USBORNE GUIDE TO

PROGRAMMING TRICKS & SKILLS

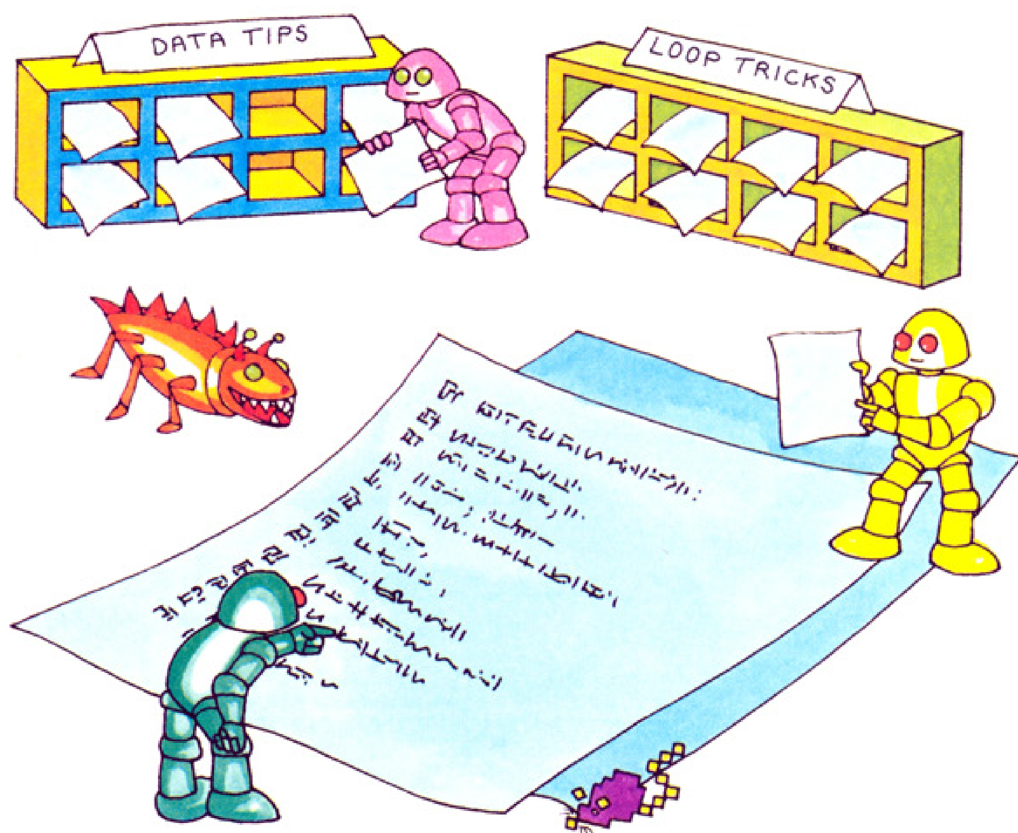
Professional tips and hints for better program writing

WITH LOTS OF PROGRAMS AND SUBROUTINES

Usborne Computer Books

PROGRAMMING TRICKS & SKILLS

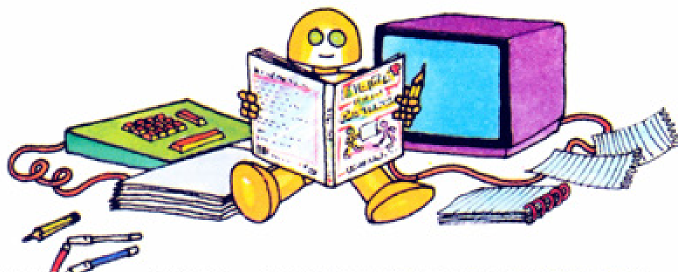
Lisa Watts and Les Howarth



Designed by Kim Blundell
Illustrated by Naomi Reed

Contents

4 What are programming tricks?
6 Program planning skills
8 What to do first
9 General purpose subroutines
11 Defining your own functions
12 Variables tricks
14 IF/THEN skills
16 Repeating things: loops
18 Storing data in arrays
20 Data tricks
27 Checking input
28 Screen display tips
30 Writing menus
32 Speeding up programs
33 Space-saving tricks
35 Flip-file program
43 Guide to BASIC
48 Index



First published in 1984 by Usborne Publishing Ltd, 20 Garrick Street, London WC2E 9BJ.

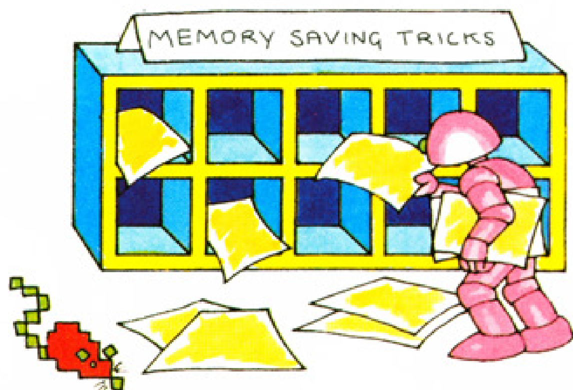
© 1984 Usborne Publishing
All rights reserved. No part of this publication may be reproduced, stored in a retrieval system or transmitted in any form or by any means, electronic, mechanical, photocopying, recording or otherwise, without the prior permission of the publisher.

The name Usborne and the device are Trade Marks of Usborne Publishing Ltd.

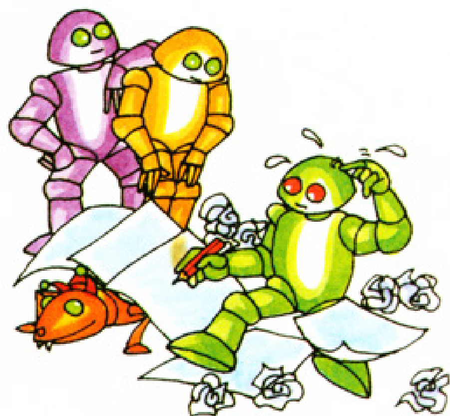
Printed in Spain. D.L.B. 28538-1984

About this book

This book is packed with tips, hints and tricks to help you write better programs in BASIC. There are tricks to make your programs run faster, ideas for ways to save memory space and ways to lock your programs so that other people cannot run them.

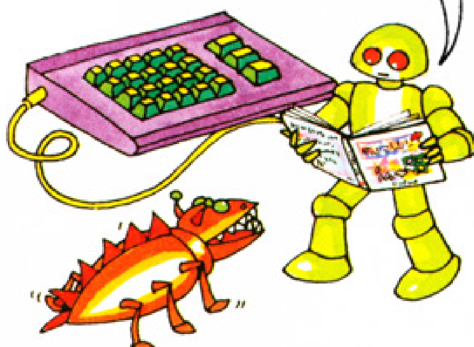


The first part of the book shows you how to plan and write clear, well-organized programs which are easy to debug. Then there are detailed instructions which show you how to use BASIC to create quite complicated programming techniques.

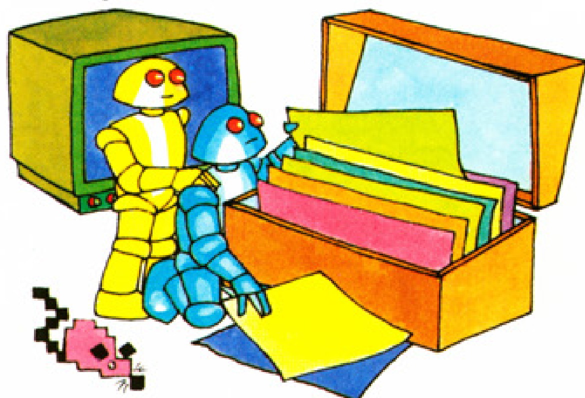


The book also contains lots of short routines which you can build into your own programs. There are routines to decorate the screen display and create professional-looking menus, for example, and lots of routines for packing data into the memory to save memory space.

If you are a beginner you will find it helpful to read a beginner's book, such as the *Usborne Introduction to Computer Programming*, before you use this book.



On pages 35-42 there is a long program called *Flip-file* which uses many of the tricks and routines described in this book. *Flip-file* is a simulation of a card index file in which you can store names and addresses or other information. Alongside the program there are detailed explanations of how it works, so you can study the programming techniques in action.



All the programs and routines in this book are written in a standard form of BASIC, which, with minor alterations, will run on most of the main makes of home computer. At the back of the book there is a Guide to BASIC with explanations of all the main BASIC words and, where necessary, conversions for each of the main makes of home computer.

What are programming tricks?

A programming trick is a short, neat way of carrying out a task in a program. Many of the tricks in this book are in the form of subroutines which can be used in lots of different programs. On the next few pages you can find out how to write a program as a series of subroutines, with each subroutine carrying out a separate task. This makes writing a long program more like writing lots of short programs. Frequently, too, you can use the same subroutine for carrying out a particular task in lots of different programs. Professional program writers hardly ever write a new program entirely from scratch.

What is a subroutine?

A subroutine, sometimes just called a routine, is a set of program lines for carrying out a particular task. The subroutine can be placed anywhere in the program and can be carried out as many times as you like. Each time you want the computer to perform that task you use the BASIC command GOSUB with the number of the first line of the subroutine.

```
200 PRINT "DO YOU WANT TO SAVE YOUR"
210 PRINT "DATA? PLEASE TYPE Y OR N."
220 INPUT A$
230 IF A$="Y" THEN GOSUB 2000
240 REM REST OF PROGRAM
500 END

1995 REM SUBROUTINE TO SAVE DATA
2000 PRINT "START CASSETTE"
2010 PRINT "THEN PRESS RETURN"
2020 INPUT R$
2030 SAVE ""
2040 RETURN
```

Subroutine (lines 2000–2040)

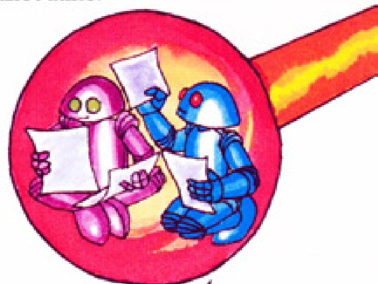
At the end of the subroutine you need the command RETURN. This tells the computer to go back to the instruction after GOSUB.

Parts of a program

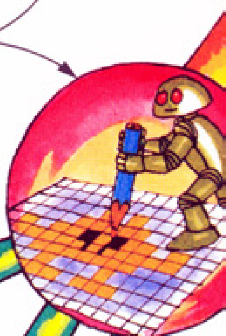
Most programs consist of three main parts: a program control centre with worker subroutines to carry out the main tasks and service routines to help them.

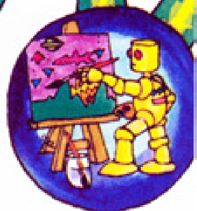
The control centre is the part which controls the sequence of events in the program, telling the computer when to carry out each subroutine.

Worker subroutines

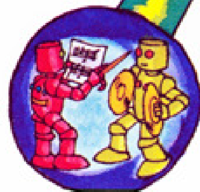


The worker subroutines carry out the tasks which are special to that program. For example, in a missile game the worker routines would calculate the current positions of the missiles and update the score.





Service routines



When you are planning and writing a program, look out for tasks which need to be carried out frequently at different stages during the program. For example, displaying a message on the screen or creating a delay. Putting these tasks in service subroutines will make your program shorter, faster and

Control centre

SCORE

The service subroutines help the worker routines by carrying out often needed tasks, such as plotting the screen graphics and displaying the score.

easier to understand. It does not matter how long or short the routines are – a subroutine to create a delay can be a single line containing a delay loop. For more examples of the sort of tasks that can be carried out by service routines, study the *Flip-file* program at the back of this book.

REM tricks

To help you remember what each subroutine is for, label it with a REM statement. The computer ignores all instructions starting with REM.

When you have finished the program and it is running smoothly, you may want to delete the REM statements, to save memory space.

```
200 GOSUB 500

495 REM DELAY SUBROUTINE
500 FOR J=1 TO 200:NEXT J
510 RETURN
```

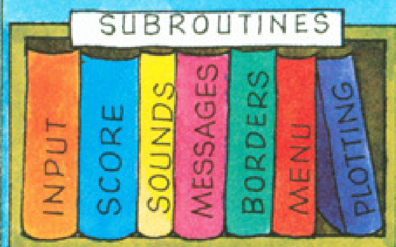
To make this task easier, put the REM statement on the line before the one listed in the GOSUB command, so you do not have to renumber the program.

```
200 GOSUB 500

500 FOR J=1 TO 200:REM DELAY
510 NEXT J
520 RETURN
```

Alternatively, you can put the REM as the last statement in a multistatement line, as shown above.

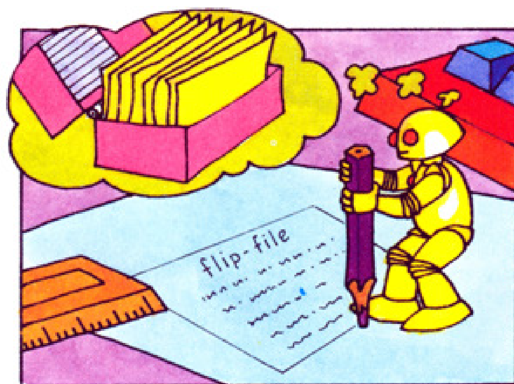
Making a subroutines library

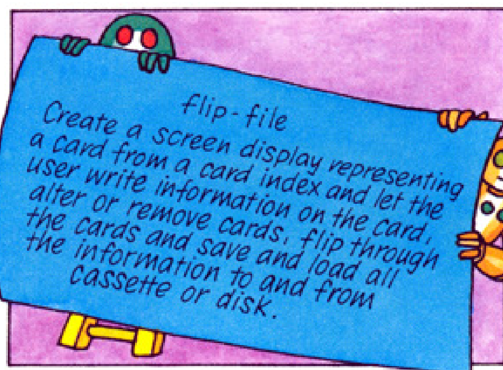


It is a good idea to collect useful subroutines which you can use in lots of different programs. As well as writing your own subroutines you can study other people's programs in books or magazines and note down useful routines.

Program planning skills

Spending time on planning your program can save you hours of debugging later. In professional software houses, a person called a systems analyst usually plans the program down to the last detail before handing it to a coder who translates it into a programming language. If you are writing your own programs it is enough to make a short list – a "computer jobsheet" – of the main tasks the computer will have to do, and then develop the program on the computer. There are some guidelines to help you on these two pages.



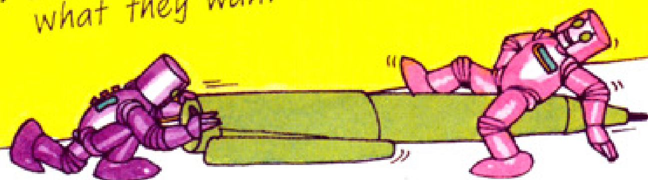


Before you write the jobsheet you need to have a clear idea of what you want the program to do. It is a good idea to write a short description of the program. For example, the picture above shows a description of the *Flip-file* program which is listed at the back of this book.

Flip-file jobsheet

flip-file jobsheet

1. Introduce program and ask users what they want to do: add a card; remove a card; alter a card; flip through cards; load cards or save a set of cards.
2. Input routine for user's choice.
3. Send computer to subroutine to carry out task user chooses.
4. Subroutines to carry out each of the tasks.
5. Branch back to main screen display to ask users what they want to do next.



Spending time on planning your program can save you hours of debugging.

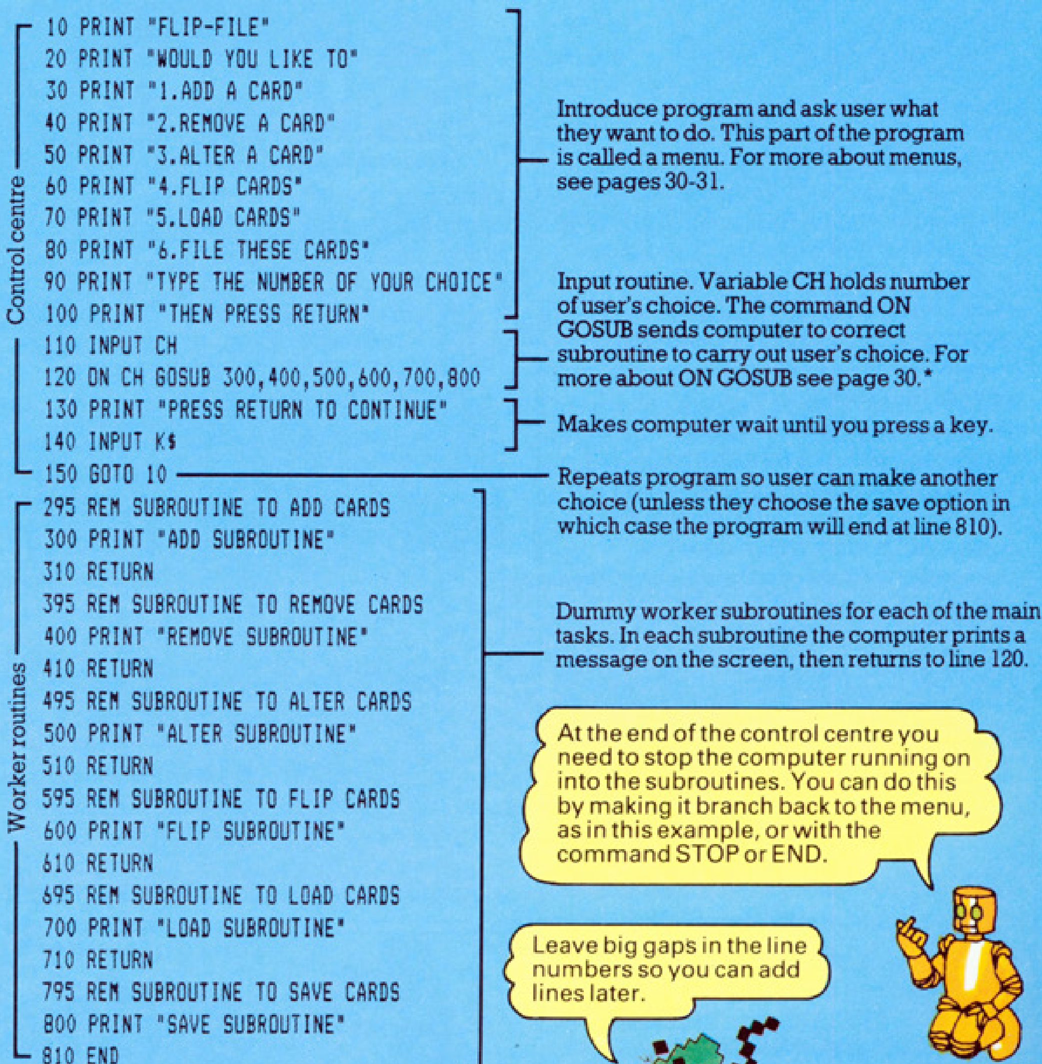
Once you have a clear idea of what you want the program to do you can write the jobsheet. This should show the main tasks the computer will need to do to carry out the program. The jobsheet for the *Flip-file* program is shown above. Keep it simple at this stage; you can always add extra features later.

Writing a skeleton program

The next thing to do is to write a skeleton version of your program. This is a very simple version of the program which shows all the main features but does not yet carry out any of the tasks. You can do this using "dummy subroutines" as shown in the skeleton version of *Flip-file* below.

A dummy subroutine is one in which you use GOSUB to tell the computer to go to a certain line and then at that line you print a message on the screen to remind yourself that this is where the subroutine should be, followed by the RETURN instruction. Using dummy subroutines enables you to run the program to test that the computer will carry out the various tasks in the correct sequence.

