTO** 8Ø
8Ø				Condition true: **PRINT** 2*
9Ø	525			Condition true
1ØØ				Condition true: **GOTO** 8Ø
8Ø				Condition false
9Ø				Condition false
1ØØ				Condition false
11Ø		3		
12Ø				Condition true: **PRINT** 3*
13Ø	175			Condition true
14Ø				Condition true: **GOTO** 12Ø
12Ø				Condition false
13Ø				Condition false
14Ø				Condition false
15Ø				Condition false
16Ø		5		
17Ø				**GOTO** 12Ø unconditionally
12Ø				Condition true: **PRINT** 5*
13Ø	35			Condition true
14Ø				Condition true: **GOTO** 12Ø
12Ø				Condition true: **PRINT** 5*
13Ø	7			Condition true
14Ø				Condition true: **GOTO** 12Ø
12Ø				Condition false
13Ø				Condition false
14Ø				Condition false
15Ø				Condition true: **GOTO** 18Ø
18Ø				Condition false
19Ø				Condition true: **PRINT** 7
2ØØ–23Ø				Sign off

Looking down the final column, you can see that the print-out will be:

$$2100 = 2 * 2 * 3 * 5 * 5 * 7$$

which is correct. Watch how the value in column N steadily decreases as each factor is found and divided out.

DRILL PROBLEMS

Factorize the following into primes, by running the program.

(a) 60
(b) 555555
(c) 121771
(d) 3778125
(e) 11111117 (The CPC464 takes about a minute over this!)
(f) 1024

There's a very tiny bug that occurs in (f). Can you work out why?

ANOTHER SERIOUS PROGRAM FROM COPSAC

(a) Modify the program above so that it loops, listing the factorizations of all numbers N from 2 to 10,000 in turn. (This will take some time to run, so you may prefer to let N go from 5000 to 5500, say.)
(b) Write a program that lists all prime numbers in turn, starting from a given **INPUT** value, and continuing indefinitely.

ANSWERS

The Gasquets Try Again . . .

CMA 63: 217
CMA 64: 13678
CMA 65: 136

The INTeater Problem

5Ø74

Nineteen Eighty-Four

```
1Ø  PRINT "BIG BROTHER IS WATCHING YOU"
2Ø  PRINT "INPUT DAY NUMBER"
3Ø  INPUT D
4Ø  IF D < 1 OR D > 366 THEN GOTO 2Ø
5Ø  LET R = D-7*INT(D/7)
6Ø  IF R = 1 THEN PRINT "SUNDAY"
7Ø  IF R = 2 THEN PRINT "MONDAY"
8Ø  IF R = 3 THEN PRINT "TUESDAY: LEGAL
    BEANO DAY"
9Ø  IF R = 4 THEN PRINT "WEDNESDAY"
1ØØ IF R = 5 THEN PRINT "THURSDAY"
11Ø IF R = 6 THEN PRINT "FRIDAY"
12Ø IF R = Ø THEN PRINT "SATURDAY"
```

protect against nonsense inputs

Drill Problems

(a) 2 * 2 * 3 * 5

(b) 3 * 5 * 7 * 11 * 13 * 37

(c) 13 * 17 * 19 * 29

(d) 3 * 5 * 5 * 5 * 5 * 5 *13 * 31

(e) PRIME

(f) 2 * 2 * 2 * 2 * 2 * 2 * 2 * 2 * 2 * 2

There's a minor bug in the program as it stands – you sometimes get an extra '*' on the end. This is harmless – but you might like to try to eliminate it if it bothers you. (A bug that doesn't matter is called a *feature*.)

(a) Delete lines 30 and 40. Add:

```
30  FOR M = 2 TO 10000
```

or 5000 to 5500, say

```
40  LET N = M
195 NEXT M
```

(b) One solution is to modify the original primes program. Change line 30 to:

```
30  PRINT "START NUMBER FOR PRIME
    SEARCH?"
```

Then delete all **PRINT** statements that give factors; but leave the line where 'PRIME' occurs. So you delete lines:

```
60  80  120  190
```

Change line 180 to **PRINT** the prime:

```
180 IF N = N0 AND N0 > 1 THEN PRINT N0
```

Loop indefinitely:

```
185 LET N = N0+1
```

careful! Don't use N = N+1. N has changed

```
187 GOTO 70
```

You will also have to change the **GOTO**s at the end of lines 100, 140 and 170 because lines 80 and 120 have been deleted:

```
100 IF 2*INT(N/2) = N THEN GOTO 90
140 IF K*INT(N/K) = N THEN GOTO 130
170 GOTO 130
```