Flip-file skeleton



```
10 PRINT "FLIP-FILE"
20 PRINT "WOULD YOU LIKE TO"
30 PRINT "1.ADD A CARD"
40 PRINT "2.REMOVE A CARD"
50 PRINT "3.ALTER A CARD"
60 PRINT "4.FLIP CARDS"
70 PRINT "5.LOAD CARDS"
80 PRINT "6.FILE THESE CARDS"
90 PRINT "TYPE THE NUMBER OF YOUR CHOICE"
100 PRINT "THEN PRESS RETURN"
110 INPUT CH
120 ON CH GOSUB 300,400,500,600,700,800
130 PRINT "PRESS RETURN TO CONTINUE"
140 INPUT K$
150 GOTO 10
295 REM SUBROUTINE TO ADD CARDS
300 PRINT "ADD SUBROUTINE"
310 RETURN
395 REM SUBROUTINE TO REMOVE CARDS
400 PRINT "REMOVE SUBROUTINE"
410 RETURN
495 REM SUBROUTINE TO ALTER CARDS
500 PRINT "ALTER SUBROUTINE"
510 RETURN
595 REM SUBROUTINE TO FLIP CARDS
600 PRINT "FLIP SUBROUTINE"
610 RETURN
695 REM SUBROUTINE TO LOAD CARDS
700 PRINT "LOAD SUBROUTINE"
710 RETURN
795 REM SUBROUTINE TO SAVE CARDS
800 PRINT "SAVE SUBROUTINE"
810 END
```


You can write the skeleton program at the keyboard, or on paper if you prefer. Once you have got your skeleton program running you can write each subroutine as a separate unit, then slot it into the program and test it. As you write the worker subroutines, look out for tasks which can be carried out by service routines. It is quite all right to send the computer to a subroutine from within a subroutine. You can use dummy subroutines for the service routines, too, until you are ready to write them.

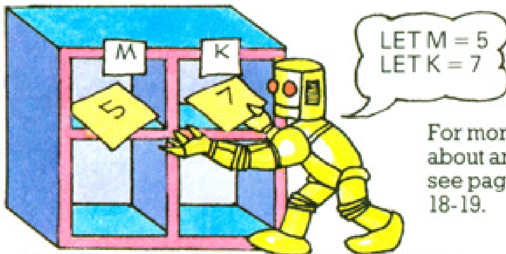
*The Spectrum does not have the command ON GOSUB. See the Guide to BASIC.

What to do first

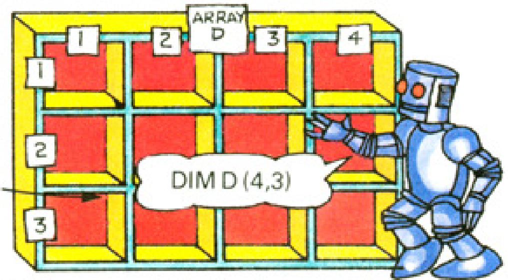
There are two things to decide before you start writing the subroutines for your skeleton program. You need to decide how you are going to store the data for the program and how you are going to arrange information on the screen.*

Data can be stored in variables or arrays. A variable is a labelled space in the computer's memory where a single item of data can be stored. An array is a way of storing several items of data under one variable name.

Setting up variables and arrays



For more about arrays see pages 18-19.



You need to give the computer starting values for the variables. If you are using an array you must tell it how many items the array will contain. This is called dimensioning an array. The process of setting up variables and arrays is called initialization and it is the first task the computer needs to carry out. It is best, though, to put the initialization routine at the end of the program, with a GOSUB command at the beginning.

```
10 GOSUB 1500
20 REM REST OF PROGRAM

1495 REM INITIALIZATION SUBROUTINE
1500 LET K=10:LET T=100
1510 DIM D(4)
1520 FOR J=1 TO 4
1530 READ D(J)
1540 NEXT J
1550 DATA 25,15,39,22
1560 RETURN
```

Lines 1520-1540: Loop to read data into array.

If you put it at the beginning it will slow the program down because of the way the computer searches for subroutines (see page 10). If you need to set your computer's graphics mode you should do this at the beginning of the initialization routine as it may affect the memory available for storing data.

Variable tricks

You can save memory space and make programs run faster by using variables for any data which is used more than once in the program. You can do this for constants, i.e, data which does not change, as well as for data which alters during the program.

LET M1$="HIT ANY KEY"

LET M2$="PLEASE WAIT"

LET M3$="YOUR SCORE IS:"

LET CL$="   "

LET L$="———"

For example, use variables for messages which will be used several times in the program. If you have lots of messages you could put them in an array and refer to them by the number of their position in the array, e.g. PRINT M$(5).

A variable containing a line of dashes is useful for drawing lines on the screen, or underlining titles. You can use PRINT TAB (or your computer's word**) to draw the line where you want it. To rub out lines or messages, use a variable containing as many spaces as there are dashes in the line, or characters in the message.

*There are ideas for screen designs on pages 28-29.

**See the Guide to BASIC.

General purpose subroutines

Once you have got your skeleton program running, and you have added an initialization routine, you can start writing the main subroutines. As you write them, look out for jobs which need to be carried out by several different worker routines and see if you can write them as service routines.

The service routines must be general purpose routines which do not contain any data which is specific to one part of the program. You can do this by using variables in the service routines and giving the variables specific values before you send the computer to the subroutine. This is called "passing a value".

General purpose message routine

```
200 LET M$="PRESS ANY KEY"
210 GOSUB 2000
.
.
530 LET M$="ANOTHER GO?"
540 GOSUB 2000
550 END
1995 REM MESSAGE SUBROUTINE
2000 PRINT M$
2010 INPUT REPLY$
2020 RETURN
```

There is a message routine like this in *Flip-file*, lines 1135-1160.

A message subroutine like this can be used anywhere in the program, or even in other programs, because each time you use it you can put a different message in the variable M$.

General purpose graphics routine

```
130 LET SIZE=10:LET X=12:LET Y=12
140 GOSUB 300
150 END
295 REM SUBROUTINE TO DRAW SQUARE
300 PLOT X,Y
310 DRAW X+SIZE,Y
320 DRAW X+SIZE,Y+SIZE
330 DRAW X,Y+SIZE
340 DRAW X,Y
350 RETURN
```

You need to convert the graphics commands for your computer.

This routine draws a square with sides SIZE units long, at screen position X,Y. To draw different squares at different positions you give the variables SIZE, X and Y different values each time you use the subroutine.

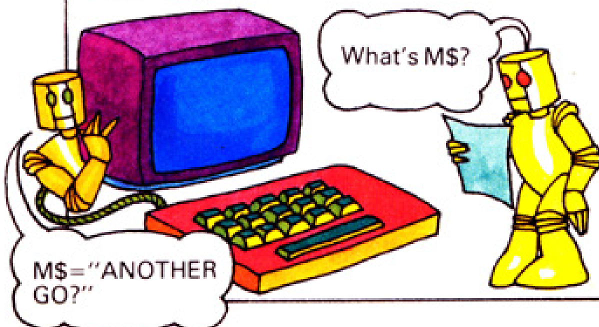
Subroutines order

As the program gets longer, it becomes quite tricky to follow the flow of action through the program and understand how it works. It helps if you arrange the subroutines in a logical order. Here are some guidelines.

1. Put all the worker subroutines first, followed by the service routines. If you mix them up it is harder to recognize the main parts of the program.

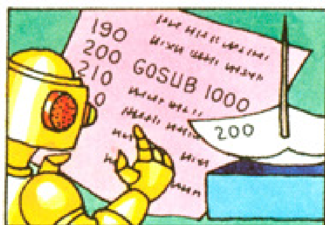
2. Within these two groups, put subroutines which do similar things together, or in the order in which they are carried out by the computer.

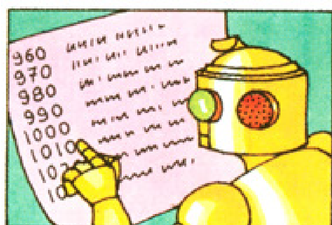
Debugging trick

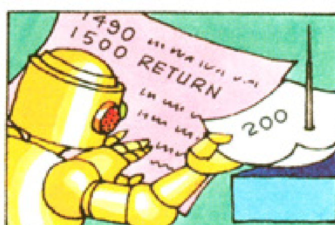


If your program breaks down with an error message and you cannot trace the error, use PRINT as a direct command (i.e. with no line number) to print the values of some of the variables. From the contents of the variables you may be able to tell what caused the bug. This is a useful debugging trick when you are developing your program.

How the computer finds subroutines







All the lines of a program are stored in the computer's memory and the computer carries out each instruction in turn. When it reaches a GOSUB command it makes a note in a special area of the memory called the GOSUB stack, of the memory location where the GOSUB command is stored.

Then it goes to the beginning of the program and looks at each line number until it finds the subroutine. It carries out the subroutine instructions until it reaches RETURN. Then it looks in the GOSUB stack to find where to return to and goes straight back to that part of the program.

Subroutine trick

If you have a subroutine for a task which needs to be carried out very quickly (e.g. a plotting routine for animated graphics), put it at the beginning of the program so the computer finds it as soon as it starts looking through the lines. You will need a GOTO command before the subroutine to make the computer jump over it the first time it runs the program.

Another kind of subroutine

Some computers have another kind of subroutine, called a procedure.* A procedure is like a subroutine, except that it is referred to by name, rather than by line number. This makes the program easier to read and understand.

```
10 LET CX=30:LET CY=45:LET RD=20
20 PROCCIRCLE(CX,CY,RD)
30 REM REST OF PROGRAM
40 END
```

CX, CY and RD are messenger variables.

```
500 DEF PROCCIRCLE(X,Y,R)
510 LOCAL A,B,I
520 MOVE X,Y:MOVE X+R,Y
530 FOR I=0.2 TO 6.4 STEP 0.2
540 LET A=X+R*COS(I):B=Y+R*SIN(I)
550 DRAW A,B
560 NEXT I
570 ENDPROC
```

X, Y and R are local variables.

X, Y and R and A, B and I are all local variables because their values are used only in the procedure.

PROC is short for procedure. It tells the computer to carry out the procedure called CIRCLE, using the values in the variables CX, CY and RD. DEF PROC is short for "define procedure" and lines 500-570 tell the computer how to carry it out. END PROC is like RETURN and tells the computer to go back to where it left the main program. The variables CX, CY and RD are used as messengers to carry values to the variables X, Y and R in the procedure. X, Y and R are local variables which means their values are used only in the procedure. You can use the same variable names elsewhere in the program and the computer will not mix them up. This feature makes procedures completely separate from the rest of a program and very useful as general purpose routines.

*The Commodore 64 with Simon's BASIC, and the BBC and Lynx all have procedures.

Defining your own functions

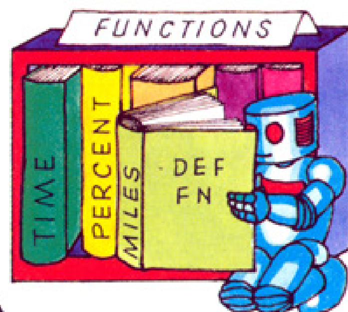
A function is a calculation which the computer can carry out on number or string data. There are several BASIC function commands. For example, the SQR function finds the square root of a number and LEN is a function which calculates the number of characters in a string. Most versions of BASIC also allow you to define your own functions and refer to them by name.

```
30 REM MILES TO KILOMETRES
40 DEF FNKILO(M)=M*1.6093
```

Tells computer how to carry out function FNKILO.

```
50 PRINT "DISTANCE IN MILES?"
60 INPUT D
70 PRINT D;" MILES IS ";
80 PRINT FNKILO(D);" KILOMETRES"
```

Makes computer carry out function FNKILO on data in variable D.



DEF FN is short for "define function" and PRINT FNname tells the computer to carry out the function. In the example above, the variable D is a messenger variable. It carries the data to variable M in the DEF FN line. Variable M is a local variable. You can have another variable called M in the program and it will not be affected by the value of M in the function.

Useful functions

Here are some functions that can be used in any programs.

Put DEF FN lines in the initialization part of a program.

Fahrenheit to centigrade

```
DEF FNCENT(F)=(F-32)/1.8
```

Converts fahrenheit temperature F to centigrade.

Centigrade to fahrenheit

```
DEF FNFAHR(C)=C*1.8+32
```

Converts centigrade temperature C to fahrenheit.

Kilometres to miles

```
DEF FNMILES(K)=K/1.6093
```

Converts kilometres distance K to miles.

Percentages

```
DEF FNPERCENT(A,B)=A/B*100
```

Calculates A as a percentage of B.

Area of a rectangle

```
DEF FNAREA(W,L)=W*L
```

L W

Works out the area of a rectangle of width W and length L.

Volume of rectangular solid

```
DEF FNVOL(L,D,H)=L*D*H
```

H L D

Calculates volume of solid with length L, depth D and height H.

Circumference of a circle

R

```
DEF FNCIRC(R)=(R*2)*3.14159265
```

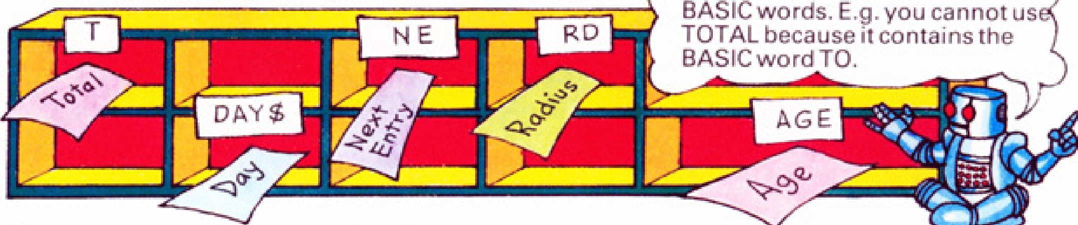
Calculates the circumference of a circle with radius R.

On some computers the function's name must be only one letter and you are only allowed to use one variable after the function name.

Variables tricks

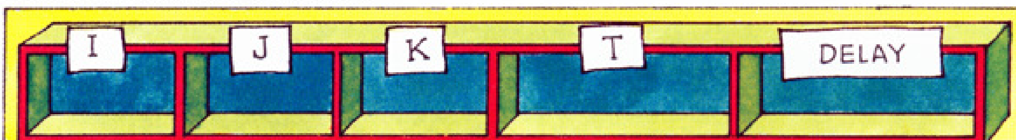
When you are developing your program it is a good idea to use words as variable names to make the program easier to read and understand. Long variable names, though, take up a lot of memory space and slow down your program, so when the program is finished you may want to shorten the names to one or two letters. Also, some computers do not allow you to use words, or will only recognize the first two letters.* Here are some tips on choosing and using variables.

Choosing variable names



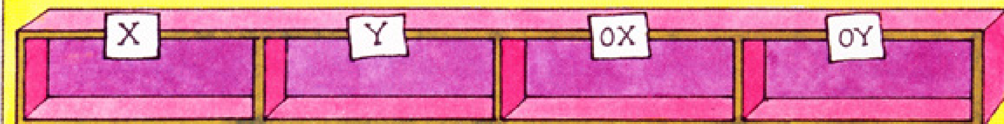
If your computer only accepts single letters as variable names, use the first letter of a word which describes the variable, e.g. T for "total". If you can use words, use a short word. For some computers, though, you must make sure that the first two letters of each name are different. It is a good idea to do this anyway, so you can easily shorten the names when the program is written. Alternatively, you could use mnemonic names, i.e. abbreviations which remind you what the variable is for, e.g. LGTH for length.

Standard variable names



These are used in FOR/NEXT loops. I stands for Iteration which means "to repeat". J and K were chosen because they follow I in the alphabet.

T (short for Time) or DELAY are often used for delay loops.



X and Y are used for TAB positions and for the co-ordinates of a pixel, (as on a graph with a horizontal X axis and vertical Y axis).

These stand for Old X and Old Y and are used when you need to calculate the next position of X and Y.

W, for Width, is for the number of pixels or characters across the screen. H, for Height, is for the number down.

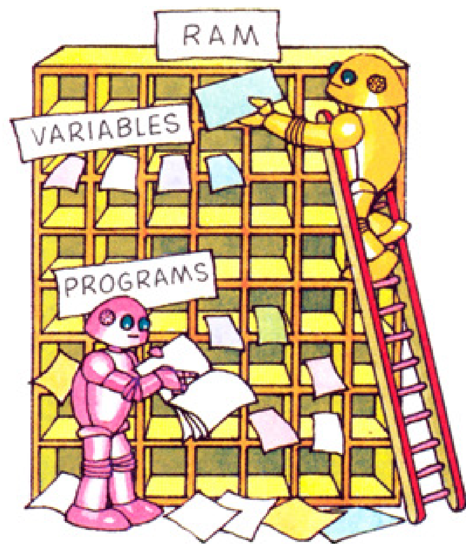
F stands for Flag, a variable used to indicate the presence of a certain condition.

These are frequently used with INPUT. They stand for Answer and Input.

It is a good idea to develop your own standard variables and use them for the same tasks in all your programs. They will be instantly recognizable to you, and will also make it easier to use the same subroutines in different programs. There are also some names which everybody tends to use for certain variables. These are shown in the picture above.

*Check the rules for your computer in your manual.

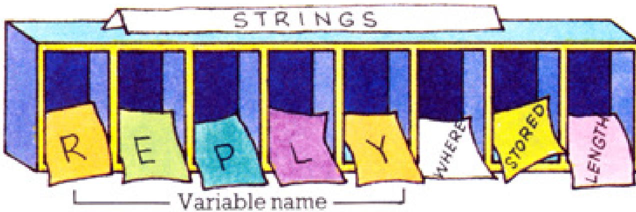
Inside the computer's memory



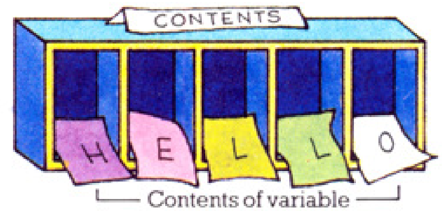
Variables are stored in a special area of the computer's RAM (random access memory), separate from the program. This enables the computer to look up the value of a variable quickly when it is carrying out a program. This is one of the reasons why you cannot use all the RAM for storing your program. For example, if you have 16K RAM, only about 12K may be available to you. The rest is used for storing variables, and as workspace for the computer while it carries out your program.

How variables are stored

The computer's memory is like lots of little numbered boxes. The boxes are called locations and the numbers are called addresses. Each box can hold one byte of computer code, i.e. the code for one letter, number or symbol. Memory size is measured in bytes and 1024 bytes make 1K (K stands for Kilobyte).

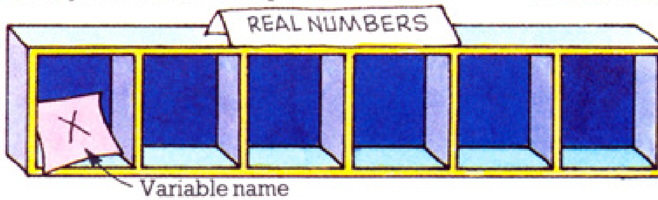


Variable name

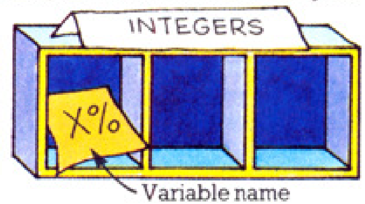


Contents of variable

For string variables, the name of the variable is stored in a different place from the contents. Each letter in the name takes one byte. Then the computer needs two bytes to indicate where the string is stored and another for how long it is. The string itself needs one byte for each character. The variable REPLY$ above would take 13 bytes.



Variable name



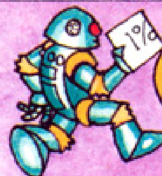
Variable name

The computer stores numbers to a precision of nine decimal places. It uses a special coding system and on most computers this takes five bytes, plus a byte for each character in the name. It uses five bytes for the number even if it is a low, whole number, i.e. one without a decimal point. Whole numbers are called integers and ones with a decimal point are called real numbers.

Some computers have a special kind of variable called an integer variable which you can use for whole numbers.* Integer variables have a % sign after their name and only take two bytes. If your computer has integer variables, use them whenever possible as they save memory space and make the program run faster.

Loop trick

If your computer allows it, use integer variables in FOR/NEXT loops. They take up less memory space and make the program run faster.




```
FOR I%=1 TO 15
READ A$
NEXT I%
```


*The VIC 20, Commodore 64, BBC, Electron and Apple all have integer variables.

IF/THEN skills

At various stages in a program you usually need to make the computer choose between several different courses of action, depending on certain conditions. You do this using the words IF/THEN.

IF | CONDITION IS TRUE | THEN | ACTION

To make the computer choose, you test data using the symbols shown on the right. These symbols are called operators. If the test is true the computer will carry out the instruction following THEN. Any BASIC statement can follow THEN, or you can send the computer to another line with GOSUB or GOTO.

Operators

- = Equal
- <> Not equal
- > Greater than
- < Less than
- >= Greater than or equal to
- <= Less than or equal to

```
IF X<40 AND X>1 AND Y<40 AND Y>1 THEN PRINT X,Y

IF N$<>A$ OR N$="" THEN PRINT "GO AWAY"

IF Y=1 AND X>0 AND X<10 OR R/2=50 THEN GOSUB 500
```

You can create more complicated tests using the words AND and OR as shown above. With AND, the THEN instruction will only be carried out if both tests are true.

With the OR test, the instruction is carried out if either condition is true. Remember to repeat the name of the data after each OR or AND.

```
50 IF A=10 THEN PRINT "CORRECT" ELSE PRINT "WRONG"

80 IF A=10 THEN PRINT "CORRECT"
90 IF A<>10 THEN PRINT "WRONG"
```

The word ELSE enables you to tell the computer what to do if the test is not true, without repeating the IF/THEN. For example, line 50 above performs the same function as lines 80 and 90, but it is shorter. (Not all computers have this command.)

```
IF X<1 THEN LET X=ABS(X) ELSE IF X>100
THEN LET X=100 ELSE IF X=Y THEN LET Y=Y*2
```

```
IF X=5 AND (Y=2 OR Y=4) THEN PRINT "TRUE"

IF X=5 AND Y=2 OR Y=4 THEN PRINT "TRUE"
```

You can string together lots of IF/THEN tests using ELSE, but be careful not to make the program difficult to read. If you use lots of ANDs and ORs in one line you may need brackets to tell the computer how to read them.

In the first example above, the computer will only carry out the THEN instruction if X is 5 *and* Y is either 2 or 4. In the second example, the test is true when X is 5 *and* Y is 2, or *whenever* Y is 4.

Following a course of action

One of the drawbacks of BASIC is that there is no neat way of giving the computer a whole series of instructions to carry out if the IF/THEN test is true. The only way you can do this is with a multiple statement line, or by sending the computer to a subroutine, or by jumping over a block of instructions with GOTO. In other languages, such as Pascal, there is no limit to the number of instructions the computer can carry out depending on a test.

1 Multiple statement lines

```
100 IF X<10 THEN LET Y=1:LET R=R+X:INPUT K:FOR J=1 TO 6: LET
X=X+K:NEXT J
```

Check the number of characters your computer will accept in one program line.

In a multistatement line starting with IF, all the statements in that line are only carried out when the IF test is true. When the test is not true, none of them is carried out and the computer jumps to the next line. This is one way of giving the computer several instructions to carry out after a test. The number of statements you can have, though, is limited by the number of characters your computer will accept in one program line (up to 255 on most computers).*

2 GOTO

```
100 IF X>1 THEN GOTO 150
110 REM ONLY IF X<1
120 REM INSTRUCTIONS
130 REM INSTRUCTIONS
140 GOTO 180
150 REM ONLY IF X>1
160 REM INSTRUCTIONS
170 REM INSTRUCTIONS
180 REM PROGRAM CONTINUES
```

Goes to line 150 if test is true.

Instructions to carry out if test is not true. Line 140 makes computer jump over other set of instructions.

You can make the computer jump over a block of instructions using GOTO as shown above. If you make it leap-frog about like this, though, it becomes very difficult to follow the flow of action through the program and this makes it difficult to read – and debug.

3 GOSUB

```
110 IF X>1 THEN GOSUB 500 ELSE GOSUB 550
120 REM PROGRAM CONTINUES
400 END
500 REM ONLY IF X>1
510 REM INSTRUCTIONS
520 RETURN
550 REM ONLY IF X<1
560 REM INSTRUCTIONS
570 RETURN
```

Instructions to carry out if X>1.

Instructions to carry out if X<1.

If the instructions will not fit in one multistatement line it is better to use a subroutine rather than GOTO. You will need one subroutine with the instructions to follow if the test is true and another if there are alternative instructions to carry out if the test is not true.

Other computer languages

The computer language Pascal does not use line numbers. Instructions are grouped together in blocks and each block is labelled with the words BEGIN and END to identify it for the computer. You can have as many instructions as you like in a block so it is easy to give the computer a whole series of instructions following a test.

The short program on the right is a program to print multiplication tables.

```
PROGRAM TABLES
VAR BASE, NUMBER, I:INTEGER;
BEGIN
  WRITELN ('WHICH MULTIPLICATION TABLE')
  WRITELN ('WOULD YOU LIKE TO SEE?')
  INPUTLN (BASE)
   FOR I=1 TO 12 DO
    BEGIN
      NUMBER:=BASE*I
      WRITELN (BASE,'X',I,'=',NUMBER)
    END;
  END;
```

Sets up variables BASE, NUMBER and I.

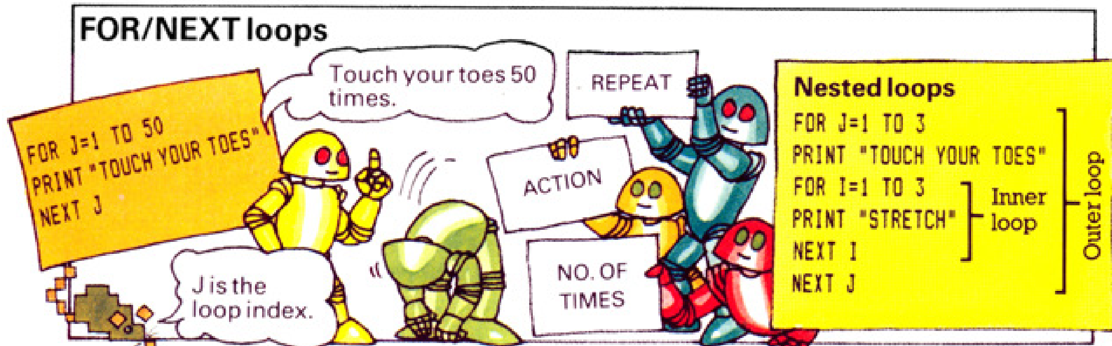
Loop to print tables.

*The Commodore 64 allows only 80 characters and the Vic 20 allows 88.

Repeating things: loops

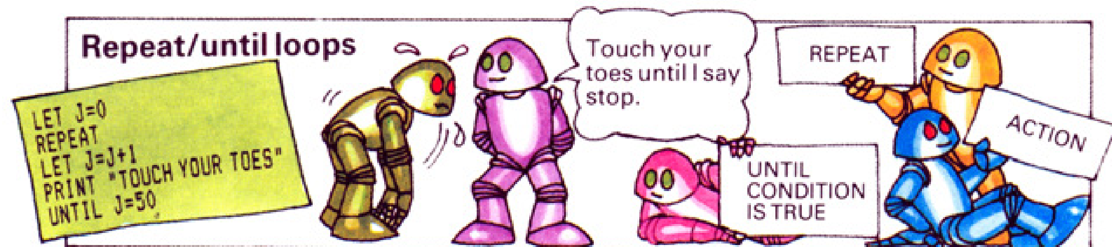
As well as making decisions you often need to make the computer repeat a set of instructions a certain number of times. A section of a program that repeats an action is called a loop. There are several different ways of making the computer loop.

FOR/NEXT loops



The simplest kind of loop is an unconditional loop where you tell the computer to repeat an action a set number of times. You do this with the words FOR, TO and NEXT as shown above. You can make more complicated repeats using loops inside loops. These are called nested loops. With nested loops, both parts of the inner loop must be inside the outer loop. The variable which counts the number of loops is called the loop index.

Repeat/until loops



Sometimes, though, you do not know how many times to repeat a loop and you want the computer to continue until a certain condition is true. For example, you may want to repeat an input line until the user types STOP, or make the computer search through an array until it finds a certain item. Some versions of BASIC have the commands REPEAT and UNTIL, but if you do not have them you will have to write a routine like one of the ones below.

```
40 REM REPEAT/UNTIL LOOP
45 DIM D$(100)
50 LET I=0
60 LET I=I+1
70 PRINT "DATA ITEM ";I
80 INPUT D$(I)
90 IF D$(I)<>"STOP" THEN
GOTO 60
```

```
40 REM DON'T DO THIS
45 DIM D$(100)
50 FOR I=1 TO 100
60 INPUT D$(I)
70 IF D$(I)="STOP" THEN GOTO 90
80 NEXT I
90 REM REST OF PROGRAM
```

This routine repeats the input line until the user inputs STOP. The variable I is the loop counter, or index, and it is good practice to set I to 0 at the beginning of the loop in case it was used somewhere else in the program.*

You should never jump out of a loop with GOTO. The computer will follow your instructions, but somewhere in its memory it will have a half-finished loop counter and this could lead to bugs later in the program.

*For Sinclair computers in line 45 you must tell the computer the number of characters in the longest item of data. E.g. DIM D$(100,12).

Tricks to terminate loops

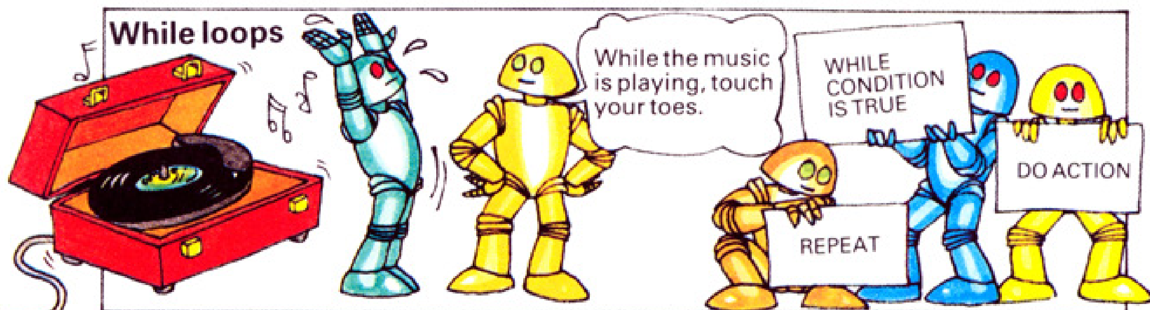
```
10 REM LOOP TO SEARCH THROUGH ARRAY
20 FOR J=1 TO 40
30 IF D$(J)=R$ THEN LET K=J
40 NEXT J
50 PRINT D$(K)
```

```
10 REM ALTERNATIVE VERSION
20 FOR J=1 TO 40
30 IF D$(J)=R$ THEN LET K=J:LET J=40
40 NEXT J
50 PRINT D$(K)
```

This sets the loop index to its final value.

These two routines show two different ways of terminating FOR/NEXT loops when the condition is true. In the example on the left, the computer searches through array D$ until it finds the item which matches R$. It keeps a record of its position in the array by setting another variable, K, to the value of the loop counter. Then it runs through the rest of the loop. If speed is important, though, you can terminate the loop by setting the loop index to its final value, as in the example on the right.*

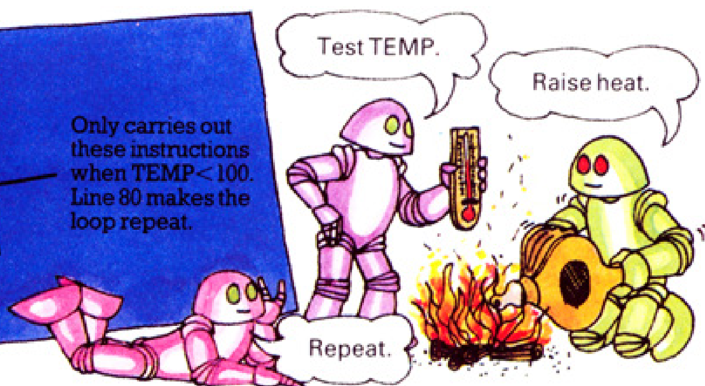
While loops



In a "while" loop, the computer repeats an action while, i.e. so long as, a certain condition is true. This sounds the same as a repeat/until loop but the difference is that the computer tests for the condition before carrying out the action. If the test is not true, it does not carry out the instructions.

```
35 REM WHILE LOOP
40 INPUT TEMP
50 IF TEMP>=100 THEN GOTO 90
60 PRINT "RAISE TEMPERATURE"
70 LET TEMP=TEMP+1
80 GOTO 50
90 PRINT "TEMPERATURE=";TEMP
```

Only carries out these instructions when TEMP<100. Line 80 makes the loop repeat.



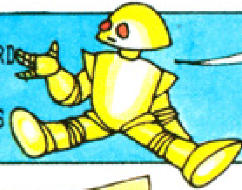
In this routine the computer will only carry out lines 60-80 when TEMP is less than 100. When TEMP is 100 it jumps to line 90. Some computer languages (e.g. Forth and Pascal) have commands for while loops, but in BASIC you have to construct a routine, like the one above, using two GOTOs. This is one of the reasons why people sometimes criticize BASIC. In fact, in this simple example you could avoid having two GOTOs by using a multistatement line: 50 IF TEMP<100 THEN PRINT "RAISE TEMPERATURE":LET TEMP=TEMP+1:GOTO 50. In a more complicated program, though, with lots of instructions to carry out so long as TEMP is less than 100, you would need both GOTOs.

*If you have a Spectrum see page 46.

Storing data in arrays

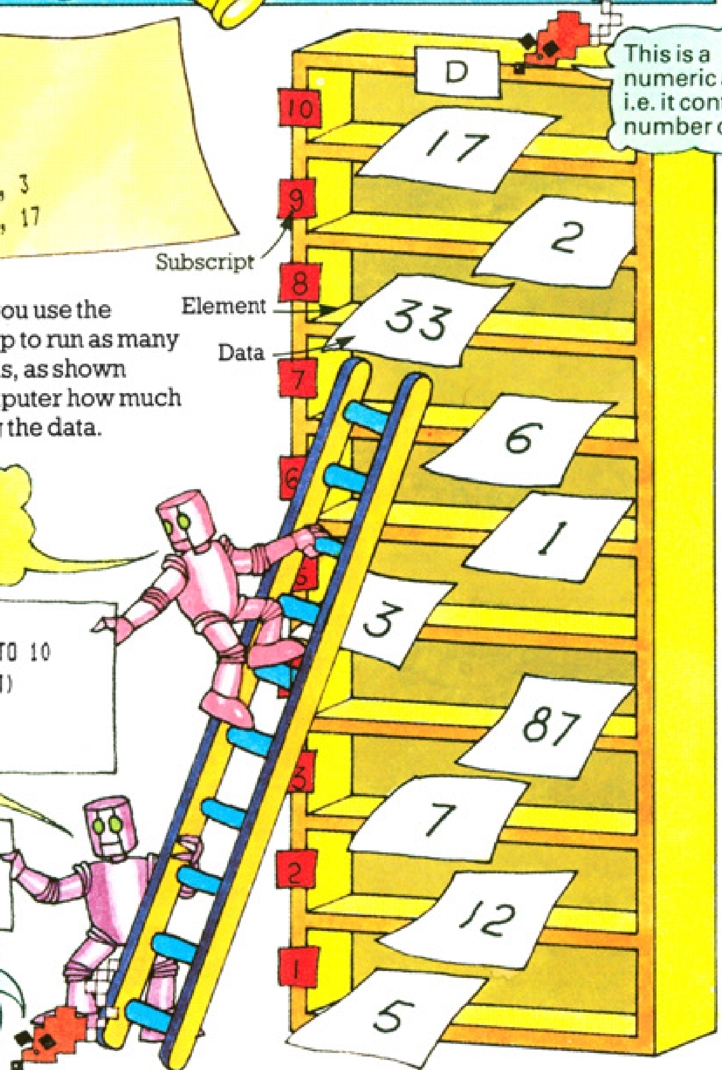
The most usual way of giving a computer large amounts of data is by listing it in the program in DATA lines, as shown below. The data has to be read into variables, or an array, before it can be used by the program. These two pages describe some of the ways in which you can use arrays. On pages 20-26 there are ideas for other ways to store data and some tricks to save memory space.

```
100 DATA GOLD RING, LAMP, SWORD
110 DATA SANDWICH, AXE, SACK
120 DATA CLOCK, RADIO, MATCHES
```



Data items are separated by commas. Some computers also need quotes round each item of string data.

```
10 DIM D(10)
20 FOR J=1 TO 10
30 READ D(J)
40 NEXT J
50 DATA 5, 12, 7, 87, 3
60 DATA 1, 6, 33, 2, 17
```



To put the data in an array you use the command READ, with a loop to run as many times as there are data items, as shown above. Line 10 tells the computer how much space to reserve for storing the data.

This loop makes the computer print out the next item in array D each time the loop repeats.

```
FOR J=1 TO 10
PRINT D(J)
NEXT J
```

This prints the fourth item in array D.

```
PRINT D(4)
```

Sinclair computers have special rules for string arrays. Check your Sinclair manual.

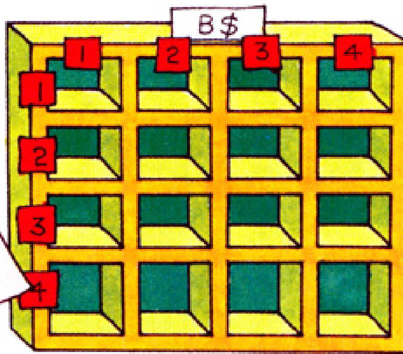
Once the data is in the array you can search through, or print out all the items using a loop. To pick out one item you use the number of its position in the array to identify it. This number is called its subscript and each position in the array is called an element. In this example, array D is a one-dimensional array. That is, it is like a list and each item has only one subscript.

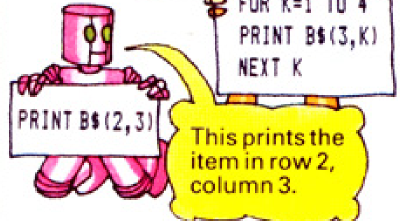
Two-dimensional arrays

If you have groups of related information, say the addresses, telephone numbers and birthdays for a number of people, you can use a two-dimensional array and put all the information about one person in one row. This makes it easier to find all the information for one person, or to look up, say, one person's birthday.

Nested loops to read data into 2-D array with 4 rows and 4 columns.

```
50 DIM B$(4,4)
60 FOR J=1 TO 4
70 FOR K=1 TO 4
80 READ B$(J,K)
90 NEXT K
100 NEXT J
```





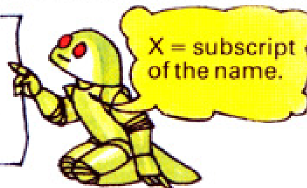
Each element in a two-dimensional array has two subscripts: the row and column numbers. To read the data into the array you use nested loops – one to count the rows and one for the columns, as shown above.