The trouble with this is it's very inefficient, because the computer finds the whole factorization: it just doesn't print it out. So laziness is not much of a virtue here. The simplest solution is to rewrite the program, looking for a factor 2 or an odd number, and stopping if one is found. Like this:

```
10  PRINT "START NUMBER FOR PRIME
    SEARCH?"
20  INPUT N
30  IF N = 1 THEN LET N = 2
40  IF N = 2 THEN PRINT N
50  IF N = 2*INT(N/2) THEN GOTO 120
60  LET K = 3
70  IF K*K > N THEN GOTO 110
80  IF N = K*INT(N/K) THEN GOTO 120
90  LET K = K+2
100 GOTO 70
110 PRINT N
120 LET N = N+1
130 GOTO 50
```

Glossary

Array A variable with numbered entries, like a list. The Nth entry in an array named ARRAY is referred to as ARRAY(N).

Back-up Spare copy of computer software, kept apart for safety.

BASIC Beginner's All-purpose Symbolic Instruction Code. Computer language widely used on home computers.

Bug A mistake in a program.

Cassette Tape used to store a program permanently. Unlike discs, cheap but slow.

Character A symbol that the computer can print on the TV screen, such as X, 7, %, and so on.

Column A line of characters vertically down the TV screen.

Command A single BASIC instruction, such as **PRINT** "FRED".

Composite A number that can be factorized into smaller ones, such as:

```
24 = 3 * 8
```

Concatenation Longwinded way of saying 'jamming together'. Otherwise known as string addition – for instance "HOT" + "DOG" = "HOTDOG".

Data list A series of **DATA** commands in a BASIC program, containing information to be used by the program.

Debugging Making a program work properly.

Delimiter Nonsensical input used to terminate what would otherwise be an endless input loop.

Disc drive A device that stores a lot of information on a magnetic disc.

Dry-running A method for debugging a program by working through parts of it by hand.

Empty string A sequence of characters that doesn't contain any!

Factorization Writing a number as a product of primes.

File A list of data stored externally to a computer, for instance on tape or disc, that can be read back into the machine for use within a program.

Floppy disc A circular disc of flexible plastic, covered in magnetic film, and encased in a paper cover, used on disc drives.

Flowchart A diagram that uses boxes linked by arrows to show what the program will do.

Function A rule that associates with each variable some particular value.

Initialize Set up values of variables at the start of a program or a part of a program.

Input loop A program loop containing an input command. Used to feed several items of data into the machine.

Integer part The largest whole number less than or equal to a given one. The symbol for this is **INT**. For instance **INT**(6.83) is 6.

Keyword Special BASIC word such as **PRINT**, **NEW**, **RUN**, **LIST**.

Logic The art of making valid deductions from valid assumptions. In a computer, how to decide what statements are *true* or *false*.

Loop Part of a program that works through the same sequence of commands several times, usually changing some of the variables as it does so.

Mnemonic A variable name that reminds you what it is, such as:

PRICE, HEIGHT, USERNAME$.

Multi-statement line A line of BASIC containing several commands separated by colons (:).

Numeric variable A variable whose values are numbers.

Output To get information out of the computer; or, information so obtained.

Pirate A person who steals programs by copying them.

Pointer A variable used to indicate a position in some array, whose contents are the main item of interest.

Prime A number not divisible by any smaller whole number, such as 17.

Program List of instructions for the computer to carry out.

Prompt Message accompanying an input command to remind the user what is required.

RAM Random Access Memory. The part of memory that can be changed by the programmer.

Remainder What's left when you divide one number by another one, not allowing fractions.

Row A line of characters horizontally across the TV screen.

Search Systematically run through a **DATA** list, looking for a particular item of information.

Software Programs stored in physical form as tapes, discs, or printed listings. Large parts of this book!

Statement Assertion that is either *true* or *false*.

Step size The value by which the loop counter changes in a **FOR...NEXT** loop.

String Any sequence of characters – including none at all!

Table look-up A method for changing information in a systematic way, by searching a list for an item and seeing what it must be changed to.

Tabulation Arranging things in neat columns.

Test line A line added to a program during debugging, to find out what is happening in a program.

Trace A command used to find out which lines of a program are being carried out during a program run.

Truth table Way of tabulating the possible combinations of truth and falsity in a compound logical statement.

User You.

User-friendly Requiring little experience to operate, giving helpful messages, accepting different versions of an input, and generally making life easy for *people*, instead of for the *computer*.