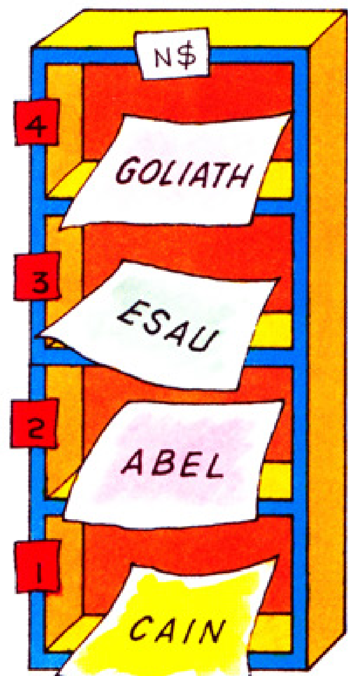
To print out all the data in one row you use a loop as shown above right. To find one particular item you just need the row and column number.

Cross-referencing arrays

When you use two-dimensional arrays you may need a second array to serve as an "index" for the first. For instance, for the example above, you could put the people's names in a one-dimensional string array. Then you could search through the names array for a particular person and look up his or her data in the other array, as shown below. The names should be in the same order as the rows containing their data in the two-dimensional array. For example, the data for the person in N$(3) should be in row three of B$.

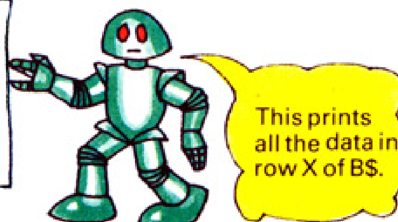
```
FOR J=1 TO 4
IF N$(J)="ESAU" THEN LET X=J
NEXT J
```





This loop makes the computer search through N$. When it finds the right name, it records the value of the loop counter, i.e. the subscript of the name, in another variable, X. This is to avoid jumping out of the loop as described on page 17.

```
FOR K= 1 TO 4
PRINT B$(X,K)
NEXT K
```



Then you can use the number in X to find the row for that person in the two-dimensional array, and print out all the data in that row. To do this you use a loop as shown above.

Data tricks

When you read data into an array, the data is stored in the memory twice – once as DATA lines in the area where the program is stored and again in an array in the area where arrays and variables are stored. This is why programs which have a lot of data, such as adventure games, take up a lot of memory. On the next few pages there are some space-saving ideas for storing data.

Arrays trick to save memory space

A two-dimensional array takes up much more space than a one-dimensional array with the same number of items. To save space you can use a one-dimensional array and organize the data in the same way as you would in a 2-D array, i.e. putting groups of related data items together. To find any item you have to use a formula to calculate its position in the array. The *Flip-file* program at the back of the book uses this method.

Formula

```
((RN-1)*RL+1)
```

(RN – 1)*RL gives the last item in the previous record, so you have to add 1.

To find the beginning of each record you use this formula. RN is the number of the record in the array (e.g. in the picture on the left, Solomon's data is record 3). RL (record length) is the number of items in each record (they must all have the same number of items).

```
FOR J=1 TO 3
PRINT D$((4-1)*3+J)
NEXT J
```

There are 3 items in each record.

To print out all the items in one record you add the value of the loop index instead of the 1 at the end of the formula.

```
PRINT D$((4-1)*3+3)
```

To find one item within a record you need to add the number of its position in the record.

Using element zero

It's usually easier to imagine arrays starting with element 1, but in fact, most computers number them from zero. So if you are short of memory space you could use element 0. To do this, make your loops to read or find data start at 0, and make the limit of the loop one less than the number of data items, as shown on the right.

```
5 REM 20 ITEMS OF DATA
10 DIM D$(19)
20 FOR J=0 TO 19
30 READ D$(J)
40 NEXT J
```

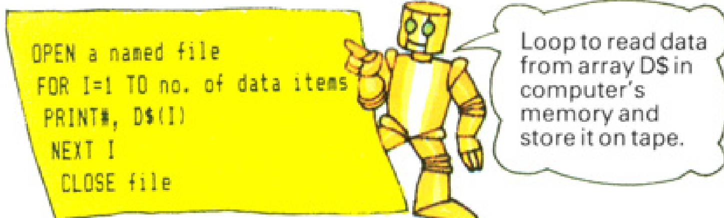
Loop repeats 20 times.

If you use element 0 you do not need to add one in the formula above.

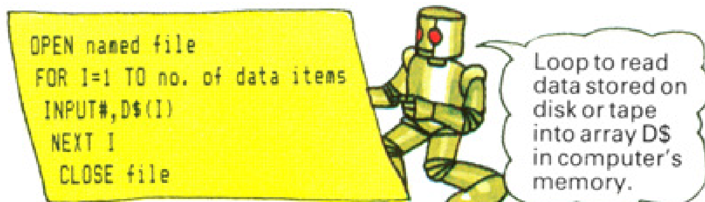
Data files trick

Instead of putting data in DATA lines in the program, you can store it on tape or disk and read it straight into an array when you run the program. This saves a lot of memory space. A list of data stored on tape or disk is called a file.

You need to find out the commands for loading and saving data for your computer. These are different on every computer although they usually follow the style shown on the right. The main ones are listed in the Guide to Basic at the back of the book.



PRINT# (print hash) is the most common command for storing data on tape. Before you store the data you have to open a file and when you have finished you must close it.



INPUT# (input hash) is the most common command for loading data from tape or disk. As with saving, you must open the file before loading the data and close it afterwards.

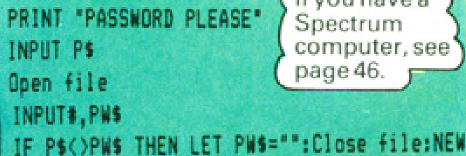
How to make a data file

1. In the initialization routine, dimension the array and add lines to read data file X. Save the program.
2. Type NEW to clear the program from the computer's memory.
3. Write a routine, as shown on the right, to store the data in file X.
4. Dimension the array to hold the data.
5. Read the data into the array.
6. Save the data in file X.
7. Run this routine.

Here are the steps to follow to store the data for a program on tape or disk instead of in DATA lines in the program. After carrying out these steps you will have a copy of your program on tape or disk, along with a data file containing all the data. When you load and run the program, the data will be loaded automatically into the array.

Protecting your files

```
LET PW$="PASSWORD"
Open file
PRINT# file, PW$ ]—— Saves password
Save data to same file
Close file
```



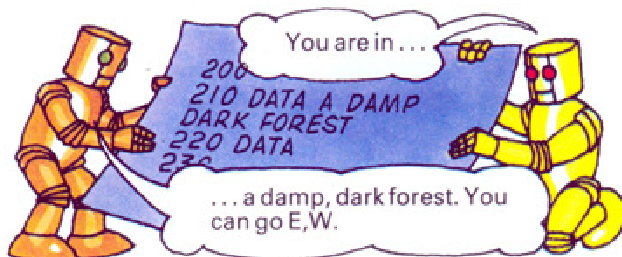
When you make your data file you could include a password so that only authorized people who know the password can load the data. You need a routine like the one above left, to save a password with your data. Then you need to add a password checking routine to the main program, just before the routine to load the data. If someone running the program types in the wrong password, the checking routine will wipe out PW$, close the file and destroy the program with the command NEW.*

*If your computer does not accept NEW in a program line, use a loop to poke numbers less than 255 into several locations with addresses between 0 and 255.

Restore trick

Here is another trick to save memory space. The command RESTORE makes the computer go to the first line of data in a program. This trick enables you to use the data listed in DATA lines without reading it all into an array. Normally you have to put the data in an array before you can use it.

```
40 PRINT "YOU ARE IN"
50 RESTORE
60 FOR J=1 TO X
70 READ D$
80 NEXT J
90 PRINT D$
```



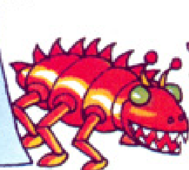
For example, you could use this routine in an adventure game to print out the description for location number X. The descriptions are stored in DATA lines. Each time the routine is carried out, RESTORE sends the computer to the first line of data. The loop makes it read each item until J=X. Then it prints the last item it read into D$ – the description for location X.

This method is slow, but if you do not clear the screen for printing D$ until the computer has found the data, the player will probably not notice.

RESTORE line number trick

The command RESTORE "line number" tells the computer to go back to a specific data line. If you have this command it is much quicker to find a particular item of data using the RESTORE trick described above.

```
50 PRINT "YOU ARE IN"
60 RESTORE ((LOC-1)*10+1200)
70 READ D$
80 PRINT D$
```



This example uses RESTORE "line number" to find the data for location LOC. The example assumes that each line holds the data for one location, the data lines start at line 1200 and the line numbers go up in tens. To find the number of the line with the data for a particular location, you subtract one from the number of the location and then multiply by ten (because the line numbers go up in tens). Then you add 1200, the number of the first data line.

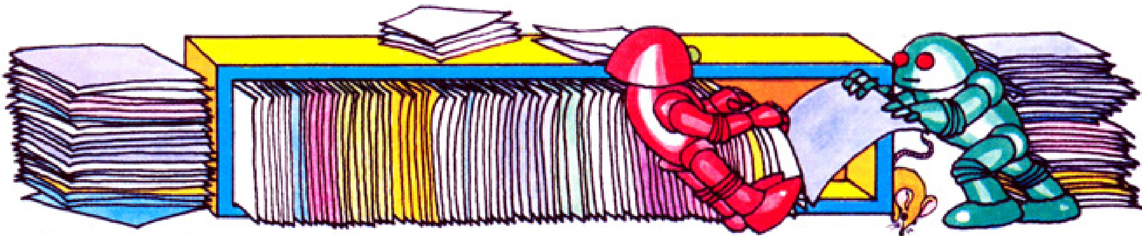
```
45 REM FOUR DATA ITEMS PER LINE
50 PRINT "YOU ARE IN"
60 LET DL=INT((LOC-1)/4)
70 LET ITEM=LOC-(DL*4)
80 RESTORE (1200+(DL*10))
90 FOR K=1 TO ITEM
100 READ D$
110 NEXT K
120 PRINT D$
```

- Line 60: DL(data line)=(LOC – 1)/4 because there are four items in each line.
- Line 70: Find position of data item in line by subtracting DL*4 (the number of items up to line holding item you want).
- Line 80: Multiply DL by ten because line numbers go up in tens.
- Lines 90–110: Loop to read each item in line up to and including ITEM.
- Line 120: Prints last item read.

If you want to put several data items in each line you need a more complicated formula, as shown above. You could use this method instead of the array trick described on page 20. Each data line should hold all the items for one record.

Data packing

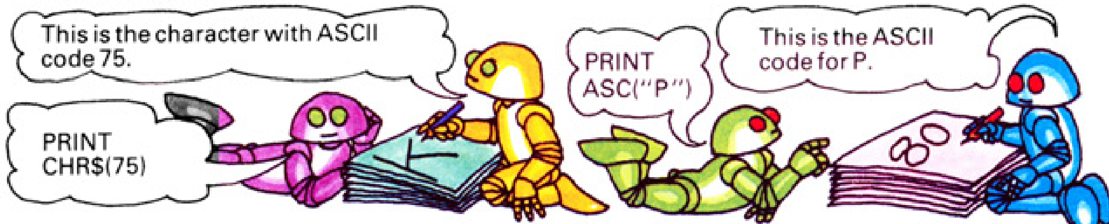
On the next few pages there are two methods of saving space when you have to store lots of numbers, for instance, all the co-ordinates for a screen display such as a maze. You can only use these tricks for numbers up to 255 because each number is stored as the character which has that number as its ASCII code. This is the code the computer uses to represent characters and there are only 256 different ASCII codes (0-255). Each character takes only one byte of memory, so you save four bytes for each number stored (numbers usually take five bytes). It is called data packing because you are squashing the numbers into a smaller space than they would normally require.



ASCII codes

Inside the computer, letters, numbers and symbols are represented by numbers coded in binary. ASCII (American Standard Code for Information Interchange) is the international standard for which number is used for each character.

The 256 different ASCII codes are used to represent all the letters of the alphabet, the numbers 0-9, punctuation marks, graphics symbols and special keyboard commands such as RETURN and SHIFT.



The BASIC command PRINT CHR$(number) tells the computer to print the character with that code number.

PRINT ASC("character") tells it to print the ASCII code for a particular character.*

Number packing trick

This trick enables you to store lots of numbers in one variable (P$). On most computers the maximum number of characters you can store in one string variable is 255, so this is the total number of numbers you can store.**

1

```
10 DIM AMOUNT(no. of numbers)
20 FOR J=1 TO no. of numbers
30 READ AMOUNT(J)
40 NEXT J
50 DATA all the numbers
```

First you need to set up an array (AMOUNT) and read all the numbers into it. Later, when the numbers are packed into the memory you can delete all these lines.

2

```
60 LET P$="*********************etc"
```

As many stars as the numbers you want to store.

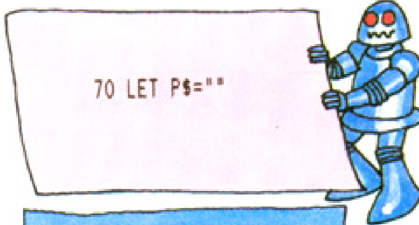
Next, set up a string variable (P$) with as many stars as you have numbers (maximum 255). This is to reserve space in the memory for the numbers. You need a string variable because the numbers will be packed as characters.

*Sinclair computers use the command CODE instead of ASC.
**If you have a Spectrum or TRS-80, see page 46.

Continued over the page.

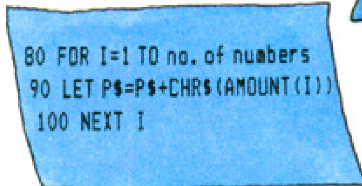
Number packing trick continued

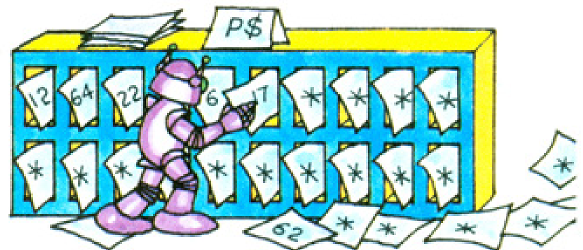
3



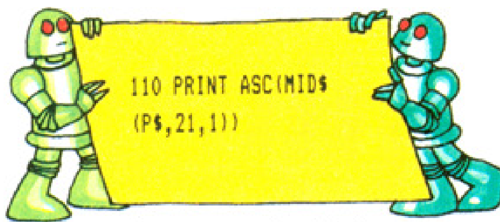
◀Then "collapse" the variable with this command. This fools the computer into thinking P$ is empty, even though the stars are still in the variables area of the memory.

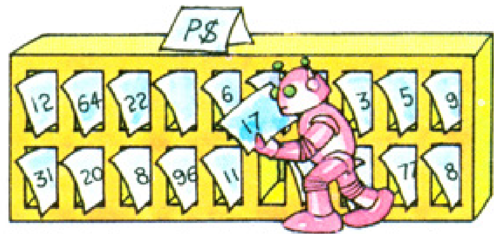
4





This loop replaces the stars in P$ with the numbers stored in array AMOUNT. The command CHR$ tells the computer to treat the numbers as ASCII codes and each number is stored as the character for that ASCII number. Because the computer thinks P$ is empty it stores each character in the space reserved by a star.





Now the numbers are packed in the memory. To retrieve any number you need to use the commands MID$ and ASC. In the example above, MID$(P$,21,1) tells the computer to take one character from P$, starting with the twenty-first character. ASC tells it to print the ASCII code for that character – your number. If you do not use ASC the computer may crash as some of your numbers will probably be codes for special effects which cannot be printed as characters on the screen.

Using the routine

```
115 REM ROUTINE TO SAVE P$
120 Open file
130 PRINT# file,P$
140 Close file
```

```
5 REM ROUTINE TO USE P$
10 LET P$="******** etc."
20 LET P$=""
30 Open file
40 INPUT# file,P$
50 Close file
60 REM REST OF
PROGRAM
```

↖As many stars as you have numbers.

Packing data like this is a good way of hiding the data from someone who might list the program.

Before you can use this method to store the data for a program you must save P$ as a data file as described on page 21 and shown on the left.

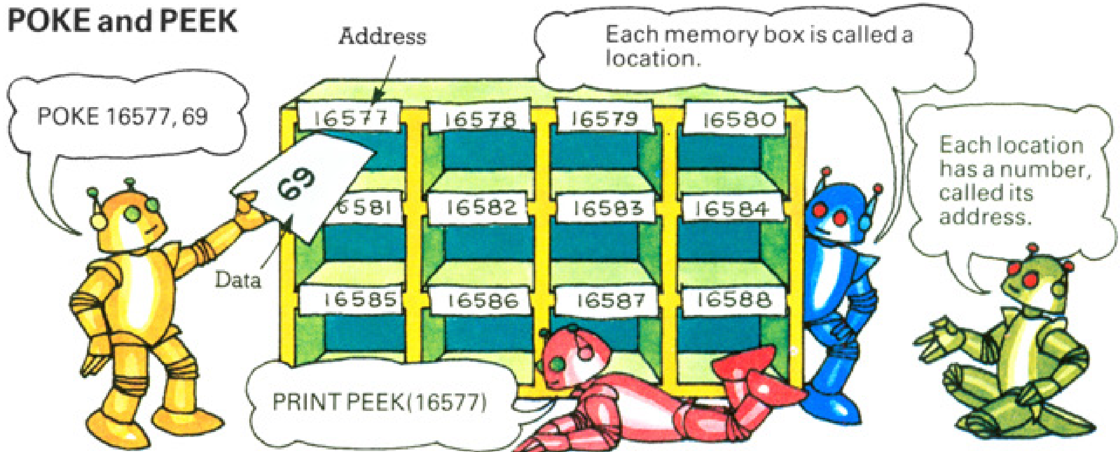
In the program which needs the data, you only need the second routine shown on the left. This reserves space for the data with a variable full of stars, collapses the variable and then loads the data from tape or disk.

You have to save P$ outside the computer because it is an "unquoted string" and the computer thinks P$ is empty. When you list the program, the data will not be listed and each time you run the program, the data in the memory will be wiped out (the command RUN clears all the data from the variables).

Another way to pack data

Here is an alternative method of packing number data. This time you use stars in a REM statement to reserve space for the data. Then you replace the stars with your data using the BASIC command POKE.* With this method the data is part of the program listing so you can run the program as many times as you like. When you save the program the data will be saved too.

POKE and PEEK

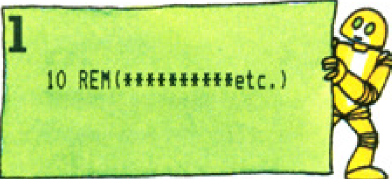


The command POKE tells the computer to store an item of data in a memory location with a certain address. The data is stored as a number which the computer thinks is an ASCII code.

PEEK does the opposite. Used with PRINT, it tells the computer to display the contents of a particular memory location i.e, the character whose ASCII code was stored there.

How to poke data

You have to be very careful using this method as if you poke data in the wrong memory locations you may confuse the computer by wiping out essential information. If this happens you have to switch the computer off and on again to get back to normal.



◀In the first line of your program you need a REM statement with as many stars as the numbers you want to store. You can put the stars in brackets to make them easier to spot.

2

```
20 FOR I=A TO A+100
30 LET C=PEEK(I)
40 PRINT "ADDRESS ";I;" = ";
50 IF C<31 OR C>127 THEN LET C=46
60 PRINT CHR$(C)
70 NEXT I
80 STOP
```

Replace A with first address where your computer stores BASIC programs.

Line 30: Peeks into each location and puts contents in variable C.

Line 50: If C<31 or >127 it is an unprintable character and is replaced with 46, the ASCII code for a full stop.