Zero On computers this is written Ø to distinguish from the letter 'Oh'.

Commands and Symbols Index

Keywords

Arithmetic and String-handling

Other titles of interest

Gateway to Computing Book 1: Amstrad CPC464 **£4.95**
Ian Stewart

Gateway to Computing Book 3: Amstrad CPC464 **£4.95**
Ian Stewart

(All the books in the Gateway Series are also available for the BBC Micro, ZX Spectrum, Dragon 32, Commodore 16 and Commodore 64)

The Complete Introduction to the Amstrad CPC464 **TBA**
Eric Deeson

On the Road to Artificial Intelligence: Amstrad CPC464 **£5.95**
Jeremy Vine

Bells and Whistles on the Amstrad CPC464 **£4.95**
Jeremy Vine

Computers in a Nutshell **£4.95**
Ian Stewart

Computing: A Bug's Eye View **£2.95**
Ian Stewart

Programming for REAL Beginners: Stage 1 **£3.95**
Philip Crookall

Programming for REAL Beginners: Stage 2 **£3.95**
Philip Crookall

Brainteasers for BASIC Computers **£4.95**
Gordon Lee

'Just the job for a wet afternoon with the computing class'—
Education Equipment

Shiva also publish a wide range of books for the BBC Micro, Electron, ZX Spectrum, Atari, VIC 20 Commodore 64, Commodore 16, Commodore Plus/4, Sinclair QL, Dragon, Oric and Atmos computers, plus educational games programs for the BBC Micro. Please complete the order form overpage to receive further details.

ORDER FORM

I should like to order the following Shiva titles:

Qty	Title	ISBN	Price
____	GATEWAY TO COMPUTING BOOK 1: AMSTRAD CPC464	1 85014 016 2	£4.95
____	GATEWAY TO COMPUTING BOOK 3: AMSTRAD CPC464	1 85014 078 2	£4.95
____	THE COMPLETE INTRODUCTION TO THE AMSTRAD CPC464	1 85014 002 2	TBA
____	ON THE ROAD TO ARTIFICIAL INTELLIGENCE: AMSTRAD CPC464	1 85014 064 2	£5.95
____	BELLS AND WHISTLES ON THE AMSTRAD CPC464	1 85014 063 4	£4.95
____	COMPUTERS IN A NUTSHELL	1 85014 018 9	£4.95
____	COMPUTING: A BUG'S EYE VIEW	0 906812 55 0	£2.95
____	PROGRAMMING FOR REAL BEGINNERS: STAGE 1	0 906812 37 2	£3.95
____	PROGRAMMING FOR REAL BEGINNERS: STAGE 2	0 906812 59 3	£3.95
____	BRAINTEASERS FOR BASIC COMPUTERS	0 906812 36 4	£4.95
____	.....	.....	.....
____	.....	.....	.....
____	.....	.....	.....

Please send me a full catalogue of computer books and software: ☐

Name .....

Address .....

.....

.....

This form should be taken to your local bookshop or computer store. In case of difficulty, write to Shiva Publishing Ltd, Freepost, 64 Welsh Row, Nantwich, Cheshire CW5 5BR, enclosing a cheque for £ .....

For payment by credit card: Access/Barclaycard/Visa/American Express

Card No ..... Signature .....

Notes

Notes

Notes

Gateway to Computing

with the Amstrad CPC464

You are going to meet some strange characters through the gateway. Be prepared to fight a few battles when you confront:

- Minnie and Max Monsoon, the Merciless Meteorologists
- J Paul Grotty, the Morecambe Bay Gas Millionaire
- Millicent MacHaddock, the Misguided Milkmaid
- The Citizens' Organization for the Promotion of Sensible Applications of Computing

They're tough but they're fun. What's more, they hold some important secrets of programming techniques and structures. Their alliance will be an invaluable guide to strategy and tactics in your quest to overcome the beasts of the chip: computer logic, bugs and data-handling.

In this, the second book of the *Gateway to Computing* series, you will pass beyond elementary BASIC. And once you have conquered the skills held within, you will be a true Knight of the Gateway and fly the Flag of Computing.

UK £ NET +004·95

ISBN 1-85014-023-5

00495

9 781850 140238

SHIVA

Shiva Publishing Limited

Gateway to Computing with the Amstrad CPC464 BOOK 2 Stewart SHIVA

AMSTRAD CPC

MÉMOIRE ÉCRITE
MEMORY ENGRAVED
MEMORIA ESCRITA

https://acpc.me/