Line 60: Prints character for code C.

Next you need these lines to peek into the computer's memory and find where the stars are stored. You need to replace the variable A in line 20, with the address of the first location where BASIC programs are stored in your computer. To find this address, look at the memory map in your computer manual. (For more about memory maps, see next page.)

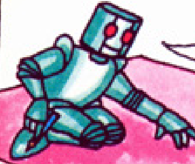
*Some computers do not use the commands POKE and PEEK. See the Guide to Basic at the back of this book.

How to poke data continued

3

```
100 DATA list your numbers here
110 DATA etc
150 FOR I=X TO X+number of data items-1
160 READ N
170 POKE I,N
180 NEXT I
```

Fill in your number of data items.



Be very careful to write down the addresses of the stars correctly. If you make a mistake, the computer will crash when you run the next part of the program.

Next, add these lines, putting your numbers in DATA lines starting at line 100. Then run the program. The computer will print the contents of each memory location and then stop at line 80. When you see the stars on the screen, write down the address of the first star and make sure that there are as many stars as there are numbers in your DATA lines.

4

```
LET X=address of first star
GOTO 100
```

Now type this LET statement as a direct command, i.e. with no line number. Then type GOTO 100. This makes the computer start running the program from line 100. Each time the loop from lines 150 to 180 repeats, the computer reads a number into N and then pokes it into location I. Now the numbers are packed in the computer's memory.

5

```
PRINT PEEK(address+data position-1)
```

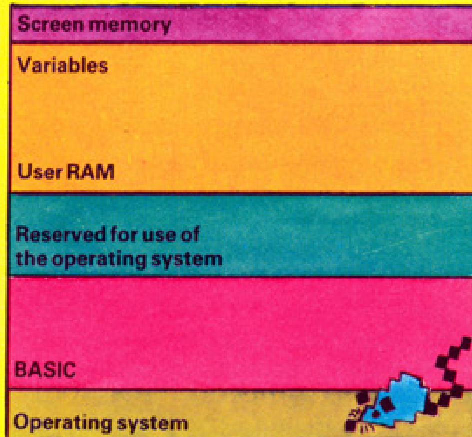
Address of first star.

Position in data list of number you want.

To find any number, use PRINT PEEK as shown above. To check the numbers are in the memory you can run the program again. Now you can delete lines 20 onwards and add line 10 to the program which needs the data. You must not list line 10 as the computer may crash when it tries to display the data on the screen. You can list the rest of the program, though, by typing LIST with a line number.

The memory map

The memory map is a diagram which shows what different areas in the computer's memory are used for. There should be a memory map in your manual. The map shows all the computer's memory, including the ROM, with the addresses of the beginning and end of each area alongside. Here are explanations of some of the terms used on a memory map.



Screen memory or **Display file.** This is where anything to be displayed on the screen is stored.

Variables. The data for variables and arrays is stored here.

User RAM or **Free RAM.** This is where your BASIC programs are stored.

Reserved for use of the operating system. The computer uses this area to keep track of what is going on.

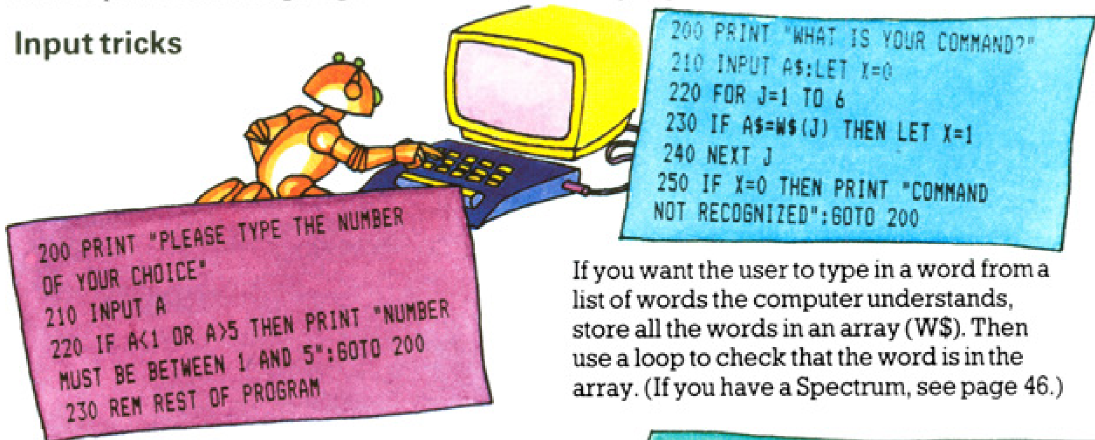
BASIC. The ROM program which enables the computer to understand BASIC.

Operating system. The program in ROM which tells the computer how to work.

Checking input

When you use the command INPUT you have very little control over what the user types in. The data the user types is immediately stored in a variable. If they type in something unexpected and it is then used in the program it can cause the program to stop with an error message. You should always check the input and make sure it is acceptable before going on with the rest of the program.

Input tricks



```
200 PRINT "PLEASE TYPE THE NUMBER
OF YOUR CHOICE"
210 INPUT A
220 IF A<1 OR A>5 THEN PRINT "NUMBER
MUST BE BETWEEN 1 AND 5":GOTO 200
230 REM REST OF PROGRAM
```

```
200 PRINT "WHAT IS YOUR COMMAND?"
210 INPUT A$:LET X=0
220 FOR J=1 TO 6
230 IF A$=W$(J) THEN LET X=1
240 NEXT J
250 IF X=0 THEN PRINT "COMMAND
NOT RECOGNIZED":GOTO 200
```

The easiest way to check input is to test it with IF/THEN as shown above. If it does not pass the test, send the computer back to ask for the input again.

```
20 IF A$<"A" OR A$>"F" THEN PRINT
"THE LETTER MUST BE BETWEEN A AND F"
```

You can check letters in the same way as numbers, as shown above. In fact, the computer compares the ASCII codes for the letters.

If you want the user to type in a word from a list of words the computer understands, store all the words in an array (W$). Then use a loop to check that the word is in the array. (If you have a Spectrum, see page 46.)

```
200 PRINT "WHAT MONTH IS YOUR BIRTHDAY?"
210 PRINT "PLEASE TYPE A FIGURE
BETWEEN 1 AND 12"
220 INPUT B$
230 IF VAL(B$)<1 OR VAL(B$)>12 THEN
GOTO 210
```

It is a good idea always to use string variables with INPUT. Then, if the user types in a word and the program is expecting a number, the program will not crash. VAL tells the computer to read a string variable as a number.

INKEY$ trick

Program line	Explanation
`400 PRINT "TYPE THE YEAR YOU WERE BORN"`	
`410 LET L=4`	No. of characters in answer you expect.
`420 LET A$=""`	Empty variable to hold answer.
`430 LET I=0`	Counter
`440 LET I$=INKEY$:IF I$="" THEN GOTO 440`	Puts next key typed in I$. Repeats line until key is pressed.
`450 LET CK=ASC(I$)`	Puts ASCII code of character in I$ into CK.
`460 IF CK>47 AND CK<58 THEN LET I=I+1:LET A$=A$+I$:PRINT I$;`	Makes sure CK is ASCII code for no. from 0-9, then adds 1 to counter and prints I$.
`470 IF I<L THEN GOTO 440`	Goes back to get another character from keyboard.
`480 LET A=VAL(A$)`	Converts A$ to a number and stores in A.
`490 IF A=0 THEN GOTO 400`	Checks A.

When you use INPUT you have no control over the length of what the user types in. If you want to check or restrict the length, use INKEY$ instead of INPUT.* INKEY$ tells the computer to take the next key typed on the keyboard and store it in a variable.

This routine uses INKEY$ with a loop so the user can only type in four characters. Another advantage of INKEY$ is that the user does not have to press RETURN, and it does not put a question mark on the screen.

*INKEY$ is not standard on all computers. See the Guide to BASIC and check your manual.

Screen display tips

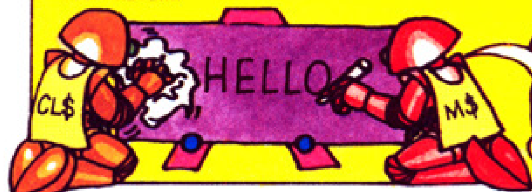
You can make your programs look really professional by arranging messages on the screen so they are clear and easy to read, and so that the screen looks interesting.

Displaying messages

A good rule to follow is always to display messages in the same area of the screen; for example, at the top or the bottom, or centred in the middle of the screen. Wherever you choose, print all your messages there so the person running the program knows where to look. You can do this with a message-printing subroutine, as shown below.

```
80 CLS
90 GOSUB 1000:REM INITIALIZATION
100 LET M$="HELLO"            Message
110 GOSUB 490
120 END
490 REM MESSAGE PRINTING ROUTINE
500 PRINT TAB(X,Y);CL$        Prints a line of spaces to clear old message off screen.
510 PRINT TAB(X,Y);M$         Prints message at column X, line Y.
520 RETURN
1000 LET CL$="       "        Make CL$ as many spaces as there are columns on your screen.
1010 LET X=column to print message   For example, to print in the top left corner,
1020 LET Y=line to print message     make X=1 and Y=1.
1030 RETURN
```

On some computers you can use the command SPC(N) to print N spaces, instead of CL$.



PRINT TAB(X,Y) tells the computer to print at column X, line Y. This is not standard on all computers so check your manual and the program conversions at the back of this book.

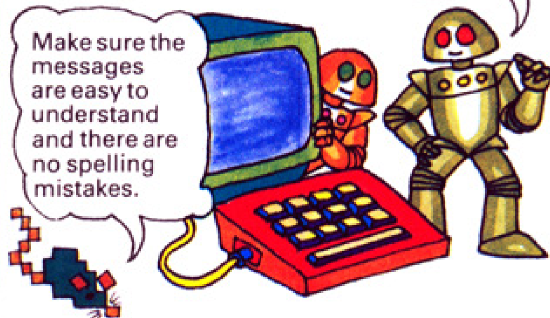
To use this message routine you need to put the number of the column and line in which you want to print your messages, in variables X and Y in the initialization part of the program. You also need a variable CL$ with as many spaces as the width of your screen, to wipe old messages off. Then each time you want to print a message, put the message in M$ and send the computer to the message-printing subroutine.

Centring messages

```
497 LET L=LEN(M$)
498 LET X=INT((W-L)/2)
1010 LET W=screen width
```

To centre messages you need to add lines 497 and 498 and replace line 1010. Line 497 calculates the number of characters in M$. Line 498 subtracts this number from the number of columns on the screen, then divides by two. INT rounds the number down if the answer is not a whole number.

Try not to leave the user looking at a blank screen. It is boring and they may think the program has crashed. To avoid this, don't clear the screen until you are ready to print the next message.



Make sure the messages are easy to understand and there are no spelling mistakes.

Displaying lots of information

If you have lots of information to give the user, think of the screen as a piece of paper and arrange the information on it with a title and border. Use headings and paragraphs to make the text easier to read. When the screen is full, do not let it scroll (i.e. automatically move up a line) or this will ruin your display. Work out the number of lines you can display on the screen and when it is full, clear the screen and start again, repeating the title and border if you like (this will be easy if you have written them as a subroutine).

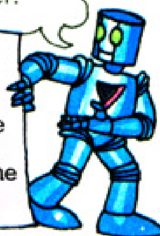
Screen border routines*

```
10 REM HORIZONTAL LINE
20 LET L$="**********etc"
30 PRINT TAB(X,Y);L$
```

◀You could use this routine to underline a title. Set up a variable with as many characters as there are in your title. Then set the variables X and Y in the PRINT TAB command, to print the line where you want it.

```
10 REM VERTICAL LINE
20 LET V$="*"+CHR$(CD)+CHR$(CL)
30 LET L$=""
40 FOR I=1 TO length of line
50 LET L$=L$+V$
60 NEXT I
70 PRINT TAB(X,Y);L$
```

After each star is printed, the cursor codes make the computer move one position down and left, ready to print the next star.



◀This routine prints a line down the screen. Replace CD and CL in line 20 with the ASCII codes for cursor down and cursor left on your computer. Set X and Y to the column and line where you want the line to start.

```
90 REM BOX ROUTINE
100 CLS
110 GOSUB 400
120 GOSUB 250
130 GOSUB 200
140 LET Y=Y+I:GOSUB 250
150 END
195 REM VERTICALS
200 FOR I=1 TO H-2
210 PRINT TAB(X,Y+I);L$            ——— Left side
220 PRINT TAB(X+W-1,Y+I);L$;       ——— Right side
230 NEXT I
240 RETURN
245 REM HORIZONTALS
250 FOR J=0 TO W-1
260 PRINT TAB(X+J,Y);L$;
270 NEXT J
280 RETURN
395 REM INITIALIZE VARIABLES
400 LET L$="*"                     ——— Or other character
410 LET H=height of box
420 LET W=width of box
430 LET X=position of left side of box
440 LET Y=position of top of box
450 RETURN
```

Here is a routine to print a border at any position round the screen. You need to fill in the variables for the size of box you want. For example, for a box round the edge of the screen, make X and Y both 1 and make H the number of lines on the screen minus 1, and W the number of columns minus 1.

Centring text in a box

Add these lines to centre text in a box drawn by the routine on the left.

```
145 GOSUB 300
295 REM CENTRE MESSAGE
300 LET C=INT((W-LEN(M$))/2)
310 PRINT TAB(X+C,Y-INT(H/2));M$
320 RETURN
405 LET M$=message
```


*See page 47 for help with converting the PRINT TAB commands in these routines

Writing menus

A menu is a way of letting the user choose what part of the program they want to run. The menu consists of a list of options from which the user can select the one they want. You can then send the computer to the correct part of the program to carry out the task the user chooses. The menu needs to be clear and easy to understand, and look interesting. On these two pages there are some ideas for ways to present menus.

```
SHAPEMAKER
Would you like to:

1) Draw a shape
2) Alter a shape
3) Use a library shape
4) Save a shape

Please type the number of your choice.
Then press RETURN.
?
```

```
SPYMASTER GAME
What would you like to do?

>Play a new game
 See the clues
 Study the codes

Please press A or Z to move the cursor
to the line of your choice.

Then press RETURN.
```

Make sure the instructions in your menus are clear.

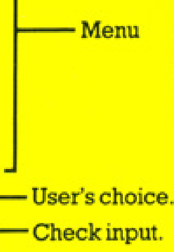
When you are planning a menu, the main decision you need to make is how users are going to indicate their choices. The simplest way is by typing in a number as in the example on the left. Another way is by moving a cursor or other symbol to point to the choice. There is a routine for a moving cursor menu on the opposite page.

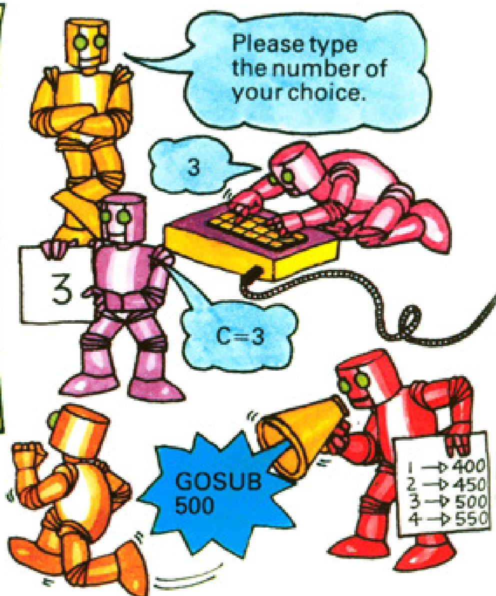
Telling the computer where to go

To tell the computer which part of the program to run you need to store the user's choice in a variable. You can do this using INPUT for a number choice menu. For the moving cursor menu you need to calculate the position of the cursor, as shown opposite. Then you can use the command ON GOSUB* to tell the computer which subroutine to go to, as shown below.

```
200 PRINT "SHAPEMAKER"
210 PRINT "Would you like to:"
220 PRINT "1) Draw a shape"
230 PRINT "2) Alter a shape"
240 PRINT "3) Use a library shape"
250 PRINT "4) Save a shape"
260 PRINT "Please type the number";
270 PRINT "of your choice."
280 PRINT "Then press RETURN."
290 INPUT C
295 IF C<1 OR C>4 THEN GOTO 260
300 ON C GOSUB 400, 450, 500, 590,
```

Menu (lines 200–280)
User's choice. (line 290)
Check input. (line 295)





You use ON GOSUB with a variable, as shown in the routine above. The value of the variable tells the computer which subroutine to carry out. For example, if C is 3, the computer will go to the third subroutine listed in line 300.

*Not all computers have the command ON GOSUB – see the Guide to BASIC.

Word input

```
250 PRINT "WHAT COLOUR DO YOU WANT?"
260 PRINT "RED, BLACK, GREEN, BLUE"
270 PRINT "PLEASE TYPE YOUR COLOUR";
280 PRINT "THEN PRESS RETURN"
290 INPUT A$
300 LET C=0
310 FOR J=1 TO 4
320 IF W$(J)=A$ THEN LET C=J
330 NEXT J
340 IF C=0 THEN PRINT "WORD NOT
RECOGNIZED, PLEASE TRY AGAIN":GOTO 270
350 ON C GOSUB etc
```

Loop to search through array holding words. (lines 310–330)

If you want the users to choose by typing a word you need to store the words from which they can choose in an array (W$). Then compare the word the user types in, with those in the array. When a match is found, store the array subscript of the word in a variable (C) to use with ON GOSUB.*

Letter input

```
200 PRINT "PLEASE TYPE A LETTER
FROM A TO F."
210 PRINT "THEN PRESS RETURN"
220 INPUT A$
230 LET A=ASC(A$)
240 IF A<65 OR A>70
THEN GOTO 200
250 LET C=A-64
260 ON C GOSUB etc.
```

Line 230: Converts letter to ASCII code.
Line 240: Checks code is in correct range for letters A-F.

To choose by typing a letter you need to convert the letter to a number which you can use with ON GOSUB. To do this, convert the letter to its ASCII code using the command ASC (or CODE on the Spectrum). Then subtract the number necessary to get a range of numbers starting with 1. For example, the ASCII codes for letters A to F start at 65, so you need to subtract 64 as shown in line 250 above.

Moving cursor menu

To use this routine you need to work out the TAB positions for each line of your menu. Then calculate the TAB numbers to position the cursor to the left of the first choice in the menu. You also need to know the number of lines from the top of the screen to the first choice in the menu, and the number of lines between each choice.**

```
270 CLS
280 GOSUB 1000:REM INITIALIZE
290 PRINT TAB(1);"SPYMASTER GAME"
300 PRINT TAB(1);"WHAT WOULD YOU LIKE TO DO?"
310 PRINT TAB(1,NL);"PLAY A NEW GAME"
320 PRINT:PRINT TAB(1);"SEE THE CLUES"
330 PRINT:PRINT TAB(1);"STUDY THE CODES"
340 PRINT
350 PRINT TAB(1);"PLEASE PRESS A OR Z TO MOVE THE"
360 PRINT TAB(1);"CURSOR TO THE LINE OF YOUR CHOICE."
370 PRINT:PRINT TAB(1);"THEN PRESS RETURN"
380 PRINT TAB(CX,CY);">":LET OY=CY
390 LET I$=INKEY$:IF I$="" THEN GOTO 390
400 IF I$="A" AND CY>NL THEN LET CY=CY-SP
410 IF I$="Z" AND CY<LC THEN LET CY=CY+SP
420 IF CY<>OY THEN PRINT TAB(CX,OY);" "
430 IF I$<>CHR$(13) THEN GOTO 380
440 LET C=INT((CY-NL)/SP)+1
450 ON C GOSUB 500,600,700
495 END
1000 LET NL=4:LET NC=3:LET SP=2:LET CX=0
1010 LET CY=NL:LET LC=NL-SP+(NC*SP)
1020 RETURN
```

Lines 380–420: Routine to move cursor.
Line 430: RETURN key = ASCII code 13.
Line 440: Calculates a value for C to use in ON GOSUB.

Flip-file uses a menu like this. See lines 885-980.

CX=Cursor X position (column)
CY=Cursor Y position (line)
NL=Number of lines from top of screen to first choice in menu.
SP=Number of lines between each choice.
NC=Number of choices.
LC=Line of last choice.

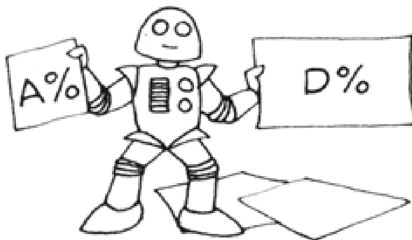
*If you have a Spectrum, see page 47.
**See page 47 for help with converting this routine.

Speeding up programs

Below there are some tricks to make programs run faster. On the opposite page there are ideas for ways to shorten or "crunch" a program so that it takes up less memory space. The shorter the program, the faster it will run, so if speed is really important you could use some of these space-saving tricks too.

Most of these tricks will be undoing the features you have written into your program to make it easier to read and understand, so it is a good idea to save a copy of the program before you crunch it. You can use the copy if you get bugs in the crunched version, or if you want to develop the program later. Here are some tricks to speed up the running of a program.

1



If your computer has integer variables, that is, variables for holding whole numbers, use them whenever possible. Integer variables have a % sign after their name (see page 13). They take up far less memory space than ordinary number variables and are handled more quickly by the computer.

2

Ordinary loop

```
FOR J=1 TO 500
PRINT J
NEXT J
```

Integer loop

```
FOR J%=1 TO 500
PRINT J%
NEXT J%
```

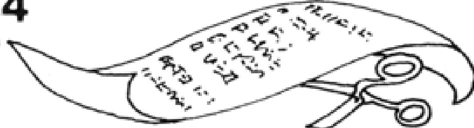
Always use integer variables (if you have them) for loop counters, as described on page 13. They run twice as fast as those with ordinary variables.

Try timing your computer with a stop-watch to see how long it takes to run an ordinary loop and an integer loop 500 times.

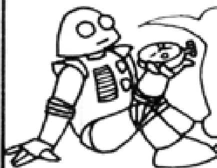
3

In a program with lots of subroutines, arrange the subroutines according to frequency of use. Put those which are used most often, just after the control centre of the program so the computer can find them quickly. If there is a routine for a task where speed is very important you could put this right at the beginning of the program, as described on page 10.

4



Shorten the program by deleting REM statements and putting several statements on one line as described opposite.



Computer speed

The speed at which a program runs also depends on the speed of your computer. The speed of a computer is measured in megahertz (MHz) and a 2MHz computer works faster than a 1MHz computer.

The speed at which they run programs also depends on how many operations they have to carry out to perform the tasks. This is controlled by the computer's software, i.e. the operating system which tells it what to do, and the interpreter which translates BASIC into the computer's own code.

Space-saving tricks

The tricks listed below and over the page save a lot of memory space, but they make programs difficult to read and debug. It is best not to use them until you have finished a program and it is running smoothly without any bugs. Save a copy of the program on tape or disk and if possible, print out a copy on paper to work on.

You could use these tricks if, for example, you want to add more data to a program and you are running out of memory, or if you want to improve the graphics, or just make the program run faster.

1

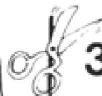
```
IF A=C THEN PRINT "YES"
IF X=Y THEN GOTO 270
IF X=Y THEN GOTO 270
LET T=3
IF R>100 THEN LET R=R-1
```

On most computers you can leave out certain BASIC commands, for example LET, THEN and GOTO. Check your manual as the rules vary on different computers, then go through your program and leave out these words.

2

SIZE ANSWER$ VOL LGTH LIMIT CHOICE CALCS

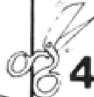
Shorten all the variable names to one or two letters. You have to be very careful doing this, because if you forget to change one occurrence of a variable you will get a bug. Make sure, too, that all the new variable names are different. This is why, when you first write the program, it is best to make sure all variables start with different letters.



3

```
395 REM MESSAGE SUBROUTINE
400 PRINT TAB(X,Y);M$
```

Delete all the REM statements. These were only necessary to make the program easy to read and are not essential to the running of the program. REMs on lines by themselves are easy to remove by just deleting the line. Make sure, though, that no GOSUB or GOTO commands send the computer to a line you are going to delete.



4

```
70IFX>1THENPRINT;X-1;" PAGES"
```

Leave out space after line number.

```
40FORI=N TOP
```

If you delete this space computer might read this as a variable NTOP.

You need these spaces.

```
1500N C GOSUB200,300,400
```

Next, leave out any unnecessary spaces. Each space takes one byte of memory, so you can save quite a lot by deleting them. Be careful, though, as some statements need spaces to make them clear.

More tricks over the page.

Programmer's toolkits

A programmer's toolkit is a special program you can buy to carry out these tasks. You load the toolkit program into your computer's memory and it will then offer you the choice of deleting selected words such as REM or LET, renumbering the program and searching for and replacing variable names. Different programs offer you different facilities. Here is an example of the range a good program would provide.

```
TOOLKIT
Please type the number of your choice.
1) Renumber program
2) Search and replace
3) Merge programs
4) List variables
5) Move lines
6) Compact program
7) Delete REMs
```

More space-saving tricks

5

```
200 FOR J=1 TO 5
210 PRINT J
220 NEXT
```

You can leave out the loop variable on most computers. The computer automatically assumes you mean NEXT J.

6

```
200PRINT"JUST A MOMENT":FORI=1TO99:
READA$(I):NEXT:PRINT:PRINT"FINISHED":
GOSUB500
```

After a subroutine the computer returns to next statement in line.

```
90LETL$=CHR$(42);GOSUB100:PRINTTAB
(X,Y);M$:LETL$=CHR$(46):GOSUB100
```

Combine as many lines as possible into multistatement lines. Do not combine IF THEN or ON GOSUB with other statements which do not depend on them. Check the maximum line length for your computer. It is usually 255 characters (that's six and a bit lines on a computer with a 40 column screen.) Make sure that subroutines still start on the lines stated in the GOSUB commands.

7

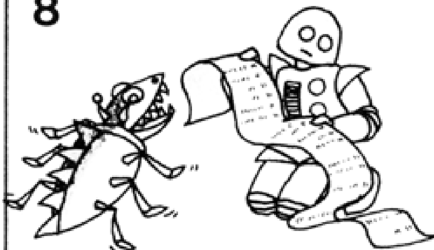
```
50 PRINT M$:GOSUB 100
```

```
50P.M$:GOS.100
```

Check your manual to see if you can use abbreviations.

On some computers you can use abbreviated forms of BASIC words, e.g. P. for PRINT. Using these enables you to fit more statements in one program line. They do not by themselves, though, save memory space because the computer stores the shortened version anyway.

8



After all these changes you may have to renumber the program. Check very carefully that all the subroutines and GOTOs go to the right line numbers.

You can also save memory space using the data tricks on pages 20-26.

More about memory

Most home computers have between about 32K and 48K of RAM and this is masses for even quite long programs. Some of the RAM, though, is always used by the computer for housekeeping tasks, that is, for storing information it needs while it carries out the program. This can take up to 3K and reduces the space available for the program itself.

Also, in high resolution graphics modes, most computers need much more RAM space for storing graphics information. In the highest resolution mode this can be up to 20K, which might only leave you about 10K for the program.

The largest memory an eight-bit computer (one which uses bytes made up of eight bits, or signals) can have, is 64K ROM and RAM combined. This is because each location in the memory has to have a number as its address. Each address is represented by two bytes of computer code and the highest number that can be made with two bytes (16 bits) is 65535. This allows 65536 locations numbered 0-65535 which is 64K (65536÷1024).

It is possible, though, to use more memory on an eight-bit computer by switching in different blocks of memory at different times. You can buy memory expansion units to do this.

Flip-file program

On the next few pages there is the listing for the *Flip-file* program. The program illustrates many of the programming techniques described in this book. It is a complicated program, with lots of variables and subroutines. Each task is carried out by a separate section of the program and it is easier to study the sections separately. The program is written in a standard version of BASIC and conversion lines for the main makes of home computer are given at the end of the listing.

How to use *Flip-file*

Flip-file is a simulation of a card index file. The program is written to store names, addresses and telephone numbers, but you could easily convert it to store other information. There are some guidelines for doing this at the end of the program.

```
FLIP--FILE
NO CARDS IN FILE
          WOULD YOU LIKE TO:
*****************************************
> ADD A CARD
  REMOVE A CARD
  ALTER A CARD
  FLIP CARDS
  LOAD CARDS
  FILE THESE CARDS

*****************************************
USE KEYS ( ) TO MOVE CURSOR UP & DOWN
PRESS RETURN TO CHOOSE AN OPTION
```

When you run the program this menu appears on the screen. To choose, you move the cursor to the line you want, then press RETURN or ENTER.

>Flip cards

When you choose the Flip Cards option the computer displays each card on the screen and you can flip through them to find the one you want.

You can also tell the computer to search for a particular line on a card. To do this you select the Flip option, then press the @ key. A blank card will appear on the screen. Type in the item you want to search for, on the correct line for that piece of information.

For example, if you want to find the cards for all the people who live in a particular city, type the name of that city on the town line. Then press RETURN. The computer will flip through the cards until it finds the one you want. You must type the item you want to search for exactly as it is written on the card or the computer will not be able to find it.

You can continue the search by hitting any key, or go back to the menu by pressing RETURN.

```
FLIP--FILE
1 CARD ON CARD 1
          ADD A CARD
*****************************************
  NAME     COMPUTER CLUB
 >STREET   75*
  TOWN
  COUNTY
  CODE
  PHONE
  INFO
*****************************************
USE KEYS ( ) TO MOVE CURSOR UP & DOWN
PRESS RETURN FOR MENU
```

To write information on a card you move the cursor to the line you want. The ★ shows where the next character will be printed.

>Alter a card

To alter a card, choose the Flip option to find the card you want. Then go back to the menu and select Alter. The card will appear on the screen and you can use your computer's delete key to rub out words.

>Remove a card

To remove a card you press the @ key. The delete message appears in the line marked INFO on that card. The card is removed when you save all the cards.

>Load cards >File these cards

You can save or load cards to and from disk or tape. The program conversion notes show how to alter the program to do this for the main makes of computer. If you are saving the cards on disk, each time you save a new set of cards, the old set will be wiped out.

Flip-file variables

Below there is a list of the main variables used in the program. It may help you to refer to this list as you study the program.

To save memory space all the data for the cards is stored in a one-dimensional array as described on page 20. The array is called C$ and each record (see page 20) contains all the information for one card. The first record is left blank as a spare card to use in the search routine (see previous page).

To find information for a card the program uses the formula "card number × card length + 1". This is the same as the formula given on page 20, except that it is not necessary to subtract 1 from the card number because the data for the cards starts at record 2 (record 1 is the spare card for the search routine).

Variable	Meaning
C$	Cards array – one-dimensional array to hold data for cards.
MX	Maximum number of cards allowed.
M$	Data for menu.
ML	Menu length, i.e. number of choices.
H$	Card headings.
CL	Card length – no. of lines on each card.
CARD	Current card number.
LAST	Number of last card in file.
SC	Search card. This the first record in array C$ and it is used as a blank card on which users type items to search for.
NC	Number of cards to be stored in or loaded from cassette or disk file.
ND	Number of cards deleted.
KP	ASCII code of key pressed.
CHOICE	User's choice from menu.
IL	Input length – maximum no. of characters allowed in input.
B$	Buffer string. Temporary variable to hold input.
S	Spacing. If S=2 there is a space between each line on card.

Variable	Meaning
L	For calculating line on which to print cursor.
OL	Old Line – previous position of cursor.
LMAX	Lowest position for cursor, i.e. bottom of menu or card.
DP	Depth. Number of lines from top of card to first line of messages.
TP	Top of card, i.e. first line of menu or heading on card.
DENT	Indent, i.e. column position for data on card.
W	Width of screen in columns.
W$	Variable holding message telling user which keys to press.
X$	Variable for messages at top of screen.
Y$ Z$	Variables for messages at bottom of screen.
L$	Line of stars.
D$	Delete message printed on card to be deleted.
UP DOWN	ASCII codes for keys to move cursor up and down.
OTHER	ASCII code for key to remove card.
BACK	ASCII code for backspace key to delete last character typed.

Flip-file listing

Here is the listing for *Flip-file*. Conversion lines for the computers listed below are given on pages 41-42. If you have another make of computer you may be able to adapt the program by studying your manual and the conversions.

Lines which need changing on all computers are marked ●. Those which need changing on only some of the computers are marked with those computers' initials. Lines marked with a * contain PRINT TAB(X,Y) commands which need converting for all computers except the BBC and Electron.

C: C64; **V:** VIC-20; **A:** Apple; **T:** TRS-80 Colour Computer with Extended BASIC; **B:** BBC; **E:** Electron; **S:** Spectrum.

Control centre

```
T    6 REM See TRS-80 conversions
    10 GOSUB1260:REM INITIALIZATION
    20 LET X$="WOULD YOU LIKE TO:"
    30 LET LMAX=ML:LET L=CHOICE
    40 GOSUB1170
TA★ 50 PRINTTAB(0,TP);
    60 FOR I=1 TO ML
A   70 PRINTTAB(1);M$(I)
    80 IF S=2 THEN PRINT
    90 NEXT I
    100 LET Y$=W$+" TO MOVE CURSOR UP & DOWN"
TS  110 LET Z$="PRESS RETURN TO CHOOSE AN OPTION"
    120 GOSUB1140
    130 GOSUB890:IF KP<>13 THEN GOTO 130
    140 LET CHOICE=L
    150 IF CHOICE=1 AND LAST=MX-1 THEN LET X$="FILE
    FULL-SAVE CARDS":GOTO 30
    160 IF LAST=0 AND CHOICE>1 AND CHOICE<>5 THEN
    LET X$="NO CARDS IN FILE":GOTO30
    170 LET X$=M$(CHOICE):LET LMAX=CL
TS  180 LET Z$="PRESS RETURN FOR MENU":LET L=1
S   190 ON CHOICE GOSUB 230,290,350,390,690,690
    200 IF CHOICE<6 THEN GOTO20
CVA 210 CLS:PRINT"CARDS SAVED. YOU ARE
    NO LONGER IN FLIP-FILE."
    220 STOP
```

Worker routine

```
    225 REM ADD A CARD TO FILE
    230 LET CARD=LAST+1
    240 LET LAST=LAST+1
    250 GOSUB810:GOSUB1140
    260 GOSUB990
    270 IF KP<>13 THEN GOTO 250
    280 RETURN
```

20: Put message in variable X$.

30: Set variables for line on which to print cursor.

40: Print heading and top message.

50: Set printing position to top of card.

60-90: Print menu. If S=2, leave a space between each line.

100-110 Set up messages for bottom of screen.

120: Print messages at bottom of screen.

130: Subroutine to make choice by moving cursor. If CHOICE<>13 (ASCII code for RETURN key), repeat subroutine.

140: Set CHOICE to current line number of cursor.

150-160: If user chooses to add cards and file is full, or if there are no cards in file and user has not chosen option 5 (load cards) then print message and repeat menu.

170: Set X$ message to print choice from menu. Set LMAX to card length.

180: Put message in Z$ and set L to print cursor on line 1.

190: Go to correct subroutine depending on menu choice.

200: Returns here after each subroutine and prints menu again if CHOICE<6 (File cards).

210-220: Stop program after filing cards.

230: Set card number to number of last card in file.

240: Add one to number of cards in file.

250: Display card contents, then print lower messages.

260: Input routine.

270: Repeat input routine until KP(key pressed)=13 (ASCII code for RETURN).

```
285 REM DELETE A CARD
290 GOSUB810
300 LET Y$="PRESS "+CHR$(OTHER)+" TO":LET Z$=X$
310 GOSUB1140
320 GOSUB890
330 IF KP=OTHER THEN LET C$(CARD*CL+CL)=D$:LET
ND=ND+1:GOTO 290
340 RETURN

345 REM ALTER A CARD
350 GOSUB810:GOSUB1140
360 GOSUB990
370 IF KP<>13 THEN GOTO 360
380 RETURN

385 REM FLIP THROUGH CARDS MANUALLY
390 LET CARD=1
400 LET Y$=W$+" TO FLIP, "+CHR$(OTHER)
+" TO SEARCH":GOSUB810
410 GOSUB1140:LET L=1:GOSUB890
420 IF KP=UP AND CARD<LAST THEN LET CARD=CARD+1
430 IF KP=DOWN AND CARD>1 THEN LET CARD=CARD-1
440 IF KP=OTHER THEN GOSUB470
450 IF KP <>13 THEN GOTO400
460 RETURN

465 REM SEARCH ROUTINE
470 LET CT=CARD:LET CARD=SC
480 FOR I=1 TO CL:LET C$(SC+I)="":NEXT I
490 LET X$="FLIP-SEARCH":LET Y$="MOVE TO A LINE
THEN TYPE SUBJECT"
TS 500 LET Z$="PRESS RETURN TO START SEARCH"
510 GOSUB810:GOSUB1140
520 LET KP=0:GOSUB990
530 IF C$(SC+L)="" THEN LET CARD=CT:RETURN
540 LET Y$="SEARCHING FOR: "+C$(SC+L):LET
Z$="PRESS ANY KEY TO CONTINUE"
550 LET KP=0: LET FOUND=0
560 FOR I=1 TO LAST
570 LET FLAG=0
580 IF C$(I*CL+L)=C$(SC+L) THEN LET CT=I: LET FLAG=1
590 LET CARD=I
```

Worker routines

290: Display card contents.

300: Messages to print at bottom of card.

310: Print messages at bottom of card.

320: Move cursor and read ASCII code of key pressed.

330: If key pressed is ASCII code for key to remove a page, let last item on card = D$.

350: Display card contents and lower messages.

360: Input routine.

370: Repeat input routine until user presses RETURN.

400: Set up messages. Display card contents.

410: Display messages. Set cursor to line 1. Go to cursor moving routine.

420-430: Adjust card number depending on which key is pressed.

440: Go to 470 if user chooses search routine.

450: If key pressed <> RETURN, repeat flip routine.

470: Store card number in temporary variable CT. Set card number to SC, i.e. card 0 which is record 1 in array C$.

480: Clear search card.

490-500: Messages

510: Display card contents and print messages.

520: Clear KP and go to input routine.

530: Abandon search routine if user does not type anything.

540: Print searching message. C$(SC+L) = item user is searching for.

550: Clear variables. FOUND = variable to record number of times item is found.

560-630: Loop to search through cards.

570: Clear variable FLAG which is used to indicate when item is found.

580: If line on card = line on search card, set CT to loop index (i.e. card number) and set FLAG to 1.

590: Set card number to loop index in order to use variable CARD in subroutines 810 and 1140.

Worker routines

```
600 GOSUB810:GOSUB1140
610 IF FLAG=1 THEN GOSUB 960 :LET FOUND=FOUND+1
620 IF KP=13 THEN LET I=LAST
630 NEXT I
640 LET CARD=CT
650 LET X$="SEARCH OVER":GOSUB1170
660 IF FOUND>0 THEN LET Z$="FOUND: "+STR$
(FOUND)+" TIMES"
670 IF FOUND=0 THEN LET Z$="DIDN'T
FIND: "+C$(SC+L)
680 RETURN
685 REM LOAD AND SAVE CARDS
●   690 IF CHOICE=5 THEN open file for
    input:input#, NC
●   700 IF CHOICE=6 THEN open file for
    output:print#, LAST-ND
A   710 GOSUB1170:GOSUB1140
S   720 LET J=0: IF CHOICE=5 THEN LET
    J=LAST:LET LAST=LAST+NC
S   730 LET J=J+1
S   740 FOR K=1 TO CL
●   750 IF CHOICE=5 THEN input#, C$(J*CL+K)
●   760 IF CHOICE=6 AND C$(J*CL+CL)<>D$ THEN
    print#, C$(J*CL+K)
S   770 NEXT K
S   780 IF J<MX AND J<LAST THEN GOTO730
●   790 close file
    800 RETURN
```

Service routines

```
    805 REM DISPLAY CONTENTS OF CARD
    810 GOSUB1170
TA★ 820 PRINT TAB(0,TP);
    830 FOR K=1 TO CL
    840 LET T$=C$(CARD*CL+K)
A   850 PRINT TAB(1);H$(K);TAB(DENT);T$
    860 IF S=2 THEN PRINT
    870 NEXT K
    880 RETURN
    885 REM MOVE CURSOR UP & DOWN
★   890 PRINTTAB(0,TP+(L-1)*S);">";
```

600: Display card contents and lower messages.

610: If item was found go to input routine for next instruction.

620: If key pressed = RETURN set loop index to final value to terminate loop as described on page 17.

640: Set CARD to number stored temporarily in CT.

650: Print card heading and top message.

660-670: Check FOUND variable and put message in Z$. STR$ converts a number variable to a string variable.

680: Back to 450.

690: If CHOICE=5 (Load cards) open a file called F$ (FLIPDAT) and load item NC (the number of cards stored in file).

700: If CHOICE=6 (File cards) open file called F$ (FLIPDAT) and save the number of cards less number of deleted cards.

710: Display messages top and bottom.

720: Set loop counter J to 0 or, for loading cards, to current number of last card. This is the beginning of a repeat/until loop (see page 16).

740: Loop to repeat once for each item on a card.

750: Load data into array C$ using formula described on page 20. J=record number, CL=record length and K=item number.

760: If last item on card <> D$ (delete message) save item in file.

780: Repeat loop if J is less than maximum cards allowed and less than last card number.

790: Close file.

810: Print heading and top message.

820: Set printing position to top of card.

830: Loop to repeat once for each item on a card.

840: Uses formula described on page 20 to copy data for next line of card into variable T$.

850: Print card heading H$ followed by T$.

860: If S=2, leave a blank line.

890: Print cursor symbol next to first line of card or menu.

```
     900 LET OL=L
     910 GOSUB960
     920 IF KP=UP AND L>1 THEN LET L=L-1
     930 IF KP=DOWN AND L<LMAX THEN LET L=L+1
   ★ 940 IF L<>OL THEN PRINTTAB(0,TP+(OL-1)*S);" ";
     950 RETURN

     955 REM WAIT FOR A KEY TO BE PRESSED
CV ABE 960 LET A$=INKEY$:IF A$="" THEN GOTO 960
   S 970 LET KP=ASC(A$)
     980 RETURN

     985 REM INPUT ONE OR MORE LINES OF "IL" LENGTH
     990 LET L=1:LET OL=1:LET B$=""
    1000 LET B$=C$(CARD*CL+L)
   S 1005 REM See Spectrum conversions
    1010 LET BL=LEN(B$):LET X=0
    1020 IF BL<IL AND KP<>UP AND KP<>DOWN
     AND KP<>BACK AND KP>31 THEN LET X=1
    1030 IF X=1 THEN LET B$=B$+CHR$(KP)
  AS 1040 IF KP=BACK AND BL>0 THEN LET
     B$=LEFT$(B$,BL-1)
   ★ 1050 PRINT TAB(DENT,TP+(L-1)*S);B$;"* ";
    1060 GOSUB890
    1070 IF L<>OL OR KP=13 THEN GOSUB1100
    1080 IF KP<>13 THEN GOTO1010
    1090 RETURN

    1095 REM UPDATE CARD WITH LINE TYPED IN
    1100 LET C$(CARD*CL+OL)=B$
   ★ 1110 PRINT TAB(DENT,TP+(OL-1)*S);C$(CARD*CL+OL);" ";
    1120 LET B$=C$(CARD*CL+L)
   S 1125  REM See Spectrum conversions
    1130 RETURN

    1135 REM DISPLAY PROMPTS AT BASE OF SCREEN
   ★ 1140 PRINTTAB(0,TP+DP);S$;
   ★ 1150 PRINTTAB(0,TP+DP);Y$:PRINTZ$;
    1160 RETURN

    1165 REM GENERAL SCREEN LAYOUT WITH CARD NUMBER
 CVA 1170 CLS
    1180 PRINT"FLIP--FILE"
    1185  IF LAST=0 THEN PRINT "NO CARDS IN FILE"
   V 1190 IF LAST>0 THEN PRINT;LAST;" CARDS";
```

Service routines

900: Record current line number of cursor in variable OL.

910: INKEY$ routine.

920-930: Calculate L (new position for cursor) depending on which key was pressed.

940: If current cursor position <> OL, wipe out cursor at position OL.

960: Put character for next key pressed into A$.

970: KP = ASCII code of key pressed.

990: Set cursor line number and old line number to 1 and clear B$ to hold new input.

1000: Put any data already in line L into B$.

1010: Count characters in B$ and record in BL. X is a flag variable to indicate when key pressed is a valid character to store in B$.

1020: Set flag variable X to 1 if BL < max. input length and key pressed <> key to move cursor up or down, or to delete character, and KP is > 31 (i.e. a printable character).

1030: If X = 1, add character KP to B$.

1040: If user presses delete key and BL (length of B$) > 0 then slice off last character from B$.

1050: Print B$ followed by prompt symbol "*"

1060: Subroutine to move cursor and input next character.

1070: If user has moved to a new line (L<>OL) or pressed RETURN, go to subroutine to put latest input into array C$.

1080: If user has not pressed RETURN, repeat input routine.

1100: Put B$ into array C$. Subscript is calculated using formula described on page 20.

1110: Print line just put into C$, followed by space to wipe out * symbol.

1120-1130: Put data in line L (current position of cursor) into B$ then return to 1080 and from there to 1010 to receive next input.

1140: Print two lines of spaces to clear lower message area.

1150: Print messages.

1180-1190: Print heading and number of cards.

Service routine

```
   1200 IF CARD<MX+1 AND LAST>0 THEN
   PRINT" CARD ";CARD;" "
   1210 LET XC=W2-(LEN(X$)/2)
   1220 PRINTTAB(XC);X$
   1230 PRINTL$
 ★ 1240 PRINTTAB(0,TP+DP-1);L$
   1250 RETURN
```

Initialization

```
    1255 REM INITIALIZATION
    1260 GOSUB1440
    1270 LET MX=30:LET SC=0:LET CL=7:LET ML=6
S   1280 DIM C$(MX*CL)
S   1290 DIM M$(ML),H$(CL)
    1300 DATA"ADD A CARD","REMOVE A CARD","ALTER A CARD"
    1310 DATA"FLIP CARDS","LOAD CARDS","FILE THESE CARDS"
    1320 FOR I=1 TO ML:READ M$(I):NEXT I
    1330 DATA"NAME","STREET","TOWN","COUNTY",
    "CODE","PHONE","INFO"
    1340 FOR I=1 TO CL:READ H$(I):NEXT I
    1350 LET LAST=0:LET CARD=1:LET ND=0
    1360 LET F$="FLIPDAT":LET D$="! CARD DELETED !"
    1380 LET CHOICE=1
AT  1390 LET UP=ASC("["): LET DOWN=ASC("]"):LET
BES OTHER=ASC("@"): LET BACK=20
    1400 LET W$="USE "+CHR$(UP)+" "+CHR$(DOWN)
    1410 LET L$="":FOR I=1 TO W:LET L$=L$+"*":NEXT I
    1420 LET S$="":FOR I=1 TO W*2:LET S$=S$+" ":NEXT I
    1430 RETURN
    1435 REM SCREEN SET UP DETAILS
VTS 1440 LET W=40:LET TP=5:LET DP=15
 VT 1450 LET S=2
  T 1460 LET W2=W/2:LET DENT=9:LET IL=W-DENT-2
    1465 REM See Commodore, Apple, BBC Electron conversions
    1470 RETURN
```

1200: Indicate current card number if not more than maximum.

1210-1220: XC is column in which to print message X$ centred in top left side of screen.

1230-1240: Print line of stars below message and across bottom of card.

1260: Subroutine with computer-specific screen details.

1270-1470: See variables chart for meanings of variables.

1280: Sets up one-dimensional array C$ to hold MX × CL items.

1300-1320: Choices for menu and loop to read them into M$.

1330-1340: Headings for cards and loop to read them into H$.*

1360: FLIPDAT is the file name used for saving cards.

1390: Puts the ASCII codes for keys chosen to move the cursor, in variables UP, DOWN, BACK and OTHER.

1400: Puts message and symbols for keys chosen to move cursor, into variable W$.

1410-1420: Sets up variable L$ to print line of stars across screen and S$ to print two lines of spaces to wipe out messages.

1460: IL (no. of characters you can type in a line)=width of screen minus left-hand margin on card minus 2.

Altering the program

To alter the program to store different information, you need to change the data for H$ in line 1330, to the headings for your cards. If you have a different number of headings you must also change CL in line 1270. If your headings contain more characters than the ones in *Flip-file*, you will need to change the variable DENT in line 1460.

Flip-file is written to hold 30 cards, but if you have enough memory you can increase the number of cards by changing MX in line 1270. Each full card takes about 200 bytes on a computer with a 40 column screen, and 100 bytes on a 20 column screen. The program itself is about 4½K, so you can work out how many cards your computer will hold.

Flip-file conversions

BBC, Electron

```
690 IF CHOICE=5 THEN FILE=OPENIN F$:INPUT
  #FILE,NC
700 IF CHOICE=6 THEN FILE=OPENOUT F$
  :PRINT#FILE,LAST-ND
750 IF CHOICE=5 THEN INPUT#FILE,C$(J*CL+K)
760 IF CHOICE=6 AND C$(J*CL+CL)<>D$ THEN
  PRINT#FILE,C$(J*CL+K)
790 CLOSE#0
960 LET A$=INKEY$(0):IF A$="" THEN GOTO
  960
1390 LET UP=ASC("["):LET DOWN=ASC("]"):LET
  OTHER=ASC("@"):LET BACK=127
1465 MODE 6
1466 *FX4,1
1467 VDU23;8202;0;0;0;
```


*For countries other than Britain you can change COUNTY to STATE.

Continued over the page.

C64, VIC-20 (★ = VIC-20 only)

Convert PRINT TAB(X,Y) commands using method given on page 45.

```
210,1170 Replace CLS with PRINT CHR$(147);
690 IF CHOICE=5 THEN OPEN 1,1,0,F$:INPUT#1
  ,NC
700 IF CHOICE=6 THEN OPEN 1,1,1,F$:PRINT#1
  ,LAST-ND
750 IF CHOICE=5 THEN INPUT#1,C$(J*CL+K)
752 IF CHOICE=5 AND C$(J*CL+K)="!!" THEN
  LET C$(J*CL+K)=""
755 IF CHOICE=6 AND C$(J*CL+K)="" THEN LET
  C$(J*CL+K)="!!"
760 IF CHOICE=6 AND C$(J*CL+CL)<>D$ THEN
  PRINT#1,C$(J*CL+K)
790 CLOSE 1
960 GET A$:IF A$="" THEN GOTO 960
★ 1190 IF LAST>0 THEN PRINT;LAST;"CARDS";
★ 1440 LET W=22:LET TP=6:LET DP=10
★ 1450 LET S=1
1465 LET C$="":FOR I=1 TO 30:LET C$=C$
  +CHR$(17):NEXT I
```

Apple

Convert PRINT TAB(X,Y) commands as described on page 45, changing TAB(0) to HTAB 1.

```
50,820 Leave out semicolons
70,850 Change TAB(1) to TAB(2)
210,1170 Replace CLS with HOME
690 IF CHOICE=5 THEN PRINT:PRINT G$;"OPEN"
  ;F$:PRINT G$;"READ";F$:INPUT NC
700 IF CHOICE =6 THEN PRINT:PRINT G$;"OPEN"
  ;F$:PRINT G$;"WRITE";F$:PRINT LAST-ND
710 Leave out this line
750 IF CHOICE=5 THEN INPUT C$(J*CL+K)
760 IF CHOICE=6 AND C$(J*CL+CL)<>D$ THEN
  PRINT C$(J*CL+K)
790 PRINT G$;"CLOSE";F$
960 LET A$="":IF PEEK(-16384)>127 THEN
  GET A$
965 IF A$="" THEN GOTO 960
1040 IF KP=BACK AND BL>1 THEN LET B$=
  LEFT$(B$,BL-1)
1045 IF BL=1 AND KP=BACK THEN LET B$=""
1390 LET UP=ASC("("):LET DOWN=ASC(")"):LET
  OTHER=ASC("@"):LET BACK=8
1465 LET G$="CTRL D"
```

Press CTRL and D keys together.

TRS-80 Colour Computer

Convert PRINT TAB(X,Y) commands as described on page 45. You will need to put brackets round the Y expression.

```
6 CLEAR 8000
50 PRINT@32*TP,"";
110,180,500 Change word RETURN to ENTER
690 IF CHOICE=5 THEN OPEN "I",#-1,F$
  :INPUT#-1,NC
700 IF CHOICE=6 THEN OPEN "O",#-1,F$
  :PRINT#-1,LAST-ND
750 IF CHOICE=5 THEN INPUT#-1,C$(J*CL+K)
760 IF CHOICE=6 AND C$(J*CL+CL)<>D$ THEN
  PRINT#-1,C$(J*CL+K)
790 CLOSE#-1
820 PRINT@32*TP,"";
1390 LET UP=ASC("("):LET DOWN=ASC(")"):LET
  OTHER=ASC("@"):LET BACK=8
1440 LET W=32:LET TP=4:LET DP=8
1450 LET S=1
1460 LET W2=W/2:LET DENT=9:LET IL=W-DENT-3
```

Spectrum

You cannot delete cards as the Spectrum saves all the cards, including the deleted ones. Use the Alter option to change cards you do not want.

Change PRINT TAB(X,Y) commands as described on page 45.

```
110,180,500 Change word RETURN to ENTER
185 LET C=CHOICE
190 GOSUB 230*(C=1)+290*(C=2)+350*(C=3)+
  390*(C=4)+690*(C>4)
690 IF CHOICE=5 THEN LOAD F$ DATA N():LET
  NC=N(1):LOAD F$ DATA C$()
700 IF CHOICE=6 THEN LET N(1)=LAST:SAVE F$
  DATA N():SAVE F$ DATA C$()
720 IF CHOICE=5 THEN LET LAST=LAST+NC
730-790 Leave out these lines
970 LET KP=CODE(A$)
1005 IF LEN(B$)>1 AND B$(LEN(B$))=" " THEN
  LET B$=B$( TO LEN(B$)-1):GOTO 1005
1040 IF KP=BACK AND BL>1 THEN LET B$
  =B$( TO BL-1)
1125 IF LEN(B$)>1 AND B$(LEN(B$))=" " THEN
  LET B$=B$( TO LEN(B$)-1):GOTO 1125
1280 DIM C$(MX*CL,22):DIM N(1)
1290 DIM M$(ML,16):DIM H$(CL,6)
1390 LET UP=CODE "(":LET DOWN=CODE ")":LET
  OTHER=CODE "@":LET BACK=12
1440 LET W=32:LET TP=5:LET DP=14
```

Guide to BASIC

On the next two pages there is an alphabetical list of the BASIC commands used in this book, with short explanations of what they do. The commands which vary on different computers have a star beside them and the conversions for the main makes of home computer are given on page 45.

AND This is for giving the computer more than one test in an IF/THEN instruction. The instructions after THEN are only carried out if both the tests linked by AND are true.

```
IF X>Y AND Y<100 THEN PRINT "TRUE"
```

***ASC** Gives the ASCII code number of a character

```
PRINT ASC("J")
74
```

ASCII code of letter J.

CHR$ Gives the character for an ASCII code number.

```
PRINT CHR$(74)
J
```

***CLS** Clears the screen.

DEF FNname This is short for "define function". You use this command to tell the computer how to carry out a certain calculation. Then, each time you want it to do that calculation you use the command FNname. The example tells the computer how to convert miles distance M to kilometres. PRINT FNKILO prints the answer for distance D.

```
DEF FNKILO=M*1.6093
INPUT D
PRINT FNKILO(D)
```

DIM Tells the computer how much space to set aside for an array. For example, DIM D$(30) sets up a one-dimensional array for 30 items of data.* DIM M(4,7) sets up a two-dimensional array for 28 items of data arranged in four rows of seven columns.

END Stops the program running.

ELSE This tells the computer what to do if an IF/THEN test is not true.

```
IF X>Y THEN PRINT X ELSE LET X=X*2
```

If your computer does not have this command you have to put the alternative instructions on a new line with another IF/THEN statement.

```
IF X>Y THEN PRINT X
IF X<=Y THEN LET X=X*2
```

FOR/TO NEXT These tell the computer to repeat a set of instructions a certain number of times. This example repeats the PRINT J instruction five times.

```
FOR J=1 TO 5
PRINT J
NEXT J
```

GOSUB Sends the computer to the subroutine which starts at a certain line number. At the end of the subroutine the command RETURN makes the computer go back to the instruction after GOSUB.

```
100 GOSUB 200
110 PRINT "NEXT INSTRUCTION PLEASE"

200 PRINT "THIS IS A"
210 PRINT "SHORT SUBROUTINE"
220 RETURN
```

GOTO Sends the computer to a specified line number from where it continues to work through the program instructions.

IF/THEN Makes the computer test some data and if the test is true, carry out the instructions following THEN. You can use the following symbols to test data:
= (equals); <> (not equals); > (greater than); < (less than); >= (greater than or equal to); <= (less than or equal to).

```
IF X>Y THEN PRINT "X IS BIGGER"
```

***INKEY$** Tells the computer to read the next character typed on the keyboard. In the example, the character is stored in the variable A$.

```
LET A$=INKEY$
```

INPUT Makes the computer wait for the user to type in some data. The data is stored in the variable given after the INPUT command.

```
INPUT T$
```

INT Converts decimal fractions to whole numbers by ignoring all the figures after the decimal point.

```
PRINT INT(5.6093)
5
```

*For string arrays on the Spectrum you must also tell the computer the length of the longest string. E.g. DIM D$(30,X) where X is the number of characters in the longest item of string data.

***LEFT$** Takes a certain number of characters from the left of a string. PRINT LEFT$(A$,3) prints 3 characters from the left of A$.

```
LET A$="ABCDEF"
PRINT LEFT$(A$,3)
ABC
```

LEN Counts the number of characters in a string.

```
LET A$="ABCDEF"
PRINT LEN(A$)
6
```

LET Puts some data into a named variable.

```
LET B$="NIGHTINGALE"
LET R=5
```

***MID$** Takes a certain number of characters from a string, starting with a specified character. The example prints 2 characters from A$, starting with character number 3.

```
LET A$="ABCDEF"
PRINT MID$(A$,3,2)
CD
```

***ON GOSUB** Sends the computer to a particular subroutine depending on the value of a variable. In the example, if C=3 the computer will go to the third subroutine listed after GOSUB, i.e. the one starting at line 250.

```
ON C GOSUB 100,170,250,400,700
```

OR This is used in IF/THEN instructions to give the computer more than one test. When tests are linked with OR the THEN instructions are carried out if either test is true.

```
IF X>Y OR Y>100 THEN PRINT "TRUE"
```

***PEEK** Makes the computer look at the contents of a particular memory location. PRINT PEEK(address) prints the contents of that location. The BBC and Electron use PRINT ?address instead of PEEK.

***POKE** Stores a number in the memory location specified in the command. E.g. POKE 4057,64 puts the number 64 into location 4057. The BBC and Electron use ?4057=64.


```
POKE 4057,64
```

PRINT Tells the computer to display some data, or the result of a calculation, on the screen. By itself, PRINT leaves an empty line.

```
PRINT "ANSWER"
PRINT LEN(B$)
PRINT 256
```

READ/DATA The command READ tells the computer to store the data listed in lines starting with the word DATA, in variables or an array. It is usually used with a FOR/NEXT loop to repeat the READ instruction.

```
FOR J=1 TO 5
READ B(J)
NEXT J
DATA 43,67,123,34,46
```

REM This is short for reminder or remark. The computer ignores anything following the word REM and it is used to put explanatory notes in a program.

RESTORE This is used with READ and DATA. Each time you use the command READ the computer reads the next item in the data list. The command RESTORE makes it start reading the data from the beginning again. Some computers have a command RESTORE "line number" which makes them read the data from a particular line onwards.

RETURN See GOSUB

***RIGHT$** Takes a certain number of characters from the right of a string.

```
LET A$="ABCDEF"
PRINT RIGHT$(A$,3)
DEF
```

STEP This tells the computer how to count the repeats in a FOR/NEXT loop. In the example the computer counts to 6 in steps of 2 and so repeats the loop three times.

```
FOR J=1 TO 6 STEP 2
PRINT J
NEXT J
```

VAL This tells the computer to treat a string as a number and is useful when you have stored numbers in a string variable.

```
LET A$="1066"
LET A=VAL(A$)
```

Conversion chart

Where no conversion is given, the computer uses the standard command.

Standard BASIC	C64/VIC-20	Apple	TRS-80	BBC/Electron	Spectrum
`PRINT ASC("J")`					`PRINT CODE("J")`
`CLS`	`PRINT CHR$(147)`	`HOME`			
`LET I$=INKEY$`	`GET I$`	`GET I$`		`LET I$=INKEY$(X)` X = length of time for computer to wait.	`INKEY$` Key pressed is stored in variable called INKEY$.

`PRINT LEFT$(A$,3)`
`PRINT MID$(A$,3,2)`
`PRINT RIGHT$(A$,3)`
					`PRINT A$(1 TO 3)`
					`PRINT A$(3 TO 5)`
					`PRINT A$(3 TO 6)`
					A$ contains 6 characters.

`PRINT TAB(X,Y);M$`	See below.	`HTAB X:VTAB Y:PRINT M$`	`PRINT @32*Y+X,M$`		`PRINT AT Y,X;M$`

Cassette file handling commands
"name" = name of file.
D$ = data to be saved.
 	
The commands given for the Apple are disk commands. You can only save integer arrays on cassette.
	`REM SAVING D$`
	`OPEN 1,1,1,"name"`
	`PRINT#1,D$`
	`CLOSE 1`
	
	`REM LOADING D$`
	`OPEN 1,1,0,"name"`
	`INPUT# 1,D$`
		`REM SAVING DATA`
		`PRINT D$;"OPEN";"name"`
		`PRINT D$;"WRITE";"name"`
		`PRINT variable name`
		`PRINT D$;"CLOSE";"name"`
		`REM LOADING DATA`
		`PRINT D$;"OPEN";"name"`
		`PRINT D$;"READ";"name"`
		`INPUT variable name`
		`PRINT D$;"CLOSE";"name"`
		Put this command in the initialization part of program.*
		`LET D$="CTRL D"`
			`REM SAVING D$`
			`OPEN "O",#-1,"name"`
			`PRINT#-1,D$`
			`CLOSE#-1`
			
			`REM LOADING D$`
			`OPEN "I",#-1,"name"`
			`INPUT#-1,D$`
			`CLOSE#-1`
				`REM SAVING D$`
				`LET A=OPENOUT "name"`
				`PRINT# A,D$`
				`CLOSE# A`
				
				`REM LOADING D$`
				`LET A=OPENIN "name"`
				`INPUT# A,D$`
				`CLOSE# A`
					`REM SAVING D$`
					`SAVE "name" DATA D$()`
					`REM LOADING D$`
					`LOAD "name" DATA D$()`
					
					N.B. You can only save arrays. You cannot save variables.

PRINT TAB(X,Y) on C64 and VIC 20

To print at column X, line Y put this line in the initialization part of program:

```
LET C$="":FOR J=1 TO number of lines on
screen:LET C$=C$+CHR$(17):NEXT J
```

CHR$(17) is the ASCII code for cursor down. This line sets up a variable C$ with as many codes as there are lines on your screen.

Use this instruction to print M$ at screen position X,Y:

```
PRINT CHR$(19);LEFT$(C$,Y);SPC(X);M$
```

CHR$(19) is the ASCII code to move cursor to top left of screen. LEFT$(C$,Y) takes Y cursor down codes from C$. SPC(X) moves X spaces across the screen.

Spectrum ON GOSUB

The Spectrum does not have this command. Instead, you can make the computer calculate the correct line to go to. For example, here is the conversion for line 120 in the *Flip-file* skeleton on page 7.

```
120 GOSUB CH*100+200
```

When CH is 1 the computer will go to line 300. When CH is 2, it will go to line 400, etc. When you write your own programs, choose subroutine line numbers which are suitable for a GOSUB calculation.

*Press CTRL and D keys together (nothing appears on the screen).
N.B. You cannot use D$ as a variable elsewhere in the program as it is used in this command.

Converting the routines in this book

On these two pages there are some conversions to help you run the routines in this book on the following makes of home computer: Commodore 64, VIC 20, Apple, TRS-80 Colour Computer with Extended BASIC, and the BBC, Electron and Spectrum. If you have another make of computer you may be able to convert the routines by studying your manual and the listings given here.

Page 17 Spectrum Loop tricks

For the Spectrum you must tell the computer how many characters to look for in D$, as shown below.

```
30 IF D$(J)( TO LEN(R$))=R$ etc.
```

Page 21 Spectrum Password routines

You can only save arrays as data files on the Spectrum, you cannot save variables. In the Password routine, you can store the Password in an array, as shown below.

```
REM PASSWORD ROUTINE
DIM P$(1,no. of characters)
LET P$(1)="password"
SAVE "file name" DATA P$()
Save data to same file
```

Line 20 sets up an array PW$ with one element. You need to fill in the number of characters in your password.

```
REM CHECKING PASSWORD
PRINT "PASSWORD PLEASE"
INPUT Q$
LOAD "file name" DATA P$()
IF Q$<>P$(1) THEN LET P$(1)="":NEW
```

You need these lines for the Password checking routine.

Page 23 Number packing trick–TRS-80

On the TRS-80, if you have more than 200 numbers you must make space in the variables area of the memory using the command CLEAR followed by the number of numbers you want to store.

Page 23 Spectrum number packing

For the Spectrum, you must store the numbers in an array called P$, so that you can save it as a data file. To do this, alter the program as shown below.

```
5 DIM P$(1, no. of numbers)
60 FOR I=1 TO no. of numbers
70 LET P$(1,I)=CHR$(AMOUNT(I))
80 NEXT I
90 PRINT CODE(P$(1,21))
120 SAVE "file name" DATA P$()
```

Delete lines 100, 110, 130 and 140.

This sets up an array P$ with one element to hold as many characters as you have numbers. You are reserving memory space by using an array so you do not need to fill P$ with stars. Lines 60-80 put the character code for each of the numbers stored in array AMOUNT, into array P$. Line 90 prints the ASCII value of the 21st character in P$. Line 120 saves array P$ as a data file.

In the routine to use P$ you need to replace lines 10-50 with the following two lines:

```
10 DIM P$(1, no. of numbers)
20 LOAD "file name" DATA P$()
```

Page 27 Checking input

For the Spectrum you need to tell the computer how many characters to check.

```
230 IF A$=W$(J)( TO LEN(A$)) THEN LET X=1
```

Page 27 INKEY$ trick

This routine will run on the TRS-80. Here are the conversions for the other computers.

C64, VIC 20, Apple

```
440 GET I$:IF I$="" THEN GOTO 440
```

BBC, Electron

```
440 LET I$=INKEY$(0):IF I$="" THEN GOTO 440
475 PRINT
```

Spectrum

```
450 LET CK=CODE(I$)
465 FOR T=1 TO 80:NEXT T
```

Pages 28-29 Screen display tips

These routines will run on the BBC and Electron. Here are the conversions for the other computers.

C64, VIC 20

These conversions use the PRINT TAB method described on page 45.

Message routine

```
80 PRINT CHR$(147)
500 PRINT CHR$(19); LEFT$(C$,Y);SPC(X);CL$
510 PRINT CHR$(19);LEFT$(C$,Y);SPC(X);M$
1025 LET C$="":FOR J=1 TO no.of lines on
screen:LET C$=C$+CHR$(17):NEXT J
```

25 for C64; 23 for VIC 20.

Horizontal line

```
25 LET C$=as line 1025 above.
30 PRINT CHR$(19);LEFT$(C$,Y);SPC(X);L$
```

Vertical line

Cursor down=17; cursor left=157

```
25 LET C$=as line 1025 above
70 PRINT CHR$(19);LEFT$(C$,Y);SPC(X);L$
```

Box routine

```
100 PRINT CHR$(147)
210 PRINT CHR$(19);LEFT$(C$,Y+I);SPC(X);L$
220 PRINT CHR$(19);LEFT$(C$,Y+I);
SPC(X+W-1);L$;
260 PRINT CHR$(19);LEFT$(C$,Y);SPC(X+J);L$;
445 LET C$= as line 1025 above
```

Centring text

```
310 PRINT CHR$(19);LEFT$(C$,Y-INT(H/2));
SPC(X+C);M$
```

Apple

Message routine

```
80 HOME
500 HTAB X:VTAB Y:PRINT CL$
510 HTAB X:VTAB Y:PRINT M$
```

Horizontal line

```
30 HTAB X:VTAB Y:PRINT L$
```

Vertical line

Cursor down=10; cursor left=8.

```
70 HTAB X:VTAB Y:PRINT L$
```

Box routine

```
100 HOME
210 HTAB X:VTAB Y+I:PRINT L$
220 HTAB X+W-1:VTAB Y+I:PRINT L$;
260 HTAB X+J:VTAB Y:PRINT L$;
```

Centring text

```
310 HTAB X+C:VTAB Y-INT(H/2):PRINT M$
```

TRS-80 Colour Computer

Message routine

```
500 PRINT @32*Y+X,CL$
510 PRINT @32*Y+X,M$
```

Horizontal line

```
30 PRINT @32*Y+X,L$
```

Vertical line

This routine will not work on this computer.

Box routine

```
210 PRINT @32*(Y+I)+X,L$
220 PRINT @32*(Y+I)+X+W-1,L$;
260 PRINT @32*Y+X+J,L$;
```

Centring text

```
310 PRINT @32*(Y-INT(H/2))+X+C,M$;
```

Spectrum

Message routine

```
500 PRINT AT Y,X;CL$
510 PRINT AT Y,X;M$
```

Change CL$ to C$. Change END to STOP.

Horizontal line

```
30 PRINT AT Y,X;L$
```

Vertical line

You cannot use this method on the Spectrum.

Box routine

```
210 PRINT AT Y+I,X;L$
220 PRINT AT Y+I,X+W-1;L$;
260 PRINT AT Y,X+J;L$;
```

Centring text

```
310 PRINT AT Y-INT(H/2),X+C;M$
```

Page 31 Word input

For the Spectrum you need to tell the computer how many characters to compare.

```
320 IF W$(J)(TO LEN(A$))=A$ THEN LET C=J
```

You will also need to convert the ON GOSUB command as described on page 45.

Moving cursor menu

C64, VIC 20

```
270 PRINT CHR$(147)
310 PRINT CHR$(19);LEFT$(C$,NL);SPC(1);
"PLAY A NEW GAME"
380 PRINT CHR$(19);LEFT$(C$,CY);SPC(CX);
">":LET OY=CY
390 GET I$:IF I$="" THEN GOTO 390
420 IF CY<>OY THEN PRINT CHR$(19);
LEFT$(C$,OY);SPC(CX);" "
1015 LET C$="":FOR J=1 TO no. of lines
on screen:LET C$=C$+CHR$(17):NEXT J
```


Continued over the page.

Moving cursor menu

Apple

In lines 290, 300, 320, 330, 350, 360, 370, change the PRINT TAB(1) command to PRINT TAB(2).

```
270 HOME
310 HTAB 2:VTAB NL:PRINT "PLAY A NEW GAME"
380 HTAB CX:VTAB CY:PRINT ">":LET OY=CY
390 GET I$:IF I$="" GOTO 390
420 IF CY<>OY THEN HTAB CX:VTAB OY:PRINT " "
```

In line 1000 set the variable CX to 1.

TRS-80 Colour Computer

Delete line 370.

```
310 PRINT @32*NL+1,"PLAY A NEW GAME"
380 PRINT @32*CY+CX,">";:LET OY=CY
420 IF CY<>OY THEN PRINT @32*OY+CX," ";
```

BBC, Electron

```
390 LET I$=INKEY$(0):IF I$="" THEN GOTO 390
```

Spectrum

```
310 PRINT AT NL,1;"PLAY A NEW GAME"
380 PRINT AT CY,CX;">":LET OY=CY
390 IF INKEY$="" THEN GOTO 390
400 IF INKEY$="A" AND CY>NL THEN LET CY=CY-SP
410 IF INKEY$="Z" AND CY<LC THEN LET CY=CY+SP
420 IF CY<>OY THEN PRINT AT OY,CX;" "
430 IF INKEY$<>CHR$(13) THEN GOTO 380
450 GOSUB C*100+400
```

Index

address, 13, 21, 25, 26, 34
AND, 13, 43
Apple, 13, 45, 46, 47
arrays, 8, 18, 19, 21, 22 27, 31
ASC, 23, 31, 43, 45
ASCII codes, 23, 24, 25, 27, 29, 31
BASIC commands, abbreviations, 34
BBC, 10, 13, 44, 45, 46, 47
brackets, 13
bytes, 13, 23, 34
cassette file handling commands, 45
centring messages, 28
CHR$, 23, 24, 43
CLS, 43, 45
CODE, 23,31
Commodore 64, 10, 13, 15, 45, 46, 47
computer languages, 15
constants, 8
cross-referencing arrays, 19
data files, 21
DATA lines, 20, 21, 26
data packing, 23
data storing, 8, 18, 19, 22
  on tape, 21
debugging trick, 9
DEF FN, 11, 43
DEF PROC, 10
delay loops, 12
DIM, 43
dimensioning an array, 8
direct command, 9, 26
display file, 26
displaying messages, 28
dummy subroutines, 7
eight-bit computer, 34
Electron, 13, 44, 45, 46, 47
element, 18, 19
element zero in an array, 20
ELSE, 13, 43
END, 7, 43
field, 20
file, 21
flag variables, 12
Flip-file program, 5, 6, 7, 20, 31, 35
FOR/NEXT loops, 12, 13, 16,43
Forth, 17
free RAM, 26
function, 11
general purpose subroutines, 9, 10
graphics mode setting, 8
GOSUB, 4, 5, 7, 8, 10, 13, 15, 33, 34, 43
GOSUB stack, 10
GOTO 10, 13, 15, 16, 17, 26, 33, 34, 43
housekeeping tasks, 34
IF/THEN, 14, 15, 27, 33, 43
initialization, 8, 9, 11, 21
INKEYs, 27, 43, 45
INPUT, 12, 27, 30, 43
INPUT #, 21
INT, 28, 43
integer variables, 13, 32
interpreter, 32
Kilobytes, 13
LEFT$, 44, 45
LEN, 11, 44
LET, 33, 4
line length, maximum, 15, 34
line numbers, 15
LIST, 26
local variables, 10, 11, 34
location, 13, 25, 34
loop index, 16, 17, 20
loops, 16, 18, 20, 32
loop trick, 13
Lynx, 10
megahertz, MHz, 32
memory, 10, 13, 26, 34
memory map, 25, 26
menu, 7, 30, 31
MID$, 44, 45
mnemonic variable names, 12
moving cursor menu, 30, 31
multiple statement lines, 5, 15, 17, 34
nested loops, 16, 19
NEW, 21
number data, 11
number packing trick, 23-24
numeric array, 18
ON GOSUB, 7, 30, 31, 34, 44, 45
one-dimensional arrays, 18, 19, 20,36
operating system, 26, 32
operators, 13
OR, 13, 44
Pascal, 15, 17
passing a value, 9
password routine, 21
PEEK, 25, 44
POKE, 25, 44
PRINT, 9, 25, 44
PRINT #, 21
PRINT FN, 11
PRINT PEEK, 26, 44
PRINT TAB (X, Y), 28, 45
PROC, 10
procedure, 10
program planning, 4, 5, 6
programmer's toolkits, 33
protecting your files, 21
random access memory, RAM 13, 34
READ/DATA, 18, 44
real numbers, 13
record, 20, 36
REM, 5, 25, 32, 33, 44
repeat/until loops, 16
reserved for use of operating system, 26
RESTORE, 22, 44
RETURN, 4, 7, 10, 23, 27, 44
RIGHT$, 44, 45
ROM, 26
saving memory space, 8, 12, 18, 20, 23, 33, 34
screen border subroutines, 29
screen memory, 26
scroll, 29
setting up variables and arrays, 8
Sinclair computers, 16, 18, 23
skeleton program, 7, 9
SPC, 28
Spectrum, 7, 21, 23, 27, 31, 43, 45, 46, 47
speed of a computer, 32
speeding up programs, 8, 32
SQR, 11
STEP, 44
STOP, 7, 16
string data, 11
string variables, 13, 23, 27
subroutines, 4, 5, 9, 32
  how the computer finds, 10
  library, 5
  order of, 9
subroutine trick, 10
subscript, 18, 19
TAB, 12, 31
terminating FOR/NEXT loops, 17
TRS-80, 45, 46, 47
two-dimensional arrays, 19, 20
unconditional loops, 16
unquoted string, 24
user RAM, 26
VAL, 27, 44
variable names, 12, 33
variables, 8, 9, 10, 13, 18, 23
variables tricks, 8, 12-13
VIC 20, 13, 15, 45
while loops, 17
word input, 27, 31
worker subroutines, 4, 5, 7, 9

Usborne Computer Books

"Highly recommended to anyone of any age." Computing Today
"Without question the best general introduction to computing I have ever seen." Personal Computer World
". . . perhaps the best introduction around . . . outstanding . . ." Educational Computing
"These books are outstanding . . . they make all other young people's computer books look meretricious." Times Educational Supplement

Guide to Computers
Understanding the Micro
Computer and Video Games
Computer Jargon
Computer Graphics
Inside the Chip
Computer Programming
Practise Your BASIC
Better BASIC
Machine Code for Beginners
Practical Things to do with a Microcomputer
Computer Spacegames
Computer Battlegames
Write Your Own Adventure Programs
Creepy Computer Games

New Titles

Programming Tricks & Skills Professional tips and tricks for better programming.

Experiments With Your Computer An exciting introduction to scientific investigation with a microcomputer.

Expanding Your Micro A detailed guide to add-ons and interfaces.

Write Your Own Fantasy Games A step-by-step guide to writing fantasy games, with program listing.

Weird Computer Games, Computer Spy Games Short listings to run on most main home computers.

Mystery of Silver Mountain, Island of Secrets Unique adventure game books containing full program listings.

ISBN 0-86020-793-5

ISBN 0 86020 793 5

£2.